Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
ФАКУЛЬТЕТ КИБЕРНЕТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи
УДК 04.003

Клюев Дмитрий Владимирович

Разработка программно-аппаратной платформы с возможностью защищенного хранения информации и защищенными протоколами передачи информации.

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.01 Информатика и вычислительная техника

Выпускная квалификационная
работа защищена
«___»_____2016 г.
Оценка _____
Секретарь ГЭК _____

г. Москва
2016

Студент-дипломник: _____ / Клюев Д.В. /

Руководитель работы: _____ / Даньшин В.В. /

Рецензент: _____ / Веселов Д.А. /

Зав. кафедрой №12: _____ / Иванов М.А. /

АННОТАЦИЯ

Пояснительная записка состоит из трех глав.

В первой главе проведен обзор аппаратной части проекта. Рассмотрены спецификации микроконтроллеров Atmega32u4 и Atmega328P. Рассматриваются программаторы и способы загрузки программ в микроконтроллеры. Приводится описание таймеров счетчиков, универсального асинхронного приемопередатчика(USART), виды прерываний и их обработка, работа с аналогово-цифровым преобразователем (АЦП), принцип работы и особенности загрузчика и область памяти в которой он размещается. Приведено описание конфигурационных (fuse) и блокировочных(lock) битов (fuses). В конце главы описаны виды атак на микроконтроллеры и некоторые способы защиты.

Вторая глава посвящена разработке программной части. В данной главе приведены программные средства разработки. Рассмотрено создание программатора из платы Arduino UNO(МК Atmega328P). Приведены основные алгоритмы загрузчика и приложения на микроконтроллере, протокол взаимодействия ПК-сервера и микроконтроллера, реализованные функции на МК, коды ответов от ключа в зависимости от ситуации. В конце главы описан жизненный цикл ключа.

Третья глава содержит в себе описание среды разработки тестовой демонстрационной программы, которая симулирует защищаемое программное обеспечение.

Приложения содержат отлаженные листинги кода загрузчика, встраиваемого приложения для МК на языке avr C и код тестового демонстрационного проекта на языке processing(java).

Данная работа посвящена проектированию аппаратного ключа для защиты программного обеспечения. Одним из главных преимуществ данного проекта это возможность защитить любое программное обеспечение при наличии исходного кода.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Исходные данные:

1.1. Система предназначена для хранения критически важных данных в flash памяти микроконтроллера в защищенном виде.
1.2. Flash память должна хранить константы, алгоритм работы и другую ключевую информацию защищаемого ПО.
1.3. Система должна выполнять следующие функции:
  1.3.1. Обеспечивать защиту данных от несанкционированного доступа.
  1.3.2. Производить форматирование содержимого памяти микроконтроллера при несанкционированном доступе.
  1.3.3. Иметь защищенный протокол взаимодействия микроконтроллера и ПК.

1.4. **Содержание задания:**
1.5. Обзорная часть
  1.5.1. Изучение спецификации Arduino Pro Micro, МК Atmega32u4. Подготовка рабочего места. Установка необходимых драйверов для микроконтроллера (МК). Выбор среды разработки. Изучение возможностей среды. Проверка взаимодействия среды разработки с МК. Загрузка в МК тестовой прошивки. Создать ISP программатор из дополнительной платы Arduino. Загрузка прошивки через ISP программатор.
  1.5.2. Изучение спецификации загрузчика и адреса размещения загрузчика в flash памяти МК.
  1.5.3. Изучение основных векторов атак на МК. Изучение power glitch
  1.5.4. Изучение основных векторов атак на МК. Изучение. clock glitch
  1.5.5. Изучение переключения FUSE битов УФ излучением и применение микропроб для перепрограммирования регистров контроллера.
  1.5.6. Изучение физического избирательного повреждения топологии кристалла с целью выведения из строя отдельных его узлов.
  1.5.7. Изучение FUSE битов и принципов их работы.
  1.5.8. Изучение учета времени работы МК и хода времени на рабочей станции ПК.
1.6. Текст пояснительной записки
  1.6.1. Описание протокола взаимодействия микроконтроллера и ПК.
  1.6.2. Описание схемы алгоритма работы МК. Интерфейс, функции, описание автоматических тестов.
  1.6.3. Описание схемы алгоритма типовой программы для ПК.
  1.6.4. Описание подсистемы принятия решения о форматировании памяти.
  1.6.5. Схема алгоритма.
  1.6.6. Описание подсистемы подсчета реального времени.
  1.6.7. Описание подсистемы, реализующую функцию проверки и перезаписи состояния FUSE битов МК при его запуске.
1.7. Программа
  1.7.1. Реализация функции записи нулей в flash память МК в пределах заданного диапазона адресов.
  1.7.2. Реализация функции побайтового чтения содержимого Flash памяти с выполнением отправки значений в com порт.
  1.7.3. Реализация функции чтения/очистки eeprom памяти МК.
  1.7.4. Создание программы, реализующую функцию проверки и перезаписи состояния FUSE битов МК при его запуске. Данную функцию встроить в загрузчик.
  1.7.5. Реализовать подсистему подсчета реального времени в формате Unix timestamp. Реализация с кварцем. Реализация на внутреннем тактовом генераторе. Зависимость внутренней RC цепочки от времени. Команды коррекции времени. Ограничения на возможности корректирования времени.
  1.7.6. Реализация ГПСЧ на двенадцатиразрядном XOR.

1.7.7. Реализация полиморфного протокола общения с рабочей станцией пользователя на XOR с динамически изменяющимся ключом.
1.7.8. Разработать планировщик задач в очереди прерываний. Ограничение очереди 1 секунда. Ограничение времени прерывания.
1.7.9. Разработка подсистемы авторизации ПК-сервера на ключе МК.
1.7.10. Подсистема принятия решения о форматировании памяти. Подсистема ограничения количества вызова различных функций в единицу времени, подсистема ограничений на допустимый температурный диапазон для режимов работы МК, ограничения на последовательности команд (перебор пароля), ограничения на "накрутку" времени в меньшую сторону, получение команд с ПК на старт форматирования, форматирование чипа вследствие превышения порога изменения приема радиостанций ФМ диапазона.
1.7.11. Реализация таблиц хранения служебных переходов для алгоритмов программы ПК (кодируем последовательности вызова блоков алгоритма программы ПК в блоке памяти ПЗУ флэш МК).

1.8. Тесты
1.8.1. Разработать автоматический тест подсистемы подсчета реального времени.
1.8.2. Реализация тестовой программы для ПК на языке processing. Интерфейсы, функции, тесты
1.8.3. Разработка тестов подсистемы авторизации ПК-сервера на ключе МК.

3. Основная литература:

1. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. — М.: Издательский дом «Додэка-XXI», 2007. — 592 с: ил. (Серия «Программируемые системы»).

2. DATASHEET 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller. Atmel Corporation, 2015. — 438 с.. www.atmel.com
3. Мортон Дж. Микроконтроллеры AVR. Вводный курс. /Пер. с англ. – М.: Издательский дом «Додэка - XXI», 2006. – 272 с.: ил. (Серия «Мировая электроника»).

4. Отчетный материал:

пояснительная записка;

макетно-экспериментальная часть:

1. Листинги отлаженных программ.
2. Материалы тестирования и отладки.
3. Дистрибутив автоматизированной системы на CD.
4. Руководство пользователя / администратора.

Оглавление

Введение

По данным BSA (Business Software Alliance), доля нелегальных копий программных продуктов в России составляет 95 % — это дает основание относиться к компьютерному пиратству как к национальному бедствию.

С ростом объемов продаж отечественного софта стали расти и потери российских производителей программного обеспечения от пиратства. Соответственно, все более высокие требования стали предъявляться ими к качеству систем защиты от нелегального копирования.

Время тоже диктует свои коррективы. В середине 90-х годов программная защита (то есть защита, построенная на основе создания некопируемых дискет, “привязки” ПО к уникальным характеристикам компьютера и т.п.) начала сдавать свои позиции. Она морально устарела и уже в принципе не могла надежно защищать новейшие программные продукты. Вместо нее на российский рынок начала выходить программно-аппаратная защита — **электронные ключи.**

Электронные ключи — это небольшие электронные устройства, подсоединяемые, как правило, к параллельному порту ПК и служащие для защиты программ от их нелегального использования. За несколько лет своего существования эти устройства прошли большой путь развития. Современные электронные ключи — это устройства с энергонезависимой памятью (то есть памятью, не требующей для хранения записанных в нее данных никаких источников энергии), предназначенной для хранения данных по защите.

Сегодня электронные ключи — абсолютные лидеры отечественного рынка защиты. Универсальность применения, удобства для пользователей защищенных программ, высокий уровень совместимости с новейшим программным и аппаратным обеспечением — все это стало гарантом их постоянно растущей популярности и обеспечило им прекрасные перспективы.

Разумеется, электронные ключи имеют и качественно более высокий уровень защищенности от взлома, чем программная защита. Однако не следует думать, что они являются панацеей, обеспечивающей абсолютно “не снимаемую” защиту. Отечественные хакеры находятся в постоянном поиске новых методов взлома; и в плане взлома программно-аппаратной защиты ими уже накоплен богатый опыт. Сегодня в первую очередь страдают морально устаревшие электронные ключи, либо те, разработчики которые не учитывают особенностей хакерского движения в России. В результате среди имеющихся сегодня на рынке систем программно-аппаратной защиты есть и такие, взломать которые русским хакерам не сложнее, чем простую программную защиту.

Для выполнения задачи, поставленной в рамках дипломного проекта, необходимо:

- Собрать и обработать требования
- Спроектировать систему
- Разработать программную часть на микроконтроллере
- Разработать систему тестирования на ПК
- Исправить выявленные ошибки
- Создать документацию
- Внедрить разработанную систему

1.ОБЗОРНАЯ ЧАСТЬ

1.1. Платформа Arduino

Arduino[8] - аппаратная платформа-конструктор, предназначенная для удобной и быстрой разработки электронных устройств для новичков и профессионалов.

Сама платформа Arduino представляет из себя высокоуровневое программирование микроконтроллеров с помощью набора готовых библиотек, написанных на языке C, а сама среда разработки создана на языке Java. Платформа является очень популярной и пользуется огромным спросом во всем мире, благодаря своему удобство, простоте языка, а также открытой архитектуре и программному коду. Устройство имеет возможность программироваться с помощью USB без использования программаторов.

Печатные платы Arduino основаны на базе микроконтроллеров ATmega. Существуют следующие виды: Arduino Uno, Nano, Micro, Leonardo и другие. Различие этих плат состоит в том, что на каждой из них установлены разные микроконтроллеры, а также обладают дополнительным функционалом самой печатной платы (дополнительные пины и т.д.).

В проекте для прототипирования устройства использовались 2 вида плат: Arduino Uno и Arduino Micro.

1.1.1. Плата Arduino Uno

Arduino Uno (рис 1.1.) – плата с микроконтроллером ATmega328P, которая имеет 14 цифровых входных/выходных пинов (6 из которых могут использоваться в режиме широтно-импульсной модуляции выходного сигнала или PWM), 6 аналоговых входов, кварцевый кристалл с частотой 16 МГц, USB, разъем для подключения питания, ICSP и кнопку аппаратной перезагрузки. Технические характеристики платы приведены в таблице 1.1.

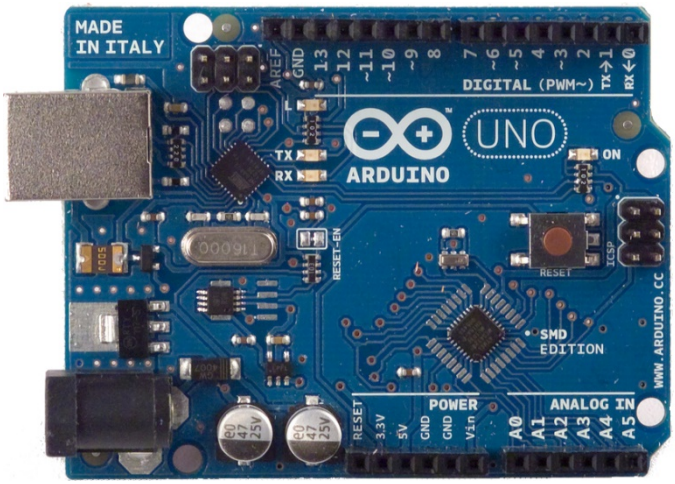


Рис. 1.1: Arduino Uno

Таблица 1.1: Технические характеристики платы Arduino Uno.

Микроконтроллер	Atmega328P
Рабочее напряжение	5 В
Входное напряжение питания (рекомендуемое)	7 - 12 В
Входное напряжение (пределы)	6 – 12 В
Цифровые I/O Пины	14 (6 из них поддерживают PWM)
PWM цифровые I/O пины	6
Аналоговые входные пины	6
Предельный постоянный ток I/O пинов	20 мА
Предельный ток для 3.3 В пинов	50 мА
Flash память	32 KB (Atmega328P), 0.5 KB занимает bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Частота кварца	16 МГц
Длина платы	68.6 мм
Ширина платы	53.4 мм
Вес платы	25

1.1.2. Плата Arduino Pro Micro

Arduino Pro Micro (рис.1.2.) – плата с микроконтроллером ATmega32u4, разработана в сотрудничестве с Adafruit. Плата содержит 20 цифровых входных/выходных пинов. Семь пинов могут использоваться в режиме широтно-импульсной модуляции выходного сигнала или PWM и двенадцать в качестве аналоговых входов. А также плата содержит кварцевый кристалл частотой 16 МГц, USB, разъем для подключения питания, ICSP. Технические характеристики платы приведены в таблице 1.2.

Рис. 1.2: Плата Arduino Pro Micro

Таблица 1.2: Технические характеристики платы Arduino Pro Micro

Микроконтроллер	Atmega32U4
Рабочее напряжение	5 В
Входное напряжение питания (рекомендуемое)	7 - 12 В
Входного напряжение (пределы)	6 – 12 В
Цифровые I/O Пины	20
PWM цифровые I/O пины	7
Аналоговые входные пины	12
Предельные постоянный ток I/O пинов	20 мА
Предельный ток для 3.3 В пинов	50 мА
Flash память	32 KB (Atmega32U4), 4KBзанимает bootloader

SRAM	2.5KB (ATmega32U4)
EEPROM	1 KB (ATmega32U4)
Частота кварца	16 МГц
Длина платы	48 мм
Ширина платы	18 мм
Вес платы	13 г

1.2. Микроконтроллеры ATmega328P и ATmega32U4

Микроконтроллеры принадлежат к семейству AVR (восьмибитных микроконтроллеров) фирмы Atmel, разрабатываемых с 1996 года. Микроконтроллеры AVR имеют гарвардскую архитектуру (программа и данные находятся в разных адресных пространствах) и систему команд, близкую к идеологии RISC. Процессор AVR имеет 32 8-битных регистра общего назначения, объединённых в регистровый файл.

Система команд достаточно развита, в разных моделях их число колеблется от 90 до 133 различных инструкций. Большинство команд занимает 16 бит (1 ячейку памяти), большая часть команд выполняется за 1 такт.

Команды можно разделить на несколько групп:

- Команды логических операций;
- Команды арифметических операций и команды сдвига;
- Команды операции с битами;
- Команды пересылки данных;
- Команды передачи управления;
- Команды управления системой;

Периферийные устройства управляются через адресное пространство данных.

AVR состоит из нескольких семейств микроконтроллеров:

- tinyAVR (ATtinyxxx);
- megaAVR (ATmegaxxx);
- XMEGA AVR (ATxmegaxxx);

Отличительными особенностями микроконтроллеров семейства ATmega являются:

- Флэш-память до 256 KB; SRAMдо 16 KB; EEPROM до 4KB;
- Число линий ввода-вывода 23-86 (общее число выводов 28-100);
- Аппаратный умножитель;
- Расширенная система команд и периферийных устройств;

1.2.1. Микроконтроллер Atmega328P

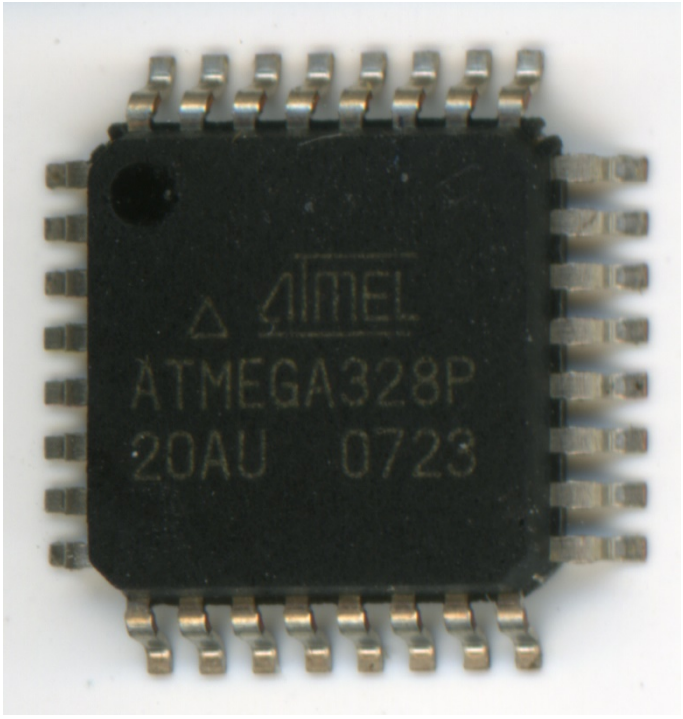


Рис. 1.3: Микроконтроллер ATmega328P

Таблица 1.3. Технические характеристики МК ATmega328P[1]

Общие	Высокая производительность с низким энергопотреблением Atmel AVR, семейство 8-битных микроконтроллеров
Продвинутая RISC архитектура	• 131 команда; большая часть выполняется за 1 такт • 32 8-битных регистра общего назначения; • Полностью статические операции; • до 20 MIPS с пропускной способностью 20 МГц; • Подстраиваемая область загрузки кода с независимыми битами блокировки;
Энергонезависимая память для хранения программы и данных	• 4/8/16/32 KB системная самопрограммируемая Flash; • 51 /1/2KBвнутренняя SRAM; • 256/512 Bytes/1KB встроенная EEPROM; • Число циклов перезаписи: 10 000 Flash/100 000 EEPROM; • Время хранения данных: 20 лет при 85°C/100 лет при 25°C; • Подстраиваемая область записи кода с независимыми запрещающими битами;

	• Возможность программировать запрещающие биты;
Поддержка библиотек AtmelQtouch	• Поддержка емкостных сенсорных кнопок; • QTouch и QMatrix; • Поддержка до 64 сенсорных каналов;
Периферийные особенности	• Два 8-разрядныых таймера/счетчика с независимыми пред делителями, режимами сравнения и режимами захвата; • Счетчик реального времени с независимым осциллятором; • 6 PWM каналов; • 8 каналов 10-битного ADCв TQFPи QFN/MLF; • 6 каналов 10-битного ADCв PDIP; • Программируемый последовательный интерфейс USART; • Master/Slave последовательный интерфейс SPI; • Байт-ориентированный 2-проводной последовательный интерфейс (I2C); • Аналоговый компаратор на чипе;
Специальные особенности микроконтроллера	• Power-on Reset и пограммируемый Brown-out Detection; • Внутренний калибровочный осциллятор; • Внешние и внутренние источники прерывания; • 6 режимов сна: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, Extended Standby;
I/O и корпуса	• 23 программируемых I\O линии; • 28-pinPDIP, 32-leadTQFP, 28-padQFN/MLFи 32-padQFN/MLF;
Рабочее напряжение	1.8 – 5.5 В
Температурный диапазон	От -40°C до 85°C
Скорость работы	0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5.V, 0 - 20MHz @ 4.5 - 5.5V
Требования к питанию при 1Мгц, 1.8 В, 25°C	• Активный режим: 0.2 мА • Power-down режим: 0.1 мкА • Power-savережим: 0.75 мкА (включая

	32 кГц RTC)

1.2.2. Микроконтроллер Atmega32U4

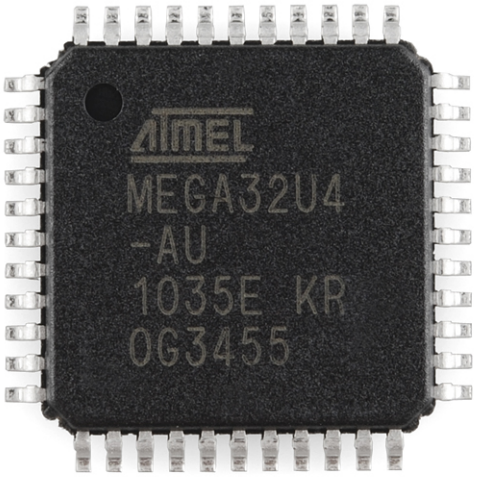


Рис. 1.4: Микроконтроллер Atmega32U4

Таблица 1.4: Технические характеристики МК Atmega32U4[2]

Общие	Высокая производительность с низким энергопотреблением Atmel AVR, семейство 8-битных микроконтроллеров
Продвинутая RISC архитектура	• 135 команд; большая часть выполняется за 1 такт • 32 8-битных регистра общего назначения; • Полностью статические операции; • до 16MIPS с пропускной способностью 16 МГц; • На чипе есть 2-тактный умножитель;
Энергонезависимая память для хранения программы и данных	• 16/32 KB системная самопрограммируемая Flash; • 1.25/2.5 KB внутренняя SRAM; • 512 Bytes/1KB встроенная EEPROM; • Число циклов перезаписи: 10 000 Flash/100 000 EEPROM; • Время хранения данных: 20 лет при 85°C/100 лет при 25°C; • Подстраиваемая область записи кода с независимыми запрещающими битами; • Возможность программировать запрещающие биты;

JTAG (IEEEstd. 1149.1 compilant) Interface	• Возможность периферийного сканирования согласно стандарту JTAG; • Обширная отладка на чипе; • Программирование FLASH, EEPROM, Fuses, LockBits через JTAG интерфейс;
USB2.0 Full-speed/Low-speed Device Module with interrupt in Transfer Completion	*за более подробной информацией обращаться в Datasheet производителя;
Периферийные особенности	• PLLна кристалле для USB и высокоскоростной таймер: с 32 до 96МГц; • Один 8-битный таймер/счетчик с независимым делителем и схемой сравнения; • Два 16-битных таймера/счетчика с независимыми делителями, схемами сравнения и захвата; • Один 10-битный высокоскоростной таймер/счетчик с PLL (64МГц) и схемой сравнения; • Четыре 8-битных PWM канала; • 4 PWMканала с программируемым разрешением от 2 до 16 Бит; • 6 PWM каналов для высокоскоростных операций с программируемым разрешением от 2 до 11 Бит; • Выходной компаратор; • 12-канальный и 10-битный ADC; • Программируемые последовательный интерфейс USARTс схемотехническим контролем потока; • Master/Slave SPI последовательный интерфейс; • Байт ориентированный 2-проводной последовательный интерфейс; • Программируемый сторожевой таймер с независимым осциллятором на чипе; • Аналоговый компаратор на кристалле; • Температурный сенсор на кристалле;
Специальные особенности микроконтроллера	• Power-on Reset и программируемый Brown-out Detection; • Внутренний 8 МГц калибровочный осциллятор;

	• Встроенный делитель счетчика и переключение на ходу между встроенной RC цепочкой и внешним осциллятором; • Внешние и внутренние источники прерываний; • 6 режимов сна: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, Extended Standby;
I/O и корпуса	• Все I/O комбинированы: КМОП логика – выход; LVTTL–вход; • 26 программируемых I/O линий; • 44-lead TQFP, 10x10mm; • 44-lead QFN, 7x7mm;
Рабочее напряжение	2.7 – 5.5 В
Температурный диапазон	От -40°C до 85°C
Скорость работы	• 8 МГц при 2.7В – промышленный режим; • 16 МГц при 4.5 В – промышленный режим;

1.3. Программатор

Программатор - аппаратно-программное устройство записи (чтения) информации в ПЗУ микроконтроллера.

По подключению микросхемы различают:

- Параллельные;
- Внутрисхемные (ISP);

Параллельное программирование обычно применяется на поточном производстве при массовой прошивке чипов в программаторе перед запайкой в устройство. Этот вид программаторов во много раз быстрее последовательного ISP, но требует подачи на ножку Reset напряжения в 12 вольт, а также необходимо 8 линий для прошивки, не включая линии управления. Так как данный вид программаторов использует высокое напряжение на линии для прошивки, то с помощью него можно «сбрасывать» Fuse-биты.

Примерами параллельных программаторов могут быть:

- HVProg;
- Paraprog;
- DerHammer;
- TurboProg;
- BeeProg;
- ChipPog++;
- Fiton;

1.3.1. Последовательный периферийный интерфейс SPI

Интерфейс SPI (Serial Peripheral Interface — последовательный периферийный интерфейс) обеспечивает высокоскоростную передачу информации в синхронном режиме между ведущим и ведомыми устройствами. На рис. 1.5. показано включение ведущего и ведомого устройств с использованием SPI.

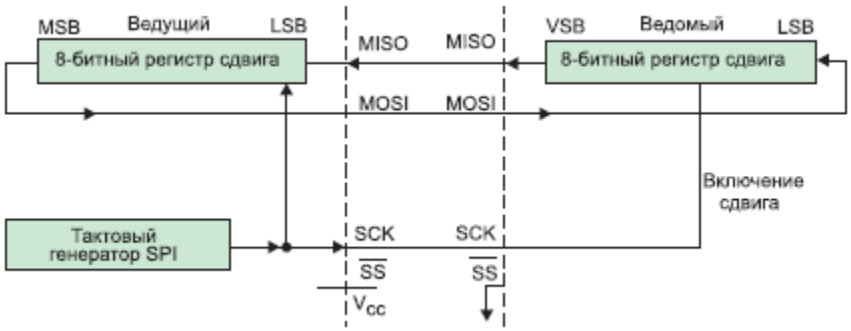


Рис. 1.5: Подключение ведущего и ведомого устройства через SPI

Внутрисхемный метод – метод, при котором микроконтроллер находится в схеме целевого устройства. Программатору выделяется несколько выводов контроллера. К выводам подключается прошивающий шнур, через который и происходит загрузка прошивки (преобразованной в бинарный вид). У AVR прошивка загружается по интерфейсу SPI (рис. 1.6.) и для работы программатора необходимы четыре линии и питание.

Линии SPI:

- MISO – Master Input/Slave Output;
- MOSI – Master Output/Slave Input;
- SCK – тактовые импульсы интерфейса SPI;
- RST – Reset, данным сигналом программатор вводит контроллер в режим программирования;
- GND – земля;
- V – питание;

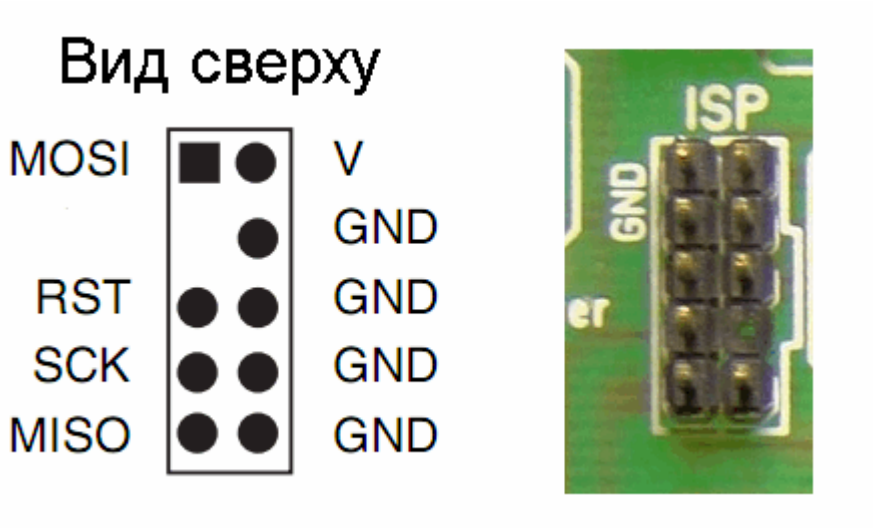


Рис. 1.6: Разъем SPI

Существует большое число программаторов, которые отличаются между собой скоростью работы и типу подключения к ПК (COM/LPT/USB).

Примерами программаторов могут быть:

1. Программатор Громова (работает через оболочку UniProf);
2. STK200 (работает через LPT порт, требует прямого доступа к порту со стороны операционной системы и наличие самого LPT);
3. FTBB-PROG;
4. USBASP;

1.3.2. Отладочный интерфейс JTAG

Данный интерфейс позволяет выполнять пошаговую отладку программы прямо на кристалле, но также позволяет загружать программу в микроконтроллер и выставлять конфигурационные Fuse-биты.

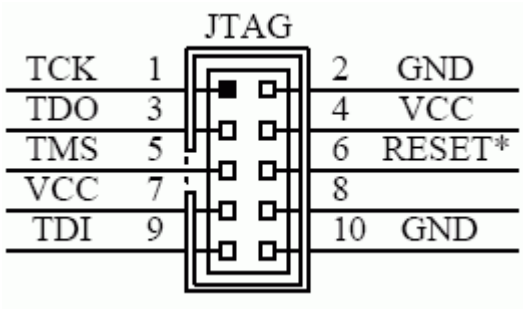


Рис. 1.7: Интерфейс JTAG

TDI – Test Data Input, вход последовательных данных периферийного сканирования, применяется для ввода данных и команд в микросхему по переднему фронту сигнала TCK;

TDO – Test Data Output, выход последовательных данных, применяется для вывода команд и данных из микросхемы по заднему фронту сигнала TCK;

TCK – Test Clock, тактирует работу встроенного автомата управления периферийным сканированием;

TMS – Test Mode Select, обеспечивает переход схемы в
и режим тестирования и переключение между разными режимами тестирования;

1.4. Организация памяти микроконтроллеров ATmega32u4 и ATmega328p

Перепрограммируемая флэш-память:

Флэш-память микроконтроллеров AVR организованная по принципу 2/4/8/16x16. В целях обеспечения защиты прошивки, память разбита на две части: область загрузчика и область приложения.

Флэш-память имеет 10 000 циклов записи/ стирания.

Таблица 1.5: Описание Флэш-памяти.

	ATmega328p	ATmega32u4
Объем	32 КБ	32 КБ
Начальный адрес	0x0000	0x0000

Конечный адрес	0x3FFF (word)	0x3FFF(word)
Область приложения	0x0000 - 0x37FF	0x0000 - 0x37FF
Область загрузчика	0x3800 - 0x3FFF	0x3800 - 0x3FFF

Таблица 1.6: SRAM память и адреса регистров.

	ATmega328p	ATmega32u4
32 Registers	0x0000 - 0x001F	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF	0x0060 - 0x00FF
Объем	2 КБ	2.5 КБ
Internal SRAM	0x0010 - 0x08FF	0x0100 - 0x0AFF

Таблица 1.7: Энергонезависимая память (EEPROM)

	ATmega328p	ATmega32u4
Объем	1 КБ	1 КБ

Энергонезависимая память EEPROM имеет 100 000 циклов записи/стирания.

1.5. Таймер-счетчик

Таймер-счетчик – один из самых используемых ресурсов микроконтроллера. Он служит счетчиком заданных временных интервалов, кроме того, он может выполнять функции формирования ШИМ сигналов, подсчет длительности и количества входящих импульсов.

Рассмотрим устройство таймер T0. Он состоит из 3 регистров:

- счетный регистр TCNT0 (рис. 1.8.);
- регистр сравнения OCR0;
- конфигурационный регистр TCCR0;

Существуют также три регистра, которые относятся ко всем таймерам:

- конфигурационный регистр TIMSK;
- регистр статуса TIFR;
- регистр специальных функций SFIOR;

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис.1.8: Регистр TCNT0

8-ми разрядный счетный регистр. При работе таймера по каждому импульсу тактового сигнала значение TCNT0 изменяется на единицу. В зависимости от режима работы таймера, значения регистр увеличивается, либо уменьшается.

Этот регистр можно считывать, а также записывать. При работе таймера менять содержимое не рекомендуется, т.к. это блокирует схему сравнения на один такт.

Регистр OCR0:

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.9: Регистр OCR0

8-ми разрядный регистр сравнения. Его значение постоянно сравнивается со значением регистра TCNT0. В случае совпадения – таймер может выполнять какие-то действия, например, вызывать прерывания, менять состояние вывода OC0 и т.д. в зависимости от режима работы.

Значение этого регистра можно считывать и записывать.

Регистр TCCR0:

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.10: Регистр TCCR0

Это конфигурационный регистр таймера-счетчика T0, он определяет источник тактирования таймера, коэффициент пред делителя, режим работы таймера-счетчика T0 и поведение вывода OC0.

Биты CS02, CS01, CS01 (Clock Select) – определяют источник тактовой частоты для таймера T0 и задают коэффициент пред делителя.

Таблица 1.8: состояния битов Clock Select

CS02	CS01	CS00	Описание
0	0	0	Источника тактирования нет. Таймер остановлен.
0	0	1	Тактовая частота МК
0	1	0	Тактовая частота МК/8
0	1	1	Тактовая частота МК/64
1	0	0	Тактовая частота МК/256
1	0	1	Тактовая частота МК/1024
1	1	0	Внешний источник на выводе T0. Срабатывание по заднему фронту
1	1	1	Внешний источник на выводе T0. Срабатывание по переднему фронту

Биты **WGM10, WGM00 (Wave Generator Mode)** - определяют режим работы таймера-счетчика T0. Существуют следующие режимы работы: нормальный режим (normal), сброс таймера при совпадении (CTC), и два режима широтно-импульсной модуляции (Fast PWM и Phase Correct PWM).

Таблица 1.9: состояния битов WGM

WGM01	WGM00	Режим работы таймера/счетчика
0	0	Normal
0	1	PWM, Phase Correct
1	0	CTC
1	1	Fast PWM

Биты **COM01, COM00 (Compare Match Output Mode)** - определяют поведение вывода OC0. Если хоть один из этих битов установлен в 1, то вывод OC0 перестает функционировать как обычный вывод общего назначения и подключается к схеме сравнения таймера счетчика T0. Однако при этом он должен быть еще настроен как выход.

Поведение вывода OC0 зависит от режима работы таймера-счетчика T0. В режимах normal и CTC вывод OC0 ведет себя одинаково, а вот в режимах широтно-импульсной модуляции его поведение отличается.

Бит **FOC0 (Force Output Compare)** - предназначен для принудительного изменения состояния вывода OC0. Он работает только для режимов Normal и CTC. При установке бита FOC0 в единицу состояние вывода меняется соответственно значениям битов COM01, COM00. Бит FOC0 не вызывает прерывания и не сбрасывает таймер в режиме CTC.

Регистр TIMSK (**Timer/Counter Interrupt Mask Register**):

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.11: Регистр TIMSK

Общий регистр для всех трех таймеров, он содержит флаги разрешения прерываний. Таймер T0 может вызывать прерывания при переполнении счетного регистра TCNT0 и при совпадении счетного регистра с регистром сравнения OCR0. Соответственно для таймера T0 в регистре TIMSK зарезервированы два бита - это TOIE0 и OCIE0. Остальные биты относятся к другим таймерам.

TOIE0 - 0-е значение бита запрещает прерывание по событию переполнение, 1 - разрешает.
OCIE0 - 0-е значение запрещает прерывания по событию совпадение, 1 разрешает.

Естественно прерывания будут вызываться, только если установлен бит глобального разрешения прерываний - бит I регистра SREG.

Регистр TIFR (Timer/Counter0 Interrupt Flag Register):

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.12: Регистр **TIFR**

Общий для всех трех таймеров-счетчиков регистр. Содержит статусные флаги, которые устанавливаются при возникновении событий. Для таймера T0 - это переполнение счетного регистра TCNT0 и совпадение счетного регистра с регистром сравнения OCR0. Если в эти моменты в регистре TIMSK разрешены прерывания и установлен бит I регистра SREG, то микроконтроллер вызовет соответствующий обработчик. Флаги автоматически очищаются при запуске обработчика прерывания. Также это можно сделать программно, записав 1 в соответствующий флаг.

TOV0 - устанавливается в 1 при переполнении счетного регистра.
OCF0 - устанавливается в 1 при совпадении счетного регистра с регистром сравнения.

Таблица 1.10: Таймеры счетчики микроконтроллеров ATmega328P и ATmega32u4

	ATmega328P	ATmega32u4
8 битные	TCNT0, TCNT2	TCNT0
16 битные	TCNT1H, TCNT1L	1. TCNT1H, TCNT1L 2. TCNT3H, TCNT3L
Высокоскоростные счетчики	-	TCNT4

1.6. Универсальный асинхронный приёмопередатчик (USART)

USART – универсальный синхронно/асинхронный приемопередатчик, который преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по цифровой линии другому аналогичному устройству.

Данные по интерфейсу в равные промежутки времени передаются по одному биту. Временной промежуток задается скоростью USART. Принято указывать его в бодах (число бит в секунду). Существует стандартный набор скоростей: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод.

При передаче данные принято разбивать на пакеты, которые начинаются и заканчиваются синхронизирующими метками (стартовым и стоповым битами, соответственно). Обычно пакет имеет размер 8 бит, хотя существуют и реализации по 5,6,7 или 9 бит.

Пассивное состояние линии данных (данных нет) описывается логической единицей на входах и выходах, поэтому стартовый бит – логический ноль.

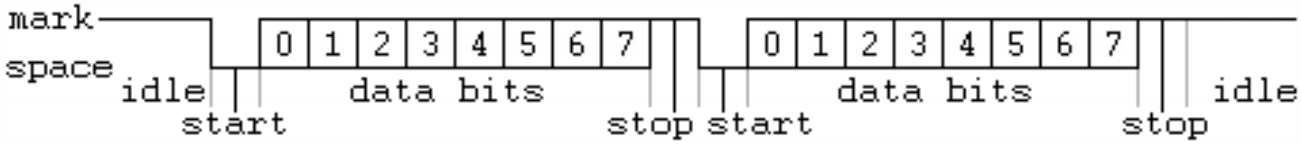


Рис. 1.13: Передача данных по USART

Регистры USART

Регистр UBRR:

Регистр, содержащий в себе значение скорости работы интерфейса. Для этого в документации к микроконтроллерам приводятся следующие 3 формулы для разных режимов работы:

- Асинхронный нормальный режим (U2X = 0):

$$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}+1)} \quad \left| \quad \boldsymbol{UBRR} = \frac{f_{OSC}}{16BAUD} - 1 \right.$$

- Асинхронный удвоенная скорость (U2X = 1):

$$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}+1)} \quad \left| \quad \boldsymbol{UBRR} = \frac{f_{OSC}}{8BAUD} - 1 \right.$$

- Синхронный режим:

$$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}+1)} \quad \left| \quad \boldsymbol{UBRR} = \frac{f_{OSC}}{2BAUD} - 1 \right.$$

Полученное значение записывают в 8-ми битные регистры UBRRL и UBRRH (младшие и старшие 8 бит скорости).

UCSRnA, UCSRnB, UCRSnC – регистры конфигурации порта:

Таблица 1.11: Регистр UCSRnA

7	**RXC** (Receive Comlete)	Бит завершения приёма. Автоматически устанавливается при приеме данных, сбрасывается при чтении.
6	**TXC** (Transmit Comlete) *(только запись)*	Бит завершения передачи. Автоматически устанавливается после отправки кадра данных, если следующих для отправки данных нет.
5	**UDRE** (UDR Empty)	Бит отсутствия данных для отправки. Установлен, если данных для отправки нет.

4	**FE** (Frame Error)	Ошибка кадра. 1-если ошибка в формате принятого пакета.
3	**DOR** (Data OverRun)	Переполнение буфера. 1-если буфер переполнен.
2	**PE** (Parity Error)	Ошибка чётности. 1-если была ошибка.
1	**U2X** (Double speed) *(только запись)*	Удвоение скорости
0	**MPCM** (Multi-Processor Communication Mode) *(только запись)*	Работа в многопроцессорном режиме.

Таблица 1.12: Регистр UCSRnB

7	**RXCIE** (Receive Comlete Interrupt Enable)	Бит прерывания приёма данных
6	**TXCIE** (Transmit Comlete Interrupt Enable)	Бит прерывания завершения передачи
5	**UDRIE** (UDR Empty Interrupt Enable)	Бит прерывания отсутствия данных для передачи
4	**RXEN** (Receiver Enable)	Разрешение приёма данных
3	**TXEN** (Transmitter Enable)	Разрешение передачи данных
2	**UCSZ2** (USART Character SiZe 2)	См. ниже UCSZ1 и UCSZ0 в UCSRC
1	**RXB8** (Receive Data Bit 8) *(только чтение)*	9 бит принятых данных, в многопроцессорном режиме

0	**TXB8** (Transmit Data Bit 8) *(только запись)*	9 бит отправленных данных, в многопроцессорном режиме

Таблица 1.13: Регистр UCSRnC

7	**URSEL** (USART Register SELect)	Всегда равен 1. Служит для выбора регистра UCSRC, иначе UBRRH.
6	**UMSEL** (USART Mode SELect)	Режим работы. 1 — синхронный, 0 — асинхронный
5-4	**UPM1-UPM0** (USART Parity Mode)	Установка режима чётности. UPM1:0=1:1 – нечётный UPM1:0=1:0 – чётный UPM1:0=0:0 – нет контроля
3	**USBS** (USART Stop Bit Select)	Количество стоп-битов. 0-1 стоп бит, 1-2 стоп бита.
2-1	**UCSZ1-UCSZ0** (USART Character SiZe)	UCSZ2-0 – устанавливают размер кадра, см таблица ниже.
0	**UCPOL** (USART Clock POLarity)	Используется в синхронном режиме. 0-данные отправляются по переднему фронту 1-по заднему фронту
7	**URSEL** (USART Register SELect)	Всегда равен 1. Служит для выбора регистра UCSRC, иначе UBRRH.
6	**UMSEL** (USART Mode SELect)	Режим работы. 1 — синхронный, 0 — асинхронный

1.7. Прерывания

Прерывание[4][2] – сигнал, сообщающий процессору о наступлении какого-либо события, при этом приводящее к приостановке текущего выполнения команд, а управление передается процедуре обработки прерывания данного события, по завершении которого происходит возврат на туже позицию.

Процедура обработки прерывания – процедура, которую следует выполнить при возникновении определенного события.

Свойства прерываний:

- У каждого устройства существует как минимум один источник прерываний;
- За каждым прерыванием закреплена ссылка на процедуру обработки прерывания (вектор прерывания). Все вектора располагаются в самом начале памяти программы, формируя таблицу векторов прерываний;
- Каждому прерыванию соответствует бит разрешающий прерывание;

1.7.1. Регистры прерывания

SREG – 8-ми битный регистр состояний прерываний.

SREG – AVR Status Register

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.14: Регистр SREG

- **Bit 7 - I: Global Interrupt Enable - Разрешение глобального прерывания.**
  Бит разрешения глобального прерывания для разрешения прерывания должен быть установлен в состояние 1. Управление разрешением конкретного прерывания выполняется регистрами маски прерывания GIMSK и TIMSK. Если бит глобального прерывания очищен (в состоянии 0), то ни одно из разрешений конкретных прерываний, установленных в регистрах GIMSK и TIMSK, не действует. Бит I аппаратно очищается после прерывания и устанавливается для последующего разрешения глобального прерывания командой RETI.
- **Bit 6 - T: Bit Copy Storage - Бит сохранения копии.**
  Команды копирования бита BLD (Bit LoaD) и BST (Bit STore) используют бит T как бит источник и бит назначения при операциях с битами. Командой BST бит регистра регистрового файла копируется в бит T, командой BLD бит T копируется в регистр регистрового файла.
- **Bit 5 - H: Half Carry Flag - Флаг полупереноса**
  Флаг полупереноса указывает на полуперенос в ряде арифметических операций. Более подробная информация приведена в описании системы команд.
- **Bit 4 - S: Sign Bit, S = N V - Бит знака**
  Бит S всегда находится в состоянии, определяемом логическим исключающим ИЛИ (exclusive OR) между флагом отрицательного значения N и дополнением до двух флага переполнения V. Более подробная информация приведена в описании системы команд.
- **Bit 3 - V: Two's Complement Overflow Flag - Дополнение до двух флага переполнения**
  Дополнение до двух флага V поддерживает арифметику дополнения до двух. Более подробная информация приведена в описании системы команд.
- **Bit 2 - N: Negative Flag - Флаг отрицательного значения**
  Флаг отрицательного значения N указывает на отрицательный результат ряда арифметических и логических операций. Более подробная информация приведена в описании системы команд.
- **Bit 1 - Z: Zero Flag - Флаг нулевого значения**
  Флаг нулевого значения Z указывает на нулевой результат ряда арифметических и логических операций. Более подробная информация приведена в описании системы команд.
- **Bit 0 - C: Carry Flag - Флаг переноса**
  Флаг переноса C указывает на перенос в арифметических и логических операциях. Более подробная информация приведена в описании системы команд.

1.7.2. Таблица векторов прерываний

У каждого прерывания есть строго определенный приоритет. Приоритет прерывания зависит от его расположения в “таблице векторов прерываний” (таблица 16). Чем меньше номер вектора в таблице, тем выше приоритет прерывания. То есть, самый высокий приоритет имеет прерывание сброса (Reset interrupt), которое располагается первой в таблице, а соответственно и в памяти программ.

Таблица 1.14: таблица векторов прерываний.

Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

1.7.3. Не строгий алгоритм прерывания в микроконтроллерах

1. Произошло срабатывание прерывания (IRQ):

   Обработка прерывания происходит в соответствии с приоритетом по таблице прерываний.

2. Проверка:

   Если бит активации данного прерывания установлен (Interrupt enable bit), а также бит I (бит глобального разрешения прерываний) регистра состояния процессора (SREG) установлен, то процессор начинает подготовку процедуры обработки прерывания, при этом бит глобального разрешения прерываний (бит I регистра SREG) сбрасывается, запрещая,

таким образом, все остальные прерывания. Это происходит для того чтобы никакое другое событие не смогло прервать обработку текущего прерывания.

3. Подготовка:

   Процессор завершает выполнение текущей ассемблерной команды, после чего помещает адрес следующей команды в стек (PC->STACK). Далее процессор проверяет, какой источник прерывания подал “запрос на обработку прерывания” (IRQ), после чего воспользовавшись вектором данного источника (ссылка) из таблицы векторов (который закреплен за каждым источником прерывания), переходит в процедуру обработки прерывания (инструкция JMP). Процессор тратит минимум 4 такта (в зависимости от момента появления запроса и длительность исполнения текущей инструкции). Это очень хорошее время реакции на IRQ, по сравнению с микроконтроллерами других производителей.
4. Выполнение тела ISR (прерывания);
5. Возврат управления:

   После того как выполнение процедуры обработки прерывания завершено, процессор извлекает адрес возврата из прерывания, который он сохранил в стеке (STACK->PC), прибавляет к указателю стека значение 2, что соответствует уменьшению стека на два байта, которые раньше занимал адрес возврата из прерывания. Далее процессор устанавливает бит I (бит глобального разрешения прерываний) регистра состояния процессора (SREG). На это процессор тратит ровно 4 такта. После этого выполнение программы продолжается ровно с того места где она была прервана.

1.8. Аналогово-цифровой преобразователь (АЦП)

Микроконтроллеры ATmega32u4, ATmega328p содержат в себе 10-битовый АЦП[3], вход, которого может быть соединен с выводами портов. АЦП необходимо опорное напряжение для целей сравнения с входным (если измеряемое равно опорному, то получаем максимальный код в двоичном виде). Опорное напряжение подается на вывод ADRef или может использоваться внутренний генератор с фиксированным напряжением 2,56 В. Полученный результат можно представить в виде:

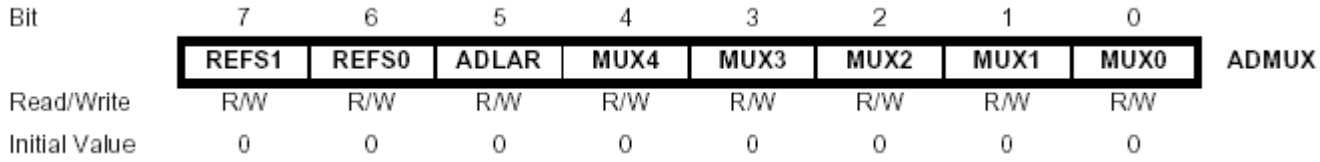
$$ADC = \frac{V_{in} * 1024}{V_{ref}}$$

АЦП включается установкой бита ADEN в регистре ADCSRA. После преобразования, 10 битный результат оказывается в 8-битных регистрах ADCL и ADCH. По умолчанию, младший бит результата находится справа (то есть в bit 0 регистра ADCL, так называемое правое ориентирование). Но порядок следования битов на левое ориентирование можно сменить, установив бит ADLAR в регистре ADMUX. Это удобно, если требуется получить 8-битовый результат. В таком случае требуется прочитать только регистр ADCH. В противном случае, нужно сначала прочитать регистр ADCL первым, а ADCH вторым, чтобы быть уверенным в том, что чтение этих двух регистров относится к результату одного преобразования.

Одиночное преобразование может быть вызвано записью бита ADSC в регистр ADCSRA. Этот бит остаётся установленным всё время, занимаемое преобразованием. Когда преобразование закончено, бит автоматически устанавливается в 0. Можно также начинать преобразования по событиям из разных источников. Модуль АЦП также может работать в режиме "свободного полёта". В таком случае АЦП постоянно производит преобразование и обновляет регистры ADCH и ADCL новыми значениями.

Для выполнения преобразования модулю АЦП необходима тактовая частота. Чем выше эта частота, тем быстрее будет происходить преобразование (оно, обычно, занимает 13 тактов, первое преобразование занимает 25 тактов). Но чем выше частота (и выше скорость преобразования), тем менее точным получается результат. Для получения максимально точного результата, модуль АЦП

должен тактироваться частотой в пределах от 50 до 200 КГц. Если необходим результат с точностью менее 10 бит, то можно использовать частоту больше 200 КГц. Модуль АЦП содержит делитель частоты, чтобы получать нужную тактовую частоту для преобразования из частоты процессора.

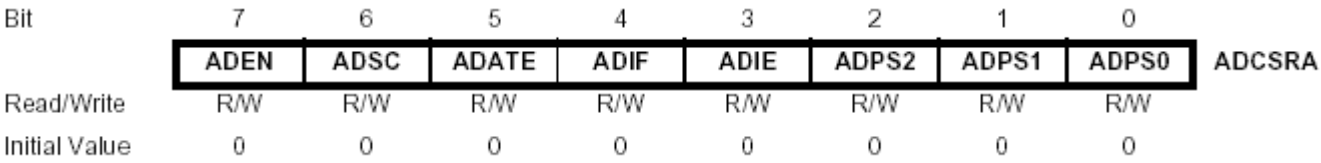


Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.15: Регистр ADMUX

Регистр ADMUX задаёт входной контакт порта A для подключения АЦП, ориентирование результата и выбор опорной частоты. Если установлен бит ADLAR, то результат лево ориентирован. Опорная частота от внутреннего генератора задаётся выставленными в 1 битами REFS1 и REFS0. Если оба бита, сброшены, то опорная частота берётся от контакта AREF. В случае, если REFS1=0 а REFS0=1, опорная частота берётся от AVCC с внешним конденсатором, подключенным к AREF.

Регистр контроля и статуса АЦП:



Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рис. 1.16: Регистр ADSRA

- Бит ADEN = 1 включает модуль АЦП;
- Бит ADSC = 1 запускает цикл преобразования;
- ADIF - флаг прерывания АЦП;
  Этот бит устанавливается в 1 когда АЦП завершено преобразование и в регистрах ADCL и ADCH находятся актуальные данные. Этот флаг устанавливается даже в том случае, если прерывания запрещены.
- ADIE = 1 - прерывания разрешены глобально, то при окончании преобразования будет выполнен переход по вектору прерывания от АЦП.

1.9. Загрузчик

Загрузчик (bootloader) - программа, расположенная в специальной области микроконтроллера и чаще всего выполняющая следующие функции: инициализация периферийных устройств, инициализация устройств на плате, разрешение на загрузку программ пользователем, загрузка выбранных программ и другие функции. Фактически загрузчик может выполнять совершенно любые функции, как и любая программа, его ключевая особенность:

1. Организация SPM режима для прошивки по ISP;
2. Загрузчик расположен в отдельной секции памяти;
3. Имеет повышенные права управления памятью;
4. Может запускаться раньше основного приложения;

1.9.1. Принцип работы загрузчика

При загрузке контроллера на схемотехническом уровне происходит прыжок с адреса памяти 0x00 на стартовый адрес загрузчика, таким образом, ему передается управление. Происходит проверка для условия запуска. Само условие может быть любым, обычно это или наличие спец байта в регистре интерфейса, либо наличие нужного логического уровня на выбранной ножке контроллера, сигнализирующий о том, что происходит обращение к области загрузчика. Если условие выполнено, то загрузчик может принять прошивку по, например, USB интерфейсу и самостоятельно прошить её во flash память, или же наоборот, считать прошивку из памяти и передать её по интерфейсу, считать или записать в EEPROM и т.д. (совершенно любые действия). Если условие не выполнено, то загрузчик завершает свою работу и происходит прыжок на начальный адрес и загрузку основной программы микроконтроллера.

1.9.2. Особенности области памяти загрузчика

Флэш-память AVR делится на 2 части: область приложений и область загрузчика (BLS). Код загрузчика хранится в области BLS (Boot Loader Section) флэш-памяти микроконтроллера, которая имеет полный доступ к микроконтроллеру.

Любой код, который выполняется из области загрузчика, может использовать самопрограммируемый режим (SPM). Данный режим позволяет записывать/считывать флэш-память микроконтроллера, включая саму секцию BLS, откуда запущен сам код. Такая особенность может быть использована для загрузки любого бинарного кода во флэш-память, а также запустить его. Требуемая инициализация периферийного устройства интерфейса USART или USB для работы с ПК, битов портов, инициализация подключенных устройств, например, LCD Дисплея и т.д. может быть выполнена из области загрузчика перед загрузкой основного приложения.

1.10. Виды атак на микроконтроллеры

1.10.1. Общие сведения

- Микропроба[6] (микрозондирование) используется, чтобы получить прямой доступ к поверхности чипа, таким образом можно рассмотреть, манипулировать и взаимодействовать с интегральной схемой;
- Реверс-инжиниринг используется для установления внутренней структуры полупроводникового чипа и запоминания или эмулирования его функционала. Данные методы использует такие же технологии, которые используются на предприятиях, изготавливающих полупроводниковые приборы, поэтому они дают такие возможности и атакующему;
- Атаки с помощью приложений (софта) используют обычные интерфейсы обмена данными процессора и используют уязвимости в безопасности протоколов, криптографических алгоритмов или их в их реализации;
- Прослушивание позволяет атакующему наблюдать в условиях высокого временного разрешения за аналоговыми характеристиками питания, взаимодействием интерфейсов и любой электромагнитной радиацией, излучаемой процессором во время выполнения обычных операций;
- Создание ошибок использует отклоняющиеся от нормы условия окружения для создания сбоев в процессоре, что обеспечивает дополнительный доступ.

Все методы микропроб или реверс инжиниринговые методы – инвазивные атаки. На них тратят часы или недели в специализированной лаборатории и в процессе разрушается корпус.

1.10.2. Сценарии атак

1. Клонирование – один из наиболее распространенных целей атак. Обычно для клонирования используется реверс инжиниринг;
2. Пиратство;
3. Кража сервиса услуг;
4. Отказ сервиса;

1.10.3. Неинвазивные атаки

1. Атаки с низким напряжением[5][6][7] и высоким напряжением могут выключить защиту схемы или заставить процессор выполнить ошибочную операцию. Именно поэтому в некоторых процессорах предусмотрены защитные схемы определения напряжения, но такие схемы не реагируют на быстрые импульсные помехи. Импульсные помехи питания и часов в некоторых случаях влияют не декодирование и выполнение отдельных инструкций.
2. Анализ тока: можно измерять (с АЦП, шин данных) флуктуации тока, потребляемые устройством.
3. Измерение остаточной намагниченности данных: исходим из того, что данные после отключения питания не мгновенно удаляются из памяти. Например, SRAM сохраняет ключи на длительные промежутки времени, которые могут использоваться после следующей подачи питания. Другой способ – «заморозить» память под действием низких температур. В таких случаях SRAM сохраняет информацию на время, достаточное для того, чтобы получить доступ к памяти чипа и считать содержимое.
4. Изучение и исследование сигналов интерфейсов и протоколов доступа: в некоторых микроконтроллерах есть тестовые заводские интерфейсы. Попытка применить различные напряжения и логические уровни к ножкам, может привести к включению тестового режима микроконтроллера.

1.10.4. Инвазивные атаки

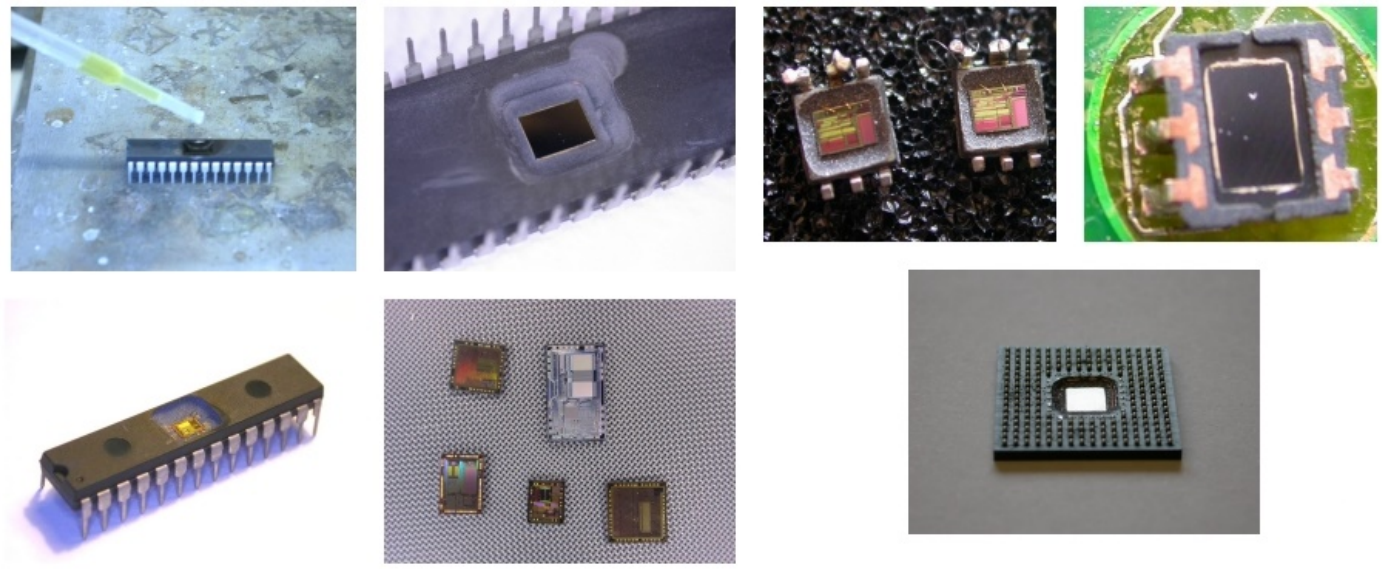


Рис. 1.17: Иллюстрация инвазивных[7] методов атак на микроконтроллеры

1. Малобюджетные злоумышленники:

Могут использовать оборудования для проведения микрозондирования (вскрывают чип), удалить пассивационный слой (обычно это оксид кремния или нитрид). Обычно (за исключением ROM) память считывают с шины памяти.

2. Некоторые способы реверс инжиниринга:

    1) Создаем карту процессора (с помощью CMOS VLSI техники или архитектуры микроконтроллера), после этого становится возможным получать доступ к каждому транзистору и взаимодействовать с ними через подложку с помощью FIB редактирования подходящего отверстия.
    2) УФ лучи используются для сброса защитных фьюз-байтов EEPROM и OTP микроконтроллеров (старый метод).
    3) Используются различного рода микроскопы, лазерное сканирование и термоделирование.
    4) Проводим атаку для возникновения ошибок, с помощью чего можно модифицировать SRAM и изменять состояния каждого транзистора в чипе.
    5) Подключение по JTAG с помощью проводов или осциллографа или логического анализатора считать все сигналы, проанализировать формы сигналов, повторять команды или проверять свои, изменять битовый поток.

1.10.5. Другие виды атак

- Ударные атаки (bumping attacks):

  Предназначены для извлечения данных из защищенной встроенной памяти, которая обычно хранит критические части алгоритма и криптографические ключи.

  Для предотвращения этого рода атак недостаточно просто ввести разрешение на чтение памяти. Обычно это приемлемо для относительно крупных блоков данных, что затрудняет полный перебор (brute force).

- Backdoor – дефект алгоритма, который намеренно встраивается в него разработчиком и позволяет получить тайный доступ к данным или управлению МК.

- Необходимо полностью заполнять таблицу пересылок + “впритык” указывать адрес загрузчика, либо заполнять её каким-либо мусором, что бы предотвратить атаку с внедренной прошивки в память;

- Сбой защиты при ошибке на одном транзисторе (полуинвазивный оптический метод - сбивание с помощью лазера)

- Временные атаки:

  Некоторые защитные операции могут занимать различное время проверки в зависимости от значения входных данных и секретного ключа. Тщательное таймирование и анализ может помочь извлечь секретный системный ключ. Общая идея защиты: смешивать входные данные со случайным числом (делаем неочевидные входные данные). Данная атака применима к микроконтроллерам, чья защита основана на пароле или системах с картами доступа или же ключах с фиксированным серийным номером. Частая ошибка – проверка по базе данных. Часто выполняют побитовую проверку, вместо всего ключа в целом, а также осуществляют остановку и переход к следующим входным данным, как только пришел неверный байт, вместо прохода до конца неверного массива данных. Поэтому атакующий может легко измерить время между вхождениями ключей и вычислить, количество совпадений, таким образом, он найдет, по крайней мере, один из используемых ключей. Для защиты также нужно внимательно считать количество циклов CPU за время сравнения/проверки пароля (они должны быть всегда одинаковы для верного и неверного пароля).

- Brute force attacks:

Длины пароля в 32 байта достаточно, чтобы выполнить прямую brute force атаку. Но пароль расположен по тем же адресами памяти, что и вектора прерывания CPU, что сужает область поиска, т.к. вектор всегда указывает на четные адреса памяти, а также при обновлении ПО изменяется только малая часть пароля, потому что большая часть прерываний подпрограмм остается по теми же адресами. Как результат, если атакующий знает один из предыдущих паролей, он может очень просто выполнить систематический поиск и найти верный пароль за разумное время.

Атаку полным перебором или атаку черного ящика также можно выполнять на оборудовании, внедренном в ASIC или CPLD. Выполняет все возможный перебор логических комбинаций на входе устройства для изучения всех выходов.

Другая возможная атака полным перебором (работает для многих чипов) – применение высокого входного логического напряжения сигнала (обычно удвоенный напряжение питания) к контактам чипа, чтобы выявить, выполняет хотя бы один из них транзакцию в фабричном тестовом или программном режиме.

Атака на коммуникационный протокол устройства для поиска всех скрытых функций, оставленных разработчиками для нужд теста или обновления.

Подчеркнем, что предприятия очень часто оставляют защищенные тестовые интерфейсы, что приведет к неминуемому взлому.

- Анализ питания (Poweranalysis)[5]:

Основываемся на том, что потребляемая мощность устройства основана на его «токовой активности». Ставим 10-20 Омный резистор на линии питания – измеряем токовые флуктуации, а для лучшего результат используем 12 битное разрешение измерений и 50 МГц частоту. Такой подход позволяет нам различать команды процессора и оценить количество бит на шине, изменяющихся во времени.

Если бит состояния не может быть измерен на прямую, то это часто причина изменений в последовательности инструкций или выполнения микрокода, который и изменяет потребление питания. Весь процесс измерения достаточно прост. Существует 2 техники анализа питания: простой анализ питания (SPA) и дифференциальный анализ питания (DPA)

3. Glitchattacks

Глитч атаки – быстрое изменение в сигналах, подаваемых на устройство для внесения изменений в его нормальную работу. Обычно глитчи вставляют в питание и часовые сигналы, но также могут быть и внешним электрическим полем переданного или электромагнитного импульса.

- *Clockglitches*

Обычно глитчи используют для замены условного перехода инструкций и тестовых предшествующих им инструкций. Они создают окно уязвимости в процессорных состояниях различных защитных криптографических барьерах с помощью простого предотвращения выполнения кода, который обнаруживает неудачную попытку аутентификации. Также можно продлевать время выполнения циклов, например, в последовательном порте вывода подпрограмм, чтобы увидеть больше памяти после выходного буфера, а также можно снизить количество криптографических циклов операций.

Для предотвращения такого рода атак, частота часов должна быть увеличена до одного или более полупериодов так, чтобы некоторые триггеры-защелки вносили свой вклад прежде, чем произойдет переход в новое состояние. Обычно клок-глитч нацелен на потоки инструкций CPU, и не эффективны на устройствах с аппаратным решением. Выгодно выполнять атаку на микроконтроллеры с программными интерфейсами или смарт-карты.

- *Powerglitches*

Флуктуации питающего напряжения могут сдвигать пороговый уровень транзисторов. Как результат некоторые триггеры-защелки выбирают свой вход в различное время или просто состояние защитного фьюз-байта будет некорректно считано. Это обычно достигается посредством увеличения напряжения питания или сброса на короткий период времени, обычно от 1 до 10 циклов часов. PowerGlitch'ры применимы к любому программному интерфейсу микроконтроллера, т.к. они могут влиять на работу процессора и схему аппаратного обеспечения.

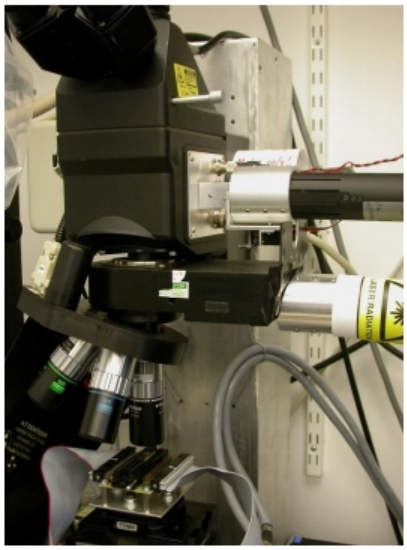


Рис. 1.18: Микроскоп для исследований

4. Извлечение остаточных данных (Data remanence):

- *Низкотемпературное (Low temperature data remanence in SRAM):*

Было установлено, что низкие температуры увеличивают сохранность данных в ячейках SRAM от нескольких секунд до минут.

- *Извлечение из энергозависимой памяти (Data remanence in non-volatile memories):*

Требования для надежного удаления данных из памяти:

- Не сохраняйте криптографические ключи, пароли и другую чувствительную информацию на длинный период времени в SRAM. Время от времени передвигайте их в новые области и обнуляйте достоверные данные.
    - Используйте схемы температурного определения;
    - Ячейки EEPROM/FLASH раз в 10-100 раз заполняйте случайными данными прежде, чем записываете критичные данные, чтобы устранить любые заметные эффекты, возникающие от использования новых ячеек.
    - Программируйте все EEPROM/FLASH ячейки после того как удаляете из них данные (предотвращаем остаточный заряд).

5. Метод обратного представления (Backside imagingtechniques):

6. Активное фотонное зондирование (Active photon probing):

- Основано на воздействии радиационного излучения на работу транзисторов.

- *Лазерное сканирование (Laser scanning techniques)*
- *Считывание логических состояний CMOS транзисторов(Reading the logic state of CMOS transistors)*

8. Атака внесением ошибки (Fault injection attacks):

- *Изменение содержимого SRAM (Changing SRAM contents)*

- *Изменение энергонезависимой памяти (Non-volatile memory contents modification)*

1.10.6. Некоторые способы защиты

- Использование многослойных печатных плат, чипы в BGA корпусе.

  Повышение стоимости за счет невозможности рассмотреть межсоединения, нельзя напрямую подключаться к ножкам BGA корпуса, необходимо специальное тестовое и выпаивающее оборудование, необходима высокая квалификация инженера, более того трудно выполнить декапсуляцию BGA корпусов (изготавливаются из лучшего материала, чем DIP, SOIC, QFP).
- Использование шифрования, если ключ программируемый или может быть изменен в любое время, то лучше поместить его в EPROM или EEPROM.
- Использовать: self-timed dual-rail circuit design techniques. Значительно усложняет проведение power attack, схема может быть сконструирована таким образом, чтобы сбой 1 транзистора не приводил к отказу всей защиты.
- Удаление маркировки с компонентов (усложняет низко бюджетные атаки), увеличивает стоимость клонирования и реверс инжиниринга.
- Добавление собственной маркировки или ложной маркировки, как на высокозащищенных продуктах (ASIC);
- Сжигание ножек, используемых для программирования памяти. Для этого подаем максимально допустимое положительное напряжение на данную ножку и ток более 1 А.

1.11. Конфигурационные и блокировочные биты

Fuse bits – область (4 байта) в AVR микроконтроллерах, отвечающая за глобальную начальную конфигурацию. С помощью фьюз битов мы выбираем, с каким генератором будет осуществляться работа (внешний или внутренний), делить ли частоту генератора коэффициентом или ненужно, использовать ли ножку сброса в качестве сброса или же в качестве дополнительно порта ввода-вывода, количество памяти для загрузчика и некоторые другие функции. У каждого микроконтроллера есть свой набор фьюзов, их описание представлено в руководстве пользователя микроконтроллера от производителя (datasheet). По умолчанию заводскими настройками является поддержка интерфейсов ISP, JTAG для прошивки, а также работа от внутреннего кварцевого генератора. Для запуска только что приобретенного микроконтроллера фактически нужно всего лишь подать на него питание. Если же необходимо выполнить более точную настройку микроконтроллера под необходимую задачу, то необходимо с помощью программатора изменить нужные фьюз биты.

Физические фьюз биты, расположены в четырех специальных байтах:

- Lock Bit Byte – блокировочные биты для защиты программы от копирования;
- Fuse Extended Byte – дополнительный байт – особые функции;
- Fuse High Byte – старший байт;
- Fuse Low Byte – младший байт;

1.11.1. Распределение фьюз битов по байтам ATmega32u4

Таблица 1.15: Lock Byte

Lock Bit Byte	Bit No	Description	Default Value	
			ATmega16U4/32U4	ATmega16U4RC/32U4RC
	7	–	1	
	6	–	1	
BLB12	5	Boot Lock bit	1	
BLB11	4	Boot Lock bit	0	1
BLB02	3	Boot Lock bit	1	
BLB01	2	Boot Lock bit	1	
LB2	1	Lock bit	0	1
LB1	0	Lock bit	0	1

Note: 1. "1": unprogrammed, "0": programmed

Таблица 1.16: Extended Byte

Fuse Low Byte	Bit No	Description	Default Value	
			ATmega16/32U4	ATmega16/32U4RC
–	7	–	1	
–	6	–	1	
–	5	–	1	
–	4	–	1	
HWBE	3	Hardware Boot Enable	0 (programmed)	1 (unprogrammed)
BODLEVEL2(1)	2	Brown-out Detector trigger level	0 (programmed)	
BODLEVEL1(1)	1	Brown-out Detector trigger level	1 (unprogrammed)	
BODLEVEL0(1)	0	Brown-out Detector trigger level	1 (unprogrammed)	

Notes: 1. See Table 8-1 on page 53 for BODLEVEL Fuse decoding.
2. "1" means unprogrammed, "0" means programmed.

Таблица 1.17: High Byte

Fuse High Byte	Bit No	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
WDTON[3]	4	Watchdog Timer always on	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM preserved)
BOOTSZ1	2	Select Boot Size (see Table 28-7 for details)	0 (programmed)[2]
BOOTSZ0	1	Select Boot Size (see Table 28-7 for details)	0 (programmed)[2]
BOOTRST	0	Select Bootloader Address as Reset Vector	1 (unprogrammed, Reset vector @0x0000)

Note: 1. The SPIEN Fuse is not accessible in serial programming mode.
2. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 28-7 for details.
3. See "Watchdog Timer" on page 55 for details.
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of Lock bits and JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.

Таблица 1.18: Low Byte

Fuse Low Byte	Bit Nr	Description	Default Value	
			ATmega16U4/32U4	ATmega16U4RC/32U4RC
CKDIV8[3]	7	Divide clock by 8	0 (programmed)	
CKOUT[2]	6	Clock output	1 (unprogrammed)	
SUT1	5	Select start-up time	0 (programmed)	
SUT0	4	Select start-up time	1 (unprogrammed)	
CKSEL3	3	Select Clock source	1 (unprogrammed)[1]	0 (programmed)[1]
CKSEL2	2	Select Clock source	1 (unprogrammed)[1]	0 (programmed)[1]
CKSEL1	1	Select Clock source	1 (unprogrammed)[1]	1 (unprogrammed)[1]
CKSEL0	0	Select Clock source	0 (programmed)[1]	0 (programmed)[1]

Note: 1. The default setting of CKSEL3..0 results in Low Power Crystal Oscillator for ATmega16U4 and ATmega32U4, and Internal RC oscillator for ATmega16U4RC and ATmega32U4RC.
2. The CKOUT Fuse allow the system clock to be output on PORTC7. See "CLKPR – Clock Prescaler Register" on page 39 for details.
3. See "System Clock Prescaler" on page 35 for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

Исторически сложилось, что у микроконтроллеров AVR:

- 0 – запрограммирован, активен;
- 1 – не запрограммирован, не активен;

Это связано с тем, что в эпоху, когда еще отсутствовала флэш-память, для конфигурации какого-либо чипа в его архитектуре имелись специальные перемычки (fuses), которые после конфигурации

разово физически сжигались. Отсюда следует, что, если перемычка цела, то это логическая единица, т.е. перемычка не задействована, иначе – логический ноль, перемычка задействована.

1.12. Устройство Fuse битов

CKSEL – выбор тактового генератора для микроконтроллера.
Для работы микроконтроллера (как и для любого процессора) нужны тактовые импульсы. В качестве источника тактового сигнала может выступать:

- Внутренний RC генератор: имеет большую погрешность (вплоть до 10%) и очень сильно зависит от температуры;
- Внешний кварцевый (или керамический) резонатор: подключается к ножкам XLAT1 и XLAT2, имеет высокую точность работы по сравнению с RC цепочкой;
- Внешний источник тактового сигнала: например, другой микроконтроллер (для синхронизации работы) или внешняя схема;

На CKSEL приходится 4 бита.

CKOUT – разрешает вывод тактовой частоты на ножку CLKO микроконтроллера (для тактирования других устройств).

- CKOUT = 1 – ножка микроконтроллера работает как обычный порт ввода-вывода;
- CKOUT = 0 – на ножку микроконтроллера выдается сигнал тактового генератора.

CKOPT – задает размах тактового сигнала на внешнем генераторе.

- CKOPT = 1 – размах небольшой – генератор работает в экономном режиме;
- CKOPT = 0 – задающий генератор работает на полную мощность, устойчив к помехам и может работать во всем диапазоне частот. ;

SCKDIV8 – деление тактовой частоты на 8.

- SCKDIV8= 1 – микроконтроллер работает на частоте задающего генератора;
- SCKDIV8= 0 – микроконтроллер работает на частоте в 8 раз меньше частоты задающего генератора;

SUT – задает скорость запуска микроконтроллера.

Микроконтроллер выжидает некоторое время, для того, чтобы нормально запустился тактовый генератор, установилось напряжение питания и т.д. Время ожидания до запуска программы задают биты SUT1…0

RSTDISBL –разрешает использовать ножку Reset как еще один порт ввода-вывода.

- RSTDISBL = 1 – ножка сброса работает как сброс;
- RSTDISBL = 0 – ножка сброса работает как еще один порт ввода-вывода, последовательное программирование отключено.

SPIEN – разрешение на последовательное программирование.

- SPIEN = 0 – разрешено последовательное программирование;
- SPIEN = 1 – запрещено последовательное программирование.

WDTON – включает Watch Dog Timer.
Это внутренний таймер микроконтроллера, работающий от своего независимого генератора. При переполнении этого таймера микроконтроллер сбрасывается и начинает выполнять программу с

начала. В программе необходимо использовать команды обнуления таймера, если микроконтроллер «повис» перестают выполняться команды обнуления, таймер переполняется и сбрасывает микроконтроллер.

- WDTON = 1 – Watch Dog Timer – отключен (можно включить программно);
- WDTON = 0 – Watch Dog Timer – включен (программно выключить нельзя).

BODLEVEL и BODEN – контроль напряжения питания микроконтроллера (Brownout Detector).

Если питание микроконтроллера опуститься к минимально допустимому или чуть ниже, то работа микроконтроллера будет нестабильной. Возможны ошибочные действия, потеря данных, случайное стирание EEPROM. Микроконтроллер умеет следить за уровнем своего питания (BODEN=0) и когда оно достигает уровня, который задается битами BODLEVEL, сбрасывается и находится в состоянии перезагрузки пока уровень не поднимется до рабочего.

JTAGEN – разрешает интерфейс JTAG (внутрисхемный отладчик).

- JTAGEN = 1 – запрещен JTAG;
- JTAGEN = 0 – разрешен JTAG;

DWEN – бит, разрешающий работу Debug Wire – еще одного отладочного интерфейса.

Debug Wire однопроводной отладочный интерфейс работающий через ножку сброса, поэтому «не отнимает» у микроконтроллера ножки портов ввода-вывода.

- DWEN= 1 – запрещен Debug Wire;
- DWEN= 0 – разрешен Debug Wire;

SELFPRGEN – бит, разрешающей программе производить запись в память программ.

- SELFPRGEN = 1 – изменение области программ запрещено;
- SELFPRGEN = 0 – разрешено изменение области программ.

EESAVE – защита EEPROM от стирания.

- EESAVE = 1 – стирать EEPROM вместе с Flash;
- EESAVE = 0 – оставлять EEPROM при очистке нетронутым;

BOOTRST – указывает микроконтроллеру начинать запуск из области загрузчика.

- BOOTRST = 1 – микроконтроллер запускает программу с нулевого адреса;
- BOOTRST = 0 – микроконтроллер запускает программу с загрузчика;

BOOTSZ0..1 – задает размер бут сектора (области памяти программ для загрузчика).

Lock Bits – Это отдельный фьюз байт который предназначен для защиты области программ и/или EEPROM от копирования. Полное стирание восстанавливает эти биты в исходное состояние.

2. ПРОГРАММНАЯ ЧАСТЬ

2.1. Средства разработки программного обеспечения

2.1.1. Arduino IDE

Arduino IDE – свободная кроссплатформенная среда разработки для микроконтроллеров, написанная на Java. Среда разработки представляет из себя встроенный текстовый редактор программного кода, с областью сообщений, предоставляет встроенную консоль, панель инструментов с кнопками часто используемых команд.

Среда использует: компилятор C/C++ GCC в виде сборки WinAVR, поддерживает разного рода программаторы, использует интегрированный AVRDude для загрузки программ в микроконтроллеры, также содержит в себе набор стандартных примеров, библиотек.

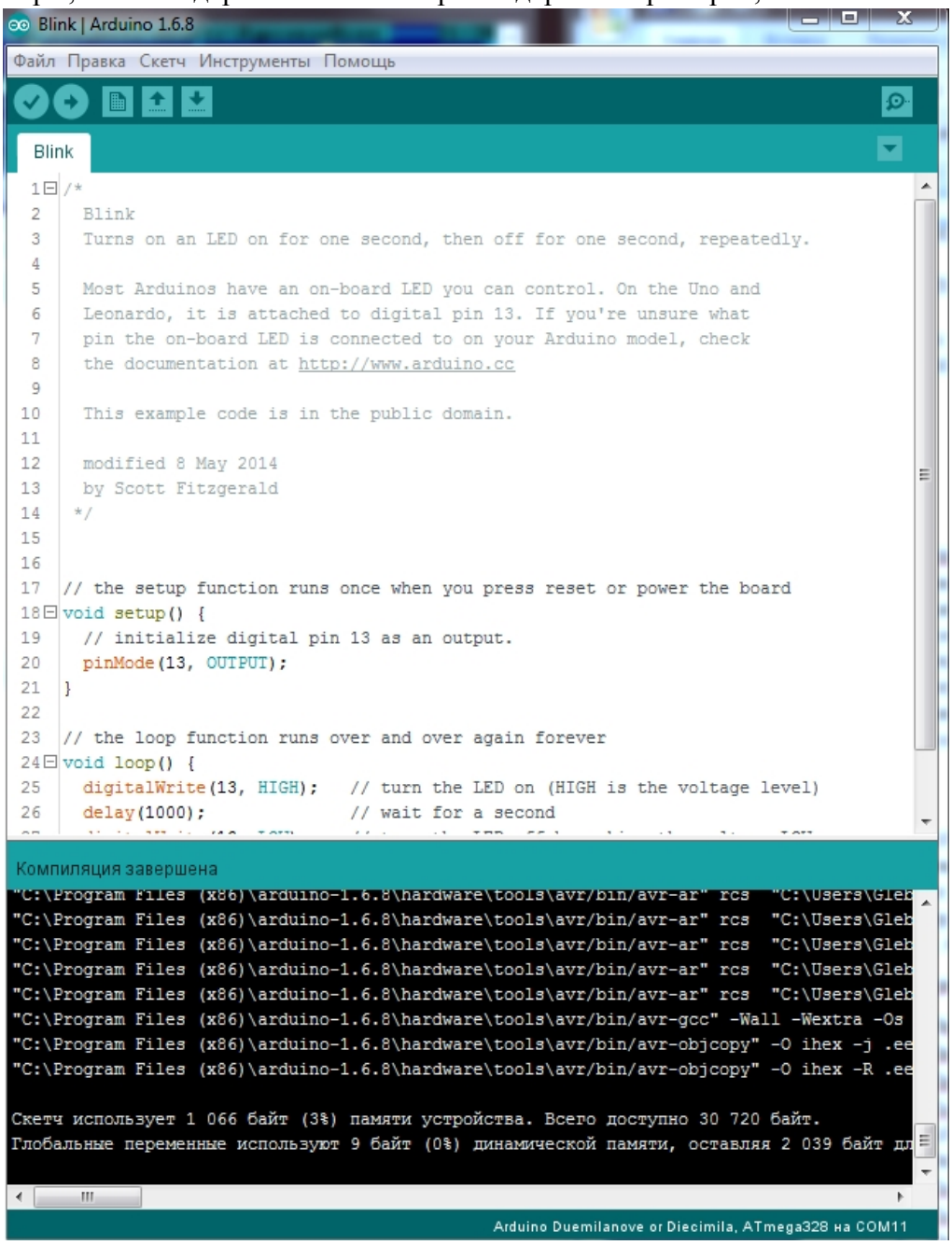


Рис. 2.1: Интерфейс среды разработки Arduino[8]

2.1.2 AVR Studio (Atmel Studio)

AVR Studio — интегрированная среда разработки (IDE) для разработки 8- и 32-битных AVR-приложений от компании Atmel, работающая в операционных системах Windows. AVR Studio содержит компилятор C/C++ и набор симуляторов, позволяющие отследить выполнение программы на различных микроконтроллерах, менеджер проектов, редактор исходного кода, инструменты виртуальной симуляции внутрисхемной отладки, позволяет писать программы на ассемблере или на C/C++.

Характеристики AVR Studio:

- Интегрированный компилятор C/C++;
- Интегрированный симулятор;
- При помощи дополнения возможна поддержка компилятора GCC в виде сборки WinAVR;
- Поддержка инструментов Atmel, совместимых с 8-разрядной AVR архитектурой, в том числе AVR ONE!, JTAGICE mkI, JTAGICE mkII, AVR Dragon, AVRISP, AVR ISPmkII, AVR Butterfly, STK500 и STK600;
- Поддержка дополнения AVR RTOS;
- Поддержка AT90PWM1 и ATtiny40;
- Интерфейс командной строки с поддержкой TPI.

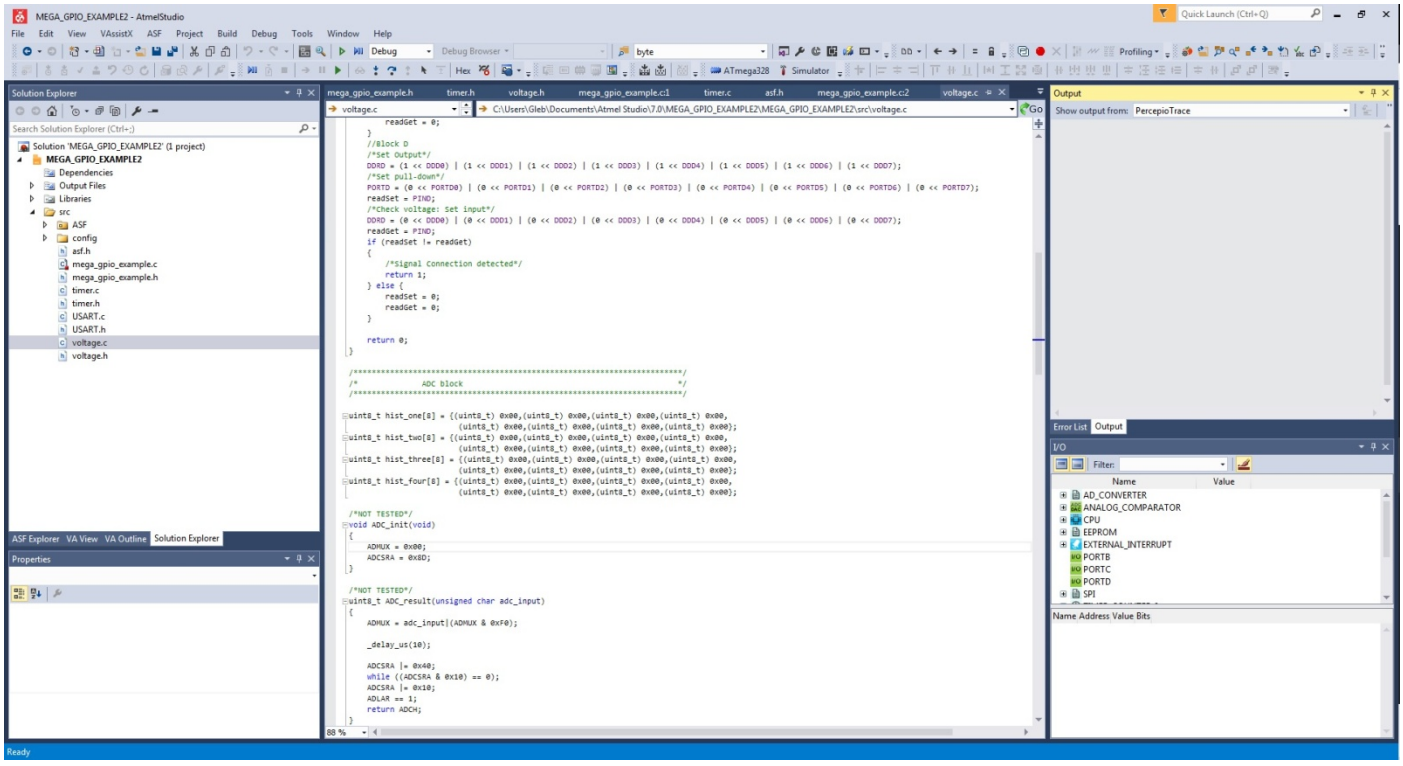


Рис. 2.2: Atmel Studio v.7.0

2.1.3. AVRDude

Кроссплатформенная свободная консольная программа, предназначенная для загрузки программ на микроконтроллеры фирмы Atmel серии AVR.

Отличается широким спектром поддерживаемых программаторов и микроконтроллеров. Кроме стандартных устройств от Atmel сюда входят самодельные любительские устройства, не поддерживаемые AVR Studio, к примеру, популярный программатор USBAsp.

Таблица 2.1: Примеры использования

Считывание данных из Flash-памяти и сохраняем в файл	avrdude -p m8 -c usbasp -P usb -v -U flash:r:/home/nixuser/m8_flash_dump.hex:i
Запись данных из файла	avrdude -p t13 -c stk500 -U flash:w:/home/nixuser/dump_m8.hex
Запись данных во Flash и EEPROM	avrdude -p t13 -c usbasp -U flash:w:"c:\temp\flash_dada.hex" -U eeprom:w:"c:\temp\eeprom_data.hex"
Чтение Fuse битов	avrdude -p t13 -c usbasp -U hfuse:r:hfuse.txt:h -U lfuse:r:lfuse.txt:h
Установка Fuse битов	avrdude -p m16 -c usbasp -U lfuse:w:0xe1:m -U hfuse:w:0x99:m

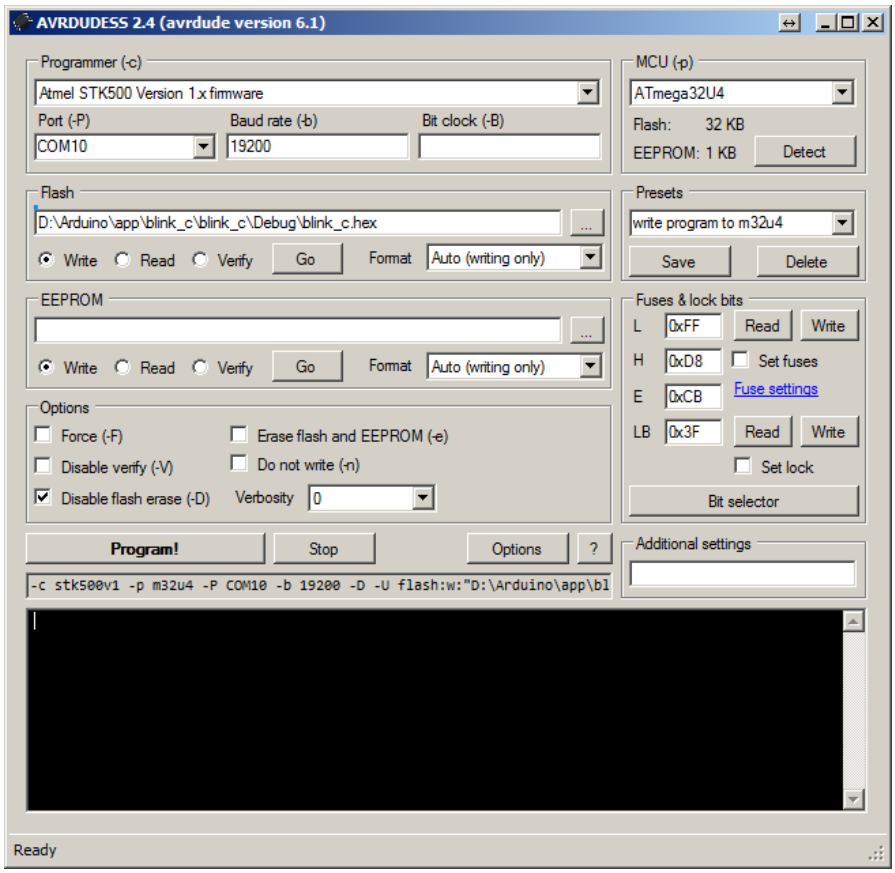


Рис. 2.3: Интерфейс AVRDUDESS для AVRDude

2.1.4. Программатор Arduino as ISP

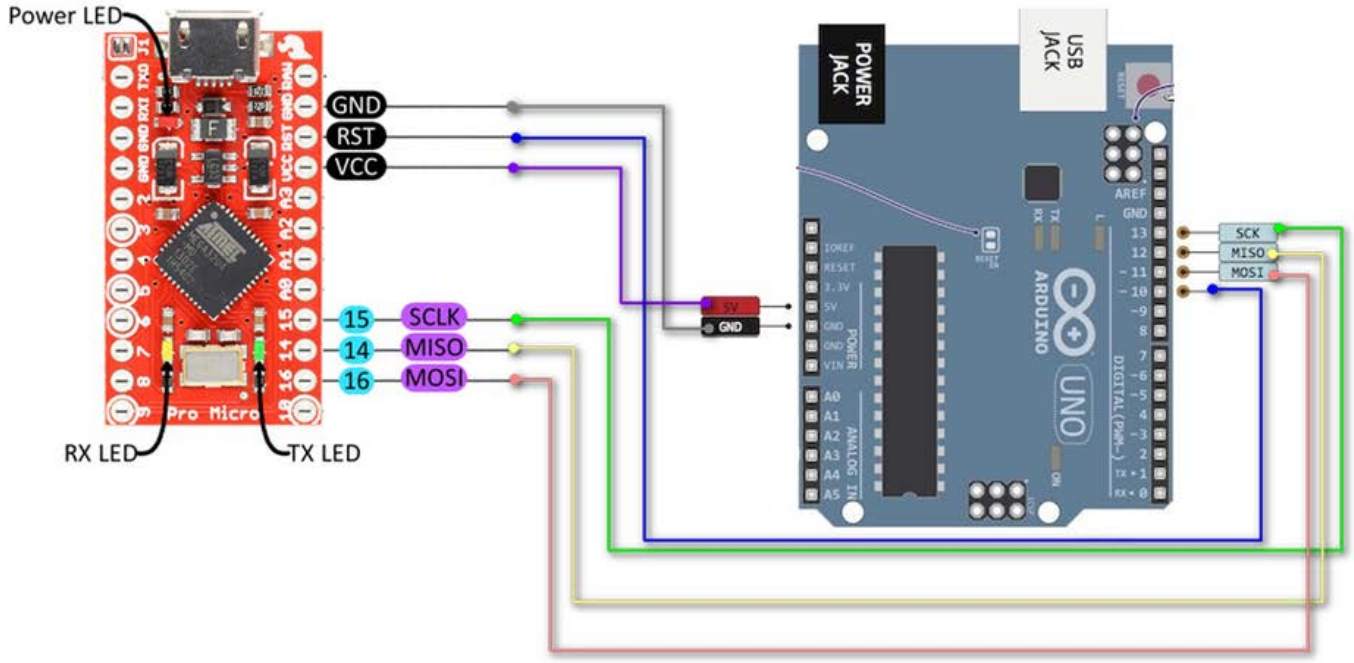


Рис. 2.4: Схема подключения Arduino as ISP

Не совсем очевидным решением является использование Arduino платы в качестве ISP программатора. Но по сути микроконтроллеру всего лишь необходимо среагировать на команду прошивки от UART интерфейса и транспонировать полученные данные и команды по ISP интерфейсу другому подключенному к нему микроконтроллеру. Такая простая идея легла в основу прошивки для Arduino, которая является стандартным примером на платформе.

Прошивка для данного режима работы является стандартной и есть в примерах IDE.

2.1.5. Компилятор WinAVR

WinAVR – программный пакет для операционных систем семейства Windows, включающий в себя кросс-компилятор и инструменты разработки для микроконтроллеров серий AVR и AVR32 фирмы Atmel.

WinAVR и все входящие в него программы являются открытым программным обеспечением, выпущенным под лицензией GNU, но распространяются в скомпилированном виде.

WinAVR включает следующий набор компонентов:

- Programmers Notepad;
- AVR GCC – компилятор для языков C/C++ для AVR;
- AVR-LIbC – стандартная C-библиотека AVR для использования с GCC;
- GNU_Binutlis – коллекция утилит, включающая в себя ассемблер avr-as, компоновщик и утилиты манипуляции файлами в форматах .elf, .coff для микроконтроллеров AVR;
- MFile – автоматический генератор управляющего файла программы make, контролирующей сборку программ с помощью AVR GCC;
- gdb – отладчик с интерфейсом командной строки;
    - insigt – оболочка графического интерфейса отладчика;

- SimulAVR – симулятор семейства микроконтроллеров AVR с поддержкой интерфейса к отладчику gdb;
- AVaRICE – программа для работы с внутрисхемным отладочным интерфейсом JTAG ICE в микропроцессорах Atmel;

- Avrdude – программатор;
- SRecord – коллекция утилит для манипуляции с загрузочными файлами EPROM различных форматов;

2.1.6. Библиотека LUFA

LUFA (Lightweight **USB** Framework for AVRs, т.е. "облегченная рабочая среда USB для микроконтроллеров AVR") open-source (имеет открытый исходный код) библиотека USB для микроконтроллеров AVR, имеющих аппаратную поддержку USB, выпущенная под лицензией MIT. Она поддерживает большое количество моделей USB AVR и макетных плат. LUFA разработана, чтобы предоставить простую в использовании рабочую среду для разработки периферийных устройств USB и хостов.

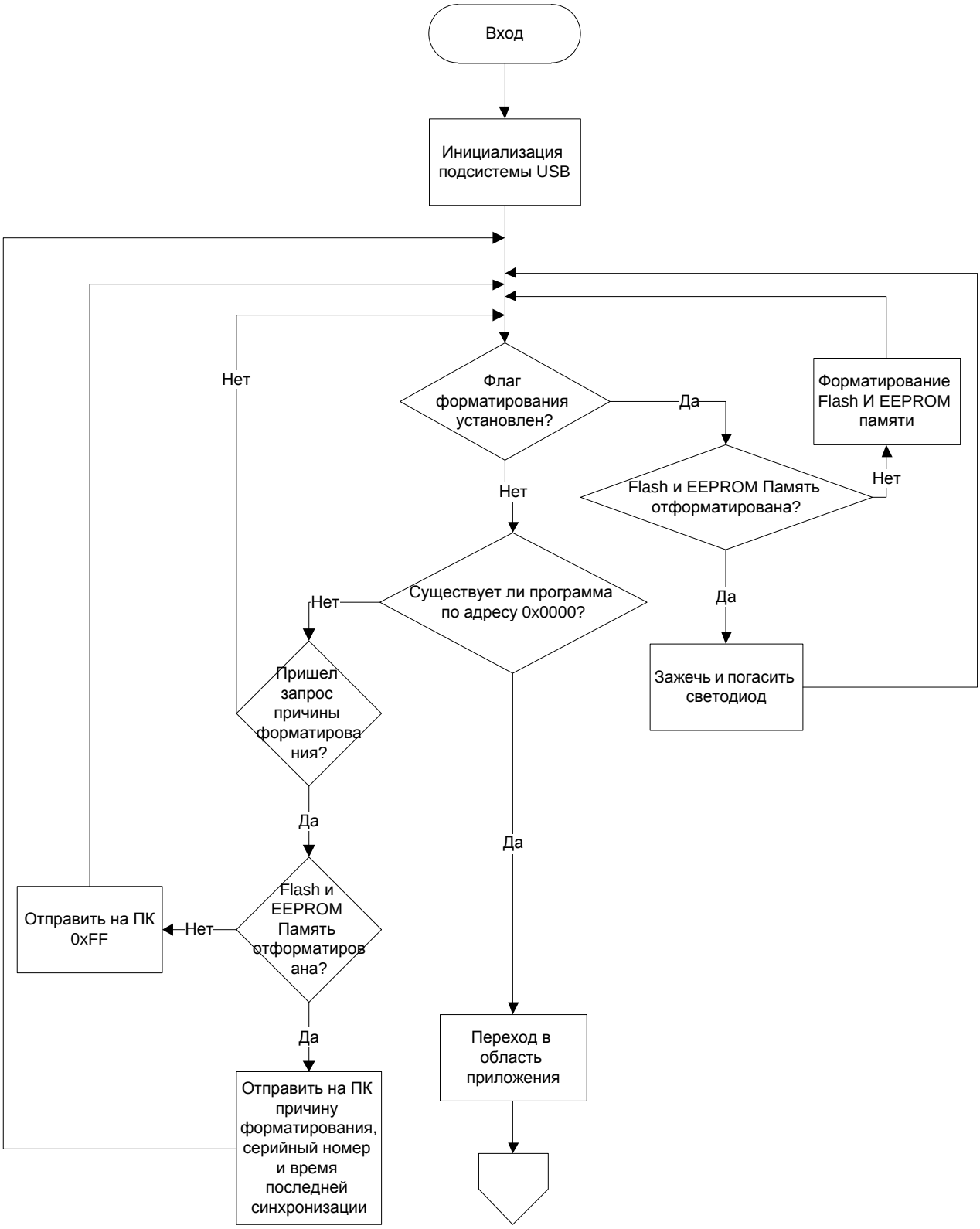
LUFA фокусируется только на разработке USB со стороны микроконтроллера; она не включает в себя возможности для разработки USB драйверов хоста PC или программ - для этого используйте другие проекты наподобие Windows Driver Development Kit, Windows USB Device Mode Framework и libusb. В то время как пользовательские USB-устройства могут быть сделаны на LUFA с использованием таких дополнительных инструментов, для простоты все встроенные в LUFA демонстрационные примеры рассчитаны на использование встроенных драйверов операционной системы для каждого класса USB.

В настоящее время библиотека имеет текущий стабильный релиз, доступный для загрузки и подходящий для встраивания в проекты пользователя в обоих режимах USB - и хоста и устройства. LUFA написана специально для использования совместно с бесплатным компилятором AVR-GCC, и использует несколько расширений, специфичных только для GCC, чтобы сделать API библиотеки более оптимизированным и устойчивым.

Из периферии AVR для LUFA нужен только сам по себе контроллер USB и его прерывания - LUFA не требует использования таймеров микроконтроллера или другой аппаратуры, оставляя больше свободных ресурсов чипа для разработчика приложения.

2.2. Основные алгоритмы

2.2.1. Алгоритм загрузчика



2.2.2. Общий алгоритм приложения

Командные и ответные пакеты описаны в таблицах 2.6 и 2.7

2.3. Описание функций приложения на микроконтроллере

Таблица 2.2: функции приложения на микроконтроллере.

№	Команда	Описание
1	1. USB_USBTask() 2. EVENT_USB_Device_ConfigurationChanged() 3. EVENT_USB_Device_ControlRequest()	1. Главная функция при USB соединении, при котором устройство поддерживается в режиме работы в качестве периферийного или управляющего устройства; 2. Используется для создания конечных точек и принятия заданных настроек устройством; 3. Управляющее USB устройство проверяет обязательное наличие конечной точки – рукопожатие (handshake);
2	1. eeprom_read_byte() 2. eeprom_read_dword() 3. eeprom_write_dword_safe() 4. eeprom_write_byte_safe()	1. Чтение 1 байта из EEPROM; 2. Чтение двойного слова из EEPROM; 3. Безопасная запись двойного слова в EEPROM; 4. Безопасная запись 1 байта в EEPROM;
3	FetchNextCommandByte()	Получить 1 байт по USART
4	_delay_ms(@)	Задержка в миллисекундах
5	1. cli() 2. sei()	1. Очистить глобальный флаг в регистре SREG; 2. Установить глобальный флаг в регистре SREG;
6	boot_lock_fuse_bits_get(@)	Прочитать значения lock бита или фьюз бита
7	1. wdt_enable(@) 2. wdt_disable()	1. Включить Watchdog таймер; 2. Отключить Watchdog таймер;
8	clock_prescale_set(@)	Установить необходимый пред делитель таймера-счетчика
9	1. Endpoint_IsINReady() 2. Endpoint_SelectEndpoint(@) 3. Endpoint_ClearIN() 4. Endpoint_IsReadWriteAllowed() 5. WriteNextResponseByte(@) 6. Endpoint_ClearOUT(); 7. Endpoint_Read_8();	1. Сигнализирует о том, получило ли конечное устройство данные; 2. Выбрать конечное устройство; 3. Отправка IN пакета, очистить конечную точку для отправки нового пакета; 4. Проверка готовности конечной точки; 5. Отправить байт; 6. Отправка OUT пакета, очистить конечную точку для отправки нового пакета; 7. Считать 1 байт из конечной точки;
10	1. LED_SETUP() 2. RX_LED_OFF() 3. TX_LED_OFF()	1. Инициализировать светодиод; 2. Выключить светодиод на линии RX; 3. Выключить светодиод на линии TX;
11	ISR(TIMER1_COMPA_vect, ISR_BLOCK);	Процедура прерывания по сравнению таймера-счетчика с регистром

12	main()	Главная процедура программы
13	ReseivePrepare()	Процедура подготовки к приему
14	pc_mc_connection()	Процедура взаимодействия ПК и МК
15	RequirementCMD(@)	Процедура проверки поддержки команды
16	MainWorkingCMD(@)	Процедура с поддерживаемыми командами управления МК
17	JoinArray(@)	Соединение 8 битного массива в 32 битное число
18	1. SplitArray(@) 2. split_int(@)	Разбить 32 битную переменную на 8 битные числа
19	SendPlusTimeMistake()	Отправка положительной временной ошибки
20	SendCurrTimeFromRegister()	Отправка текущего времени микроконтроллера
21	SendMinusTimeMistake()	Отправка отрицательной временной ошибки
22	SendSynchroTimeFromEEPROM()	Отправить время синхронизации из EEPROM
23	TimeSynchronization()	Процедура синхронизации времени МК и ПК
24	ReceivedSynchroTimeToEEPROM()	Записать поученной время синхронизации в EEPROM
25	SendMaxPositive()	Отправить на ПК допустимую максимальную отрицательную накрутку
26	SendMaxNegative()	Отправить на ПК допустимую максимальную положительную накрутку
27	Initialization()	Процедура инициализации параметров микроконтроллера
28	get_data_from_buffer(@)	Вывести данные из буфера
29	check_up_down_time_mistake(@)	Проверить скопившуюся временную ошибку
30	SetFormattingMarkAndReboot(@)	Установить флаги форматирования и перезагрузка микроконтроллера
31	send_package_ENCRYPT(@)	Отправить зашифрованный пакет
32	send_package_NO_ENCRYPT(@)	Отправить незашифрованный пакет
33	CreateAndSendPackage(@)	Объединение данных в пакет и отправка
34	SendByteArray(@)	Отправить массив байт
35	SendOneByte(@)	Отправить один байт
36	FetchNextCommandByte()	Принять следующий байт по каналу связи
37	WriteNextResponseByte(@)	Отправить следующий байт
38	GenerateKey(void)	Сгенерировать ключ шифрования

39	GenerateAndUseDefaultKey()	Сгенерировать ключ по умолчанию, если пароль установлен
40	encryptArray(@)	Зашифровать массив
41	decryptArray(@)	Расшифровать массив
42	reboot()	Перезагрузить микроконтроллер
43	1. SetPassword(@) 2. CheckPassword(@)	1. Установить пароль; 2. Проверить пароль;
44	1. write_serial_num(@) 2. send_serial_num()	1. Установить серийный номер; 2. Отправить серийный номер;
45	Send_SN_forMan()	Отправить серийный номер
45	1. SetEndLicenseTimeToEEPROM(@) 2. SendEndLicenseTimeFromEEPROM() 3. CheckLicenseTime()	4. Сохранить время конца лицензии в EEPROM МК; 5. Отправить время конца лицензии; 6. Проверить лицензию;
44	1. Action() 2. SetActionMaxQuontity(@) 3. SendActionMaxQuontity() 4. SendRemainCurrentQuontityAction() 5. IncrementCurrentActionCounter()	1. Обследование/беседа с проверяемым; 2. Установить максимальный размер бесед в сутки. Если = 0, то без ограничений; 3. Отправить на ПК максимальное количество бесед в сутки; 4. Отправить оставшееся количество бесед за текущие сутки; 5. Увеличить счетчик бесед в EEPROM за текущий день.
45	1. SetGMT_Corrector(@) 2. SendGMT_Corrector()	1. Установить корректор времени в EEPROM; 2. Отправить корректор времени, который установлен для часового пояса;
46	ComputeNewEndDayUTS_ and_SetInEEPROM()	вычислить конец текущих суток и установить в EEPROM
47	CheckFuseAndLock()	Проверка Fuse и Lock битов
48	CheckRequirement()	Проверить, установлены ли требуемые инициализационные настройки
49	1. DB_ReadAll(void) 2. DB_SetOneVal(@) 3. DB_GetOneVal(@)	1. Считать всю БД; 2. Записать 1 значение в БД по указанному индексу; 3. Считать 1 значение из БД по указанному индексу;
50	1. get_hist_max_min(@) 2. get_histogram (@) 3. find_max_min(@)	Гистограммный метод анализа показаний с ножек микроконтроллера
51	ADC_get_value(char adc_input)	Получить данные с ADC
52	1. compute_f_wzh(@) 2. compute_wzh_Array(@) 3. computeFurmula_whz(@)	1. Запустить вычисления и отправить результат на ПК; 2. Посчитать значения F0, F1..Fn;

		3. Выполнить вычисления по формуле;
53	CheckRSTflags()	Проверить флаги причин перезагрузки МК

2.4. Протокол взаимодействия ПК и МК

2.4.1. Виды секретных ключей

1. Заводской секретный ключ

Ключ, записанный жестко в коде. Этим ключом осуществляется шифрование, пока не установлен пароль. Если пароль не установлен, то другие команды не выполнять.

2. Секретный ключ по умолчанию

Ключ, сформированный из пароля. Ключ по умолчанию формируется в момент установки пароля.

3. Временный секретный ключ

Сгенерированный случайный ключ. Данный ключ генерируется по команде от ПК. После генерации, новый ключ отправляется на ПК зашифрованный старым ключом. Далее общение с МК должно происходить на новом ключе.

2.4.2. Алгоритмы взаимодействия после загрузки приложения на МК

Взаимодействие с ключом после прошивки:

1) Подача питания ключа
2) Запуск защищаемого программного обеспечения
3) Установка пароля 40 символов на заводском ключе
4) Обмен данными на секретном ключе по умолчанию
5) Установка нового случайного секретного ключа
6) Обмен данными на новом секретном ключе
7) Выход из защищаемого программного обеспечения
8) Отключение питания ключа

Взаимодействие с прошитым ключом, на котором уже установлен пароль

1) Подача питания ключа
2) Запуск защищаемого программного обеспечения
3) Обмен данными на секретном ключе по умолчанию
4) Установка нового случайного секретного ключа
5) Обмен данными на новом секретном ключе
6) Выход из защищаемого ПО
7) Отключение питания ключа

Взаимодействие с ключом при включенном питании, установленном пароле и новом сгенерированном секретном ключе.

1) Запуск защищаемого программного обеспечения
2) Отправка команды сброса секретного ключа на ключ по умолчанию
3) Обмен данными на секретном ключе по умолчанию
4) Установка нового случайного секретного ключа
5) Обмен данными на новом секретном ключе
6) Выход из защищаемого программного обеспечения

2.4.3. Порядок взаимодействия с МК

1) Отправка «Handshake» на МК
2) Прием «Handshake» от МК
3) Отправка зашифрованного командного пакета данных
4) Прием зашифрованного ответного пакета данных

2.4.4. Описание форматов пакетов данных

Таблица 2.3: Формат командного пакета данных

Команда	**Размер**	**Данные**
C	N	D - N элементов

Пример:

Package[9] = {43, 0, 0, 5, 14, 62, 15, 86, 97}, где 43 – команда, 0, 0, 5 – размер данных, 14, 62, 15, 86, 97 – данные.

Таблица 2.4: Формат ответного пакета данных

Ответ	**Размер**	**Данные**
R	N	D - N элементов

Пример:

Package[9] = {0, 0, 0, 5, 14, 62, 15, 86, 97}, где 0 – ответ, 0, 0, 5 – размер данных, 14, 62, 15, 86, 97 – данные.

Таблица 2.5: Формат зашифрованного пакета (командный/ответный)

Команда/ответ	**Размер**	**Зашифрованные данные**
C/R	N	N – элементов

Пример:

Package[6] = {5, 45, 75, 24, 14, 41}, где 5 – размер, 45, 75, 24, 14, 41 – зашифрованная последовательность.

2.4.5. Команды взаимодействия ПК и МК

Таблица 2.6: Команды доступные до и после инициализации.

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
Z	88	58	Отправить приветствие на МК. Приветствие происходит без шифрования.	88	-	-	88	-	-	После взаимного приветствия между ПК и МК необходимо отправить зашифрованный командный пакет данных и принять зашифрованный ответный пакет данных. Если этого не сделать, то МК отправит в COM порт ошибку истечения таймаута 0xFF
+	43	2B	Синхронизировать время МК и ПК.	43	0,0,5,	14, 63, 67, 86, 23	0	0,0,0	-	14, 63, 67, 86, 23 – время в формате unix timestamp.14, 63, 67, 86, 23 =463678623= 19 Май 2016 г. 20:23:43. В данном случае в ответном пакете МК

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
										сообщает, что он подтверждает успешную синхронизацию времени с ПК.
?	63	3F	Запрос текущего времени МК.	63	0,0,0		0	0,0,5,	14, 62, 16, 9, 51	14, 62, 16, 09, 51 – время в unix timestamp.
I	73	49	Отправить на ПК время синхронизации записанное в EEPROM.	73	0,0,0	-	0	0,0,5	14, 62, 16, 5, 93	14, 62, 16, 5, 93 – время в формате unix timestamp.
F	70	46	Отправить на ПК сумму положительных накруток в секундах.	70	0,0,0	-	0	0,0,5	0,0,0,0,0	0, 0, 0, 0, 0 – сумма положительных накруток.
M	77	4D	Отправить на ПК сумму отрицательных накруток в секундах.	77	0,0,0		0	0,0,5	0, 0, 0, 0, 0, 2	0, 0, 0, 0, 2 – сумма отрицательных накруток в секундах, т.е. в данном случае 2 секунды.

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
D	68	44	Отправить на ПК допустимую максимальную положительную накрутку.	68	0,0,0	-	0	0,0,5	0, 32, 16, 66, 18	Максимальная допустимая положительная накрутка вычисляется как 102% от длительности лицензии. 0, 32, 16, 66, 18 – максимальная положительная накрутка в секундах т.е. в данном случае 32166618 секунд или 372 дня, 7 часов, 10 минут и 18 секунд.
K	75	4B	Отправить на ПК допустимую максимальную отрицательную накрутку.	75	0,0,0	-	0	0,0,5	0, 0, 63, 07, 18	т.е. в данном случае 630718 секунд или 7 дней, 7 часов, 11 минут и 58 секунд. Максимальная допустимая отрицательная накрутка вычисляется как 2% от длительности лицензии. 0, 0, 63, 07, 18 – максимальная положительная накрутка в секундах

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
P	80	50	Установить пароль.	80	0,0,40	81, 87, 69, 68, 70, 76, 75, 78, 76, 68, 68, 70, 76, 75, 71, 78, 52, 75, 51, 74, 52, 71, 57, 51, 74, 71, 48, 74, 51, 52, 74, 78, 71, 51, 78, 52, 48, 71, 52, 70	0	0,0,0	-	Размер пароля всегда должен быть 40 символов. 81, 87, 69, 68, 70, 76, 75, 78, 76, 68, 68, 70, 76, 75, 71, 78, 52, 75, 51, 74, 52, 71, 57, 51, 74, 71, 48, 74, 51, 52, 74, 78, 71, 51, 78, 52, 48, 71, 52, 70 – пароль.
W	87	57	Записать серийный номер в EEPROM.	87	0,0,4	74, 52, 14, 84	0	0,0,0	-	Максимальны серийный номер может составлять 2^32.
S	83	53	Отправить серийный номер на ПК.	83	0,0,0	-	0	0,0,4	74, 52, 14, 84	74, 52, 14, 84 – серийный номер. То есть 74521484.
T	84	54	Установить время окончания лицензии в EEPROM.	84	0,0,5	14, 62, 17, 16, 29	0	0,0,0	-	14, 62, 17, 16, 29 – время окончания лицензии в формате unix timestamp.
E	69	45	Отправить на ПК время окончания лицензии из EEPROM.	69	0,0,0	-	0	0,0,5	14, 62, 17, 16, 29	14, 62, 17, 16, 29 – вермя окончания лицензии в формате unix timestamp.

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
H	72	48	Установить максимальное количество бесед в сутки.	72	0,0,1	40	0	0,0,0	-	Если максимальное количество = 0, то без ограничений. В данной команде 40 - это максимальный размер бесед в сутки.
G	71	47	Отправить на ПК максимальное количество бесед в сутки.	71	0,0,0	-	0	0,0,1	40	
R	82	52	Отправить на ПК оставшееся количество бесед за текущие сутки.	82	0,0,0	-	0	0,0,1	40	40 – количество оставшихся бесед на текущие сутки.
Q	81	51	Установить корректор времени в EEPROM.	81	0,0,3	01, 08, 00	0	0,0,0	-	01, 08, 00(10800 сек.) – корректор времени для GMT+3 для Московского часового пояса
J	74	4A	Отправить корректор времени на ПК, который установлен для часового пояса.	74	0,0,0	-	0	0,0,3	01, 08, 00	01, 08, 00 (14400 сек.) - корректор времени для GMT+3 т.е. для Московского часового пояса.

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
4	52	34	Сгенерировать новый ключ.	52	0,0,0	-	0	0,0,32	1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1	1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1 - новый ключ
N	78	4E	Отформатировать flash память МК.	78	0,0,0	-	0	0,0,0	-	Ответ 0 обозначает, что флаг форматирования успешно установлен и сейчас произойдет форматирование.
Y	89	59	Сбросить секретный ключ на ключ по умолчанию.	89	0,0,0	-	0	0,0,0	-	

Таблица 2.7: Команды, доступные после инициализации

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
U	85	55	Обследование/ беседа с проверяемым. Инкрементирует текущий счетчик бесед в сутки и возвращает его значение.	85	0,0,0	-	-	-	-	Команда ничего не возвращает
5	53	35	Считать всю БД.	53	0,0,0	-	0	0,0,32	0, 57, 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	- содержимое БД, в которой установлен только один элемент с индексом 1.
6	54	36	Записать 1 значение в БД по указанному индексу.	54	0,0,2	1, 57	0	0,0,0	-	1 – индекс БД, 57 – значение, которое будет записано по индексу.
7	55	37	Считать 1 значение из БД по указанному индексу.	55	0,0,1	-	0	0,0,1	57	

ASCII	Dec	Hex	Значение	Командный пакет			Ответный пакет			Комментарий
				Команда	Размер данных	Данные согласно размеру	Ответ	Размер данных	Данные согласно размеру	
A	65	41	Авторизация.	65	0,0,40	81, 87, 69, 68, 70, 76, 75, 78, 76, 68, 68, 70, 76, 75, 71, 78, 52, 75, 51, 74, 52, 71, 57, 51, 74, 71, 48, 74, 51, 52, 74, 78, 71, 51, 78, 52, 48, 71, 52, 70	0	0,0,0	-	81, 87, 69, 68, 70, 76, 75, 78, 76, 68, 68, 70, 76, 75, 71, 78, 52, 75, 51, 74, 52, 71, 57, 51, 74, 71, 48, 74, 51, 52, 74, 78, 71, 51, 78, 52, 48, 71, 52, 70 – пароль. Вместе с авторизацией происходит выполнение функции по команде 'U'.

Таблица 2.8: Коды ответов от МК.

Код	Значение
0	Подтверждение
1	Отказ. Неверный пароль
2	Неизвестная команда
3	Максимальное количество бесед в сутки не установлено
4	Корректор времени для GMT не установлен
5	Время не синхронизировано
6	Не удалось вычислить конец суток
8	Максимальное количество действий в сутки уже установлено
9	Устанавливаемый корректор времени превышает допустимый максимум. Допустимый максимум равен 43200.
10	Корректор времени уже установлен. Ошибка повторной установки при имеющемся корректоре

11	Не удалось установить корректор при первичной инициализации.
12	Корректор времени не установлен
13	Время окончания лицензии уже установлено
14	Время окончания лицензии не установлено
15	Пароль уже установлен
16	Не установлен пароль в EEPROM
17	Серийный номер уже установлен
18	Серийный номер не установлен
21	Максимально положительная накрутка не установлена
22	Максимально отрицательная накрутка не установлена
23	Слишком много данных. При команде от ПК, количество данных оказалось больше их размера
24	Выход за пределы размера БД.
25	Значение БД не существует. Ошибка при перезаписи элемента БД

Таблица 2.9: Коды причин форматирования flash памяти

Код	**ASCII**	**Причина**
65	A	Неверный L фьюз - конфигурационный байт
66	B	Неверный H фьюз - конфигурационный байт
67	C	Неверный E фьюз- конфигурационный байт
68	D	Неверный LOCK - блокировочный байт
69	E	Неверно введен пароль N раз
70	F	Команда форматирования от пользователя
71	G	Лицензия истекла
72	H	Превышено число положительных накруток времени
73	I	Превышено число отрицательных накруток времени
74	J	Превышено число неверных команд в сутки
75	K	Аппаратный сброс МК. Замыкание ножки RST на GND
76	L	Понижение питания МК. Срабатывание подсистемы Brown-out Detection(BOD)

2.5. Жизненный цикл ключа

1. Пустой микроконтроллер

Плата с загрузчиком по умолчанию из магазина.

2. Ключ.

Микроконтроллер с ключом и возможностью установить серийный номер, пароль и другие инициализационные параметры. Для дальнейшей реализации.

3. Отформатированный ключ.

Форматирование происходит по 2м причинам.

а) Окончание лицензии.

б) Память отформатирована по причине запрещенного действия.

3.ТЕСТИРОВАНИЕ

3.1. Среда разработки тестов. Язык Processing

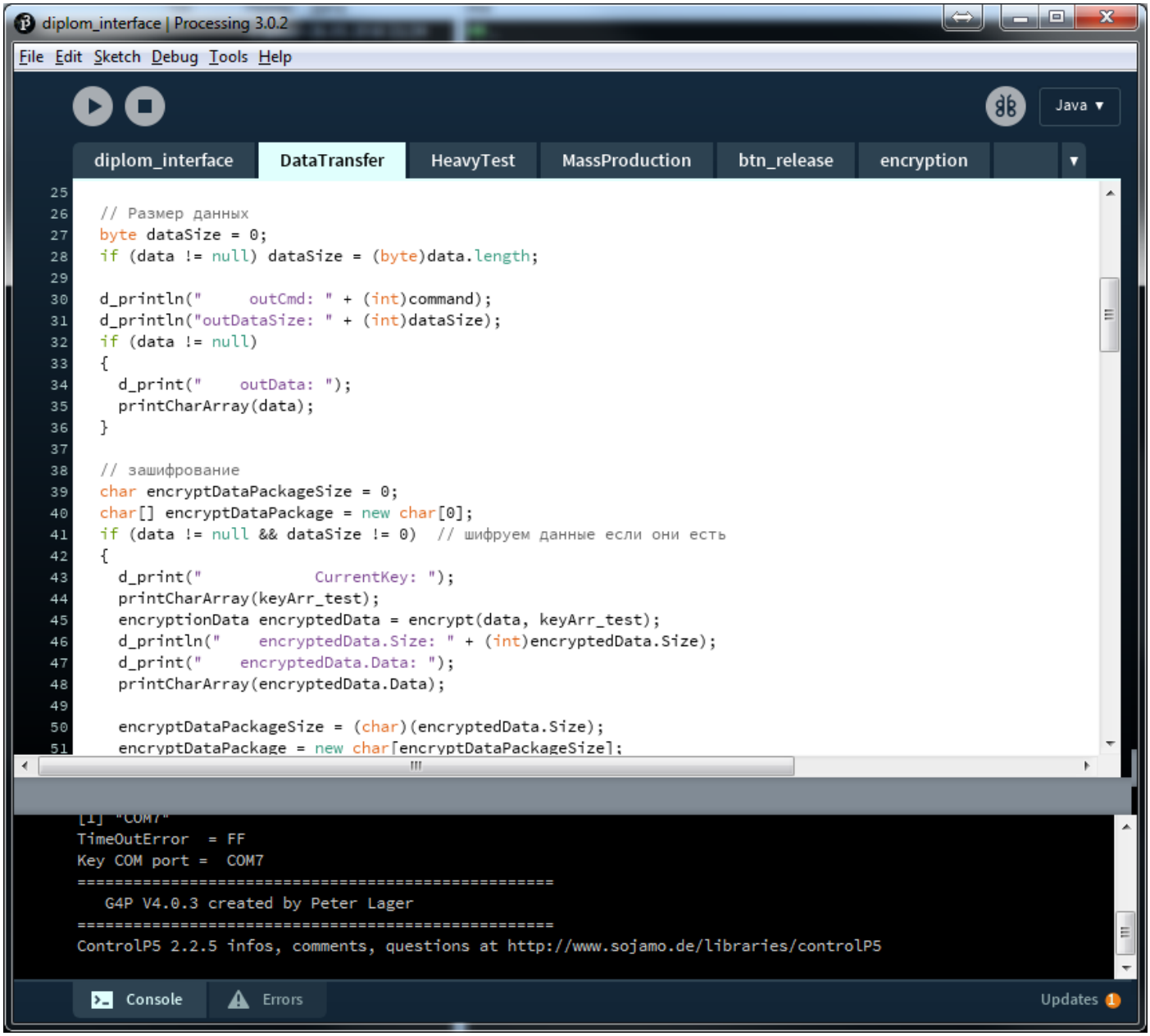


Рис. 3.1: Среда разработки Processing

Processing[9] — открытый язык программирования, основанный на Java[10]. Представляет собой лёгкий и быстрый инструментарий для людей, которые хотят программировать изображения, анимацию и интерфейсы.

Используется студентами, художниками, дизайнерами, исследователями и любителями, для изучения, прототипирования и производства. Он создан для изучения основ компьютерного программирования в визуальном контексте и служит альбомным программным обеспечением (имеется в виду то, что каждый *.pde файл визуальной оболочки Processing'а представляет собой отдельное изображение или анимацию, и т. д.) и профессиональным производственным инструментом.

Особенности языка:

- Программы на Processig называются скетчами. Идея состоит в том, чтобы программирование в Java-стиле было похоже на скриптирование для быстрого написания кода;
- Каждый скетч является классом, наследуемым от Java-класса PApplet, который перед выполнением преобразуется в код на Java для выполнения в требуемой среде;
- Все создаваемые классы являются внутренними классами основного;

3.1. Программа тестирования

Рис. 3.2: Интерфейс разработанной среды тестирования

Ключевым моментом в разработки данного программного обеспечения стало создание необходимой системы тестирования, позволяющей провести более точную отладку всех частей разработанной программы. Отладка программы проходила в 2 этапа:

1. Проверка написанного кода с помощью эмулятора среды Atmel Studio;
2. Проверка взаимодействия клиента и хоста с помощью разработанной системы тестирования;

Разработанная среда тестирование поддерживает следующие функции:

- Проверка инициализации устройства;
- Тестирование счетчика реального времени;
- Тестирование установки конфигураций микроконтроллера;
- Тестирование системы зашифрования и расшифрования;
- Поддержка единичных тестов, так и в автоматическое тестирование в течение длительного времени;
- Тестирование работы с базой данных;
- Проверка системы защиты;

- И как следствие, тестирование всех функций и алгоритмов программы;

3.2. Примеры проводимых тестов

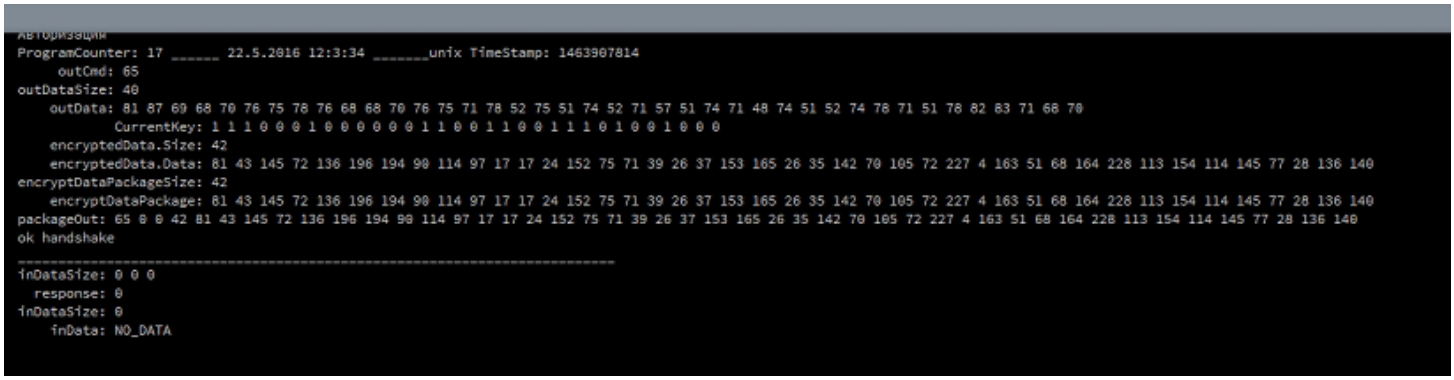


Рис. 3.3: Разовое тестирование авторизации устройства

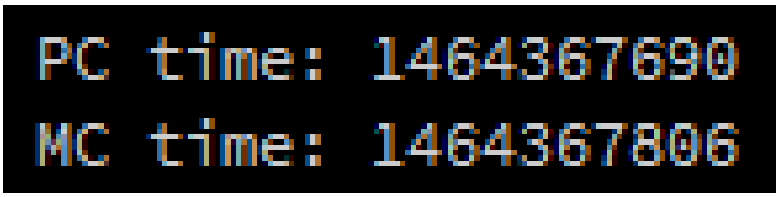


Рис. 3.4: Тестирование вывода времени МК в формате uinx timestamp

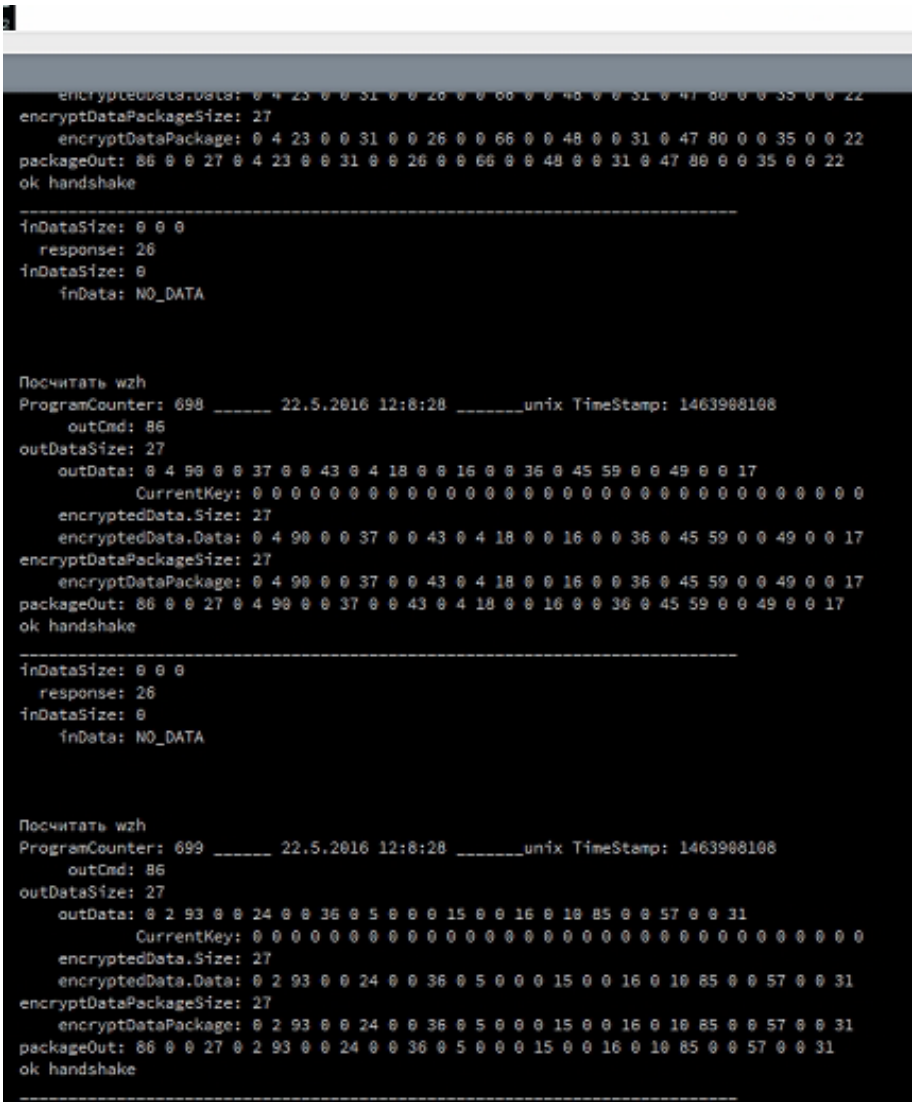


Рис. 3.5: Автоматическое тестирование

Заключение

Главным результатом проделанной работы является полностью готовый к выходу на рынок продукт «Программно-аппаратная платформа с возможностью защищенного хранения информации и защищенными протоколами передачи информации» на базе 8-битных микроконтроллеров ATmega.

В ходе выполнения данной исследовательской работы были получены следующие новые навыки:

1. Рассмотрены архитектура и устройство микроконтроллеров семейства ATmega;
2. Изучены вектора атак и различные способы взлома МК, а также методы защиты;
3. Получены углубленные знания в области программирования микроконтроллеров на язык avr C, а также языков высокого уровня Java;
4. Исследованы разработанные алгоритмы защиты данных и программно-аппаратные методы защиты от взлома;
5. Создано взаимодействие системы вида «хост – клиент»;
6. Изучена и применена методика тестирования программного обеспечения;
7. На собственном опыте пройдены основные этапы разработки программного обеспечения от этапа постановки задачи до внедрения на рынок;

Список литературы

1. Atmel Co. Datasheet Atmega328p [электронный ресурс]// Официальный сайт компании: http://www.atmel.com //URL: http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf (Дата обращения: 22.05.2016);
2. Atmel Co. Datasheet Atmega32u4 [электронный ресурс]// Официальный сайт компании: http://www.atmel.com //URL: http://www.atmel.com/images/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf (Дата обращения: 22.05.2016);
3. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. — М.: Издательский дом «Додэка-XXI», 2007. — 592 с: ил. (Серия «Программируемые системы»);
4. Мортон Дж. Микроконтроллеры AVR. Вводный курс. /Пер. с англ. – М.: Издательский дом «Додэка-XXI», 2006. – 272 с.: ил. (Серия «Мировая электроника»);
5. Sergei Skorobogatov Security, Reliability and Backdoors [электронный ресурс]// Официальный сайт компании: http://www.cl.cam.ac.uk/~sps32/ URL: http://www.cl.cam.ac.uk/~sps32/SG_talk_SRB.pdf (Дата обращения: 22.05.2016);
6. Sergei Skorobogatov Breakthrough silicon scanning discovers backdoor in military chip [электронный ресурс]// Официальный сайт компании: http://www.cl.cam.ac.uk/~sps32/ URL: http://www.cl.cam.ac.uk/~sps32/ches2012-backdoor.pdf (Дата обращения: 22.05.2016);
7. Sergei Skorobogatov Hardware Security of Semiconductor Chips: Progress and Lessons [электронный ресурс]// Официальный сайт компании: http://www.cl.cam.ac.uk/~sps32/ URL: http://www.cl.cam.ac.uk/~sps32/NCL_2011.pdf (Дата обращения: 22.05.2016);
8. Arduino Co. Reference [электронный ресурс]// Официальный сайт компании: https://www.arduino.cc/ // URL: https://www.arduino.cc/en/Reference/HomePage (Дата обращения: 22.05.2016);
9. Processing Co. Reference [электронный ресурс]// Официальный сайт компании: https://processing.org/ // URL: https://processing.org/reference/ (Дата обращения: 22.05.2016);
10. Oracle Technology Network Java Documentation [электронный ресурс]// Официальный сайт компании: https:// www.oracle.com/ // URL: https:// www.oracle.com/technetwork/java/javase/documentation/index.html (Дата обращения: 22.05.2016);

Приложение 1. Листинг программы для микроконтроллера на avr C

Файл BootLoader.h

```
#ifndef _CDC_H_
#define _CDC_H_

	//  Includes:
		#include <avr/io.h>
		#include <avr/wdt.h>
		#include <avr/eeprom.h>
		#include <avr/power.h>
		#include <avr/interrupt.h>
		#include <util/delay.h>
		#include "Descriptors.h"
		#include <LUFA/Drivers/USB/USB.h>
		#include "DongleCommon.h"
	//	Macros:
		#define FLASH_PAGE_COUNT FLASHEND / SPM_PAGESIZE

		#define CPU_PRESCALE(n)	(CLKPR = 0x80, CLKPR = (n))
		#define LED_SETUP()		DDRC |= (1<<7); DDRB |= (1<<0); DDRD |= (1<<5);
		#define L_LED_OFF()		PORTC &= ~(1<<7)
		#define L_LED_ON()		PORTC |= (1<<7)
		#define L_LED_TOGGLE()	PORTC ^= (1<<7)

	// Безопасная запись памяти EEPROM
	#define eeprom_write_byte_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_byte(addr, val); }
	#define eeprom_write_word_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_word(addr, val); }
	#define eeprom_write_dword_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_dword(addr, val); }
		#define TX_LED_OFF()		PORTD |= (1<<5)
		#define TX_LED_ON()		PORTD &= ~(1<<5)
		#define RX_LED_OFF()		PORTB |= (1<<0)
		#define RX_LED_ON()		PORTB &= ~(1<<0)
	//	Function Prototypes:
		void initialization_func(void);		//	Инициализация
		void CDC_Task(void);
		void ApplicationStart(void);				//	Старт прикладной программы
		bool FormatAndCheck(void);//форматирование с проверкой байта причины в EEPROM и
проверкой успешного форматирования
		void FlashFormatting(void); // Форматирование flash
		bool CheckFormat(void);//Проверка успешного форматирования flash
		void EEPROM_erase_withOut_range(uint16_t start, uint16_t end);	// Форматировать EEPROM
исключая участок со start по end включительно
		void SendFormattingReason(void);	//	Отправить на ПК причину форматирования
		void split_int(uint32_t num, char * c, uint8_t * size);	// получить массив char из числа
		bool ReseivePrepare(void);			//	Подготовка устройстсва к приёму данных
		void blink(void);					//	Моргнуть
		void SendByteArray(uint8_t * data, uint8_t size);	// отправить массив байт на ПК

		void SendPacket(uint8_t dataSize, uint8_t response, uint8_t * data);
		void EVENT_USB_Device_ConfigurationChanged(void);
		static uint8_t FetchNextCommandByte(void);
		static void	WriteNextResponseByte(const uint8_t Response);
		void save_mcusr_inf(void);	// информация о флагах сброса
		void sendReason(void);		// отправить причину
		void sendSN(void);			// отправить серийный номер
		void sendTime(void);		// отправить время последней синхронизации
#endif
```

Файл BootLoader.c

```
//		В загрузчике оставить только следующие функциии
//		1) форматирования по считанному биту из EEPROM
//		2) Считывание причины форматирования из EEPROM
#define  INCLUDE_FROM_CATERINA_C
#include "BootLoader.h"
static CDC_LineEncoding_t LineEncoding = { .BaudRateBPS = 0,
                                .CharFormat  = CDC_LINEENCODING_OneStopBit,
                                .ParityType  = CDC_PARITY_None,
                                .DataBits    = 8                            };
static uint16_t unixTimeStampEEPROMaddr = 10;		// адрес времени синхронизации
static uint16_t serialNumEEPROMaddr = 15;		// адрес серийного номера в EEPROM
static uint16_t formattingMarkEEPROMaddr = 30;		// адрес метки форматирования
uint8_t sizeSerNum = 10;		// размер серийного номера
////////////////////////////////////////////////////////////
// Адреса причин перезагрузки в EEPROM
static uint16_t EXTRF_EEPROMaddr = 170;		// аппаратный ресет
static uint16_t BORF_EEPROMaddr = 171;		// ресет по снижению питания
static uint16_t PORF_EEPROMaddr = 172;		// ресет по пдключению питания
static bool ApplicationIsExists = false;  // Наличие приложения по адресу 0x0000

int main(void)
{
	save_mcusr_inf();
	initialization_func();
	if (pgm_read_word(0) != 0xFFFF) ApplicationIsExists = true;
	while (true)
	{	// проверить флаг форматирования и отформатировать если он установлен
		if(FormatAndCheck() == true)
		{ blink(); }
		if (ApplicationIsExists) ApplicationStart();		// Перейти в приложение если оно есть
		USB_USBTask();
		CDC_Task();
	}
}

void initialization_func()
{
	wdt_disable();							// отключить сторожевой таймер
	clock_prescale_set(clock_div_1);		// Отключить деление тактовой частоты

	// Перенос таблицы векторов прерываний в секцию загрузчика
	MCUCR = (1 << IVCE);
	MCUCR = (1 << IVSEL);
	LED_SETUP();
	RX_LED_OFF();
	TX_LED_OFF();
	CPU_PRESCALE(0);
	OCR1AH = 0;
	OCR1AL = 250;
	TIMSK1 = (1 << OCIE1A);	// enable timer 1 output compare A match interrupt
	TCCR1B = ((1 << CS11) | (1 << CS10));		// 1/64 prescaler on timer 1 input
	USB_Init();	// Initialize USB Subsystem
	sei();	// Enable global interrupts so that the USB stack can function
}

bool ReseivePrepare()
```

```
{
	Endpoint_SelectEndpoint(CDC_RX_EPNUM);
	if (!(Endpoint_IsOUTReceived()))
	  return false;

	return true;
}

void CDC_Task(void)
{
	if (ReseivePrepare() != true) return;
	uint8_t command = FetchNextCommandByte();
	switch(command)
	{
		default:
			SendFormattingReason();
			break;
	}
}

void SendFormattingReason()
{
	sendReason();
	sendSN();
	sendTime();
}

void sendReason()
{
	uint8_t formatting_mark = eeprom_read_byte ((uint8_t *)formattingMarkEEPROMaddr);
	uint8_t reasonArr_size = 9;
	char reasonArr[9] = "Reason: ";
	reasonArr[8] = formatting_mark;
	uint8_t rByteArr[reasonArr_size];
	for (uint8_t i = 0; i < reasonArr_size; i++) rByteArr[i] = reasonArr[i];
	SendByteArray(rByteArr, reasonArr_size);
}

void sendSN()
{
	uint8_t sArr_size = 17;
	char sArr[17] = ". S/N: ";
	uint8_t sn_arr_EEPROM[sizeSerNum];
	eeprom_read_block(sn_arr_EEPROM, (uint8_t*)serialNumEEPROMaddr, sizeSerNum);

	for(uint8_t i = 0; i < sizeSerNum; i++)
	{
		if (sn_arr_EEPROM[i] != 0xFF)
			sArr[i + 7] = sn_arr_EEPROM[i];
		else
			sArr[i + 7] = ' ';
	}

	uint8_t snByteArr[sArr_size];
	for (uint8_t i = 0; i < sArr_size; i++) snByteArr[i] = sArr[i];

	SendByteArray(snByteArr, sArr_size);
}
```

```
void sendTime()
{
	uint8_t strTime_size = 19;
	char strTime[19] = ". Time: ";
	uint32_t time = eeprom_read_dword((uint32_t*)unixTimeStampEEPROMaddr);
	char time_CharArr[10];
	uint8_t time_size = 0;
	split_int(time, time_CharArr, (uint8_t*)&time_size );
	for(uint8_t i = 0; i < 10; i++)	strTime[i + 8] = time_CharArr[i];
	strTime[18] = '\r';
	uint8_t timeByteArr[strTime_size];
	for(uint8_t i = 0; i < strTime_size; i++) { timeByteArr[i] = strTime[i]; }
	SendByteArray(timeByteArr, strTime_size);
}

void split_int(uint32_t num, char * c, uint8_t * size)
{
	uint32_t n = num;
	uint32_t numQty = 0;
	// посчитать кол-во цифр в числе
	while (n != 0) { n /= 10; numQty++; }
	*size = numQty;

	for (uint8_t i = 0; i < numQty; i++)
	{
		c[i] = (char)((num % 10) + 48);
		num = num / 10;
	}
	char t;
	//инвертируем массив символов
	for (uint8_t i = 0; i < numQty / 2; i++)
	{
		t = c[i];
		c[i] = c[numQty - 1 - i];
		c[numQty - 1 - i] = t;
	}
}

void save_mcusr_inf()
{
	if ( MCUSR & ( 1 << EXTRF ))		//внешний сброс
	{
		MCUSR &=  ~( 1 << EXTRF );		// сбросить флаг
		eeprom_write_byte_safe((uint8_t*)EXTRF_EEPROMaddr, 1);
	}
	else
		eeprom_write_byte_safe((uint8_t*)EXTRF_EEPROMaddr, 0);

	if (MCUSR & ( 1 << BORF ))		// сброс по BOD
	{
		MCUSR &= ~( 1 << BORF);		// сбросить флаг
		eeprom_write_byte_safe((uint8_t*)BORF_EEPROMaddr, 1);
	}
	else
		eeprom_write_byte_safe((uint8_t*)BORF_EEPROMaddr, 0);

	if( MCUSR & ( 1 << PORF ))		// сброс по питанию
	{
```

```
		MCUSR &=  ~( 1 << PORF );   // сбросить флаг
		eeprom_write_byte_safe((uint8_t*)PORF_EEPROMaddr, 1);
	}
	else
		eeprom_write_byte_safe((uint8_t*)PORF_EEPROMaddr, 0);

	MCUSR = 0;    // очистить все флаги сброса
}

// Проверить, есть ли причина форматирования и если есть, то форматировать.
bool FormatAndCheck()
{
	uint8_t formatting_mark = eeprom_read_byte ((uint8_t *)formattingMarkEEPROMaddr);
	if (formatting_mark == 0xff) return false; // возвращаемся если флаг не установлен
	bool chk = CheckFormat();
	if (chk == true) return true;
	while (CheckFormat() != true)
	{ FlashFormatting(); }
	EEPROM_erase_withOut_range(5, 31);
	return true;
}

bool CheckFormat()
{
	uint32_t page = 0;
	for (int i = 0; i < FLASHEND; i++)
	{
		page = i * 0x80;
		if (page == BOOT_START_ADDR)
			break;
		uint8_t byte = pgm_read_byte_near(i);
		if(byte != 0xff)
			return false;
	}
	return true;
}

void FlashFormatting()
{
	uint8_t sreg = SREG;
	cli();

	uint32_t page = 0;
	for (uint32_t i = 0; i < FLASH_PAGE_COUNT; i++)
	{
		page = i * 0x80;
		if (page == BOOT_START_ADDR)
		break;
		eeprom_busy_wait ();
		boot_page_erase_safe (page);
		boot_spm_busy_wait ();      // Wait until the memory is erased.
	}
	boot_rww_enable ();
	// Re-enable interrupts (if they were ever enabled).
	SREG = sreg;
}

void EEPROM_erase_withOut_range(uint16_t start, uint16_t end)
```

```
{
	for (uint16_t i = 0; i < E2END + 1; i++)
	{
		if(start != 0 || end != 0)
		{
			if (i >= start && i <= end)
			continue;
		}
		eeprom_write_byte_safe ((uint8_t*)i, 0xff);
	}
}

void blink()
{
	TX_LED_ON();
	_delay_ms(100);
	TX_LED_OFF();
	_delay_ms(100);
}

void SendByteArray(uint8_t * data, uint8_t size)
{
	Endpoint_SelectEndpoint(CDC_TX_EPNUM);
	for(uint8_t i = 0; i < size; i++) WriteNextResponseByte(data[i]);
	// Remember if the endpoint is completely full before clearing it
	bool IsEndpointFull = !(Endpoint_IsReadWriteAllowed());
	// Send the endpoint data to the host
	Endpoint_ClearIN();
// If a full endpoint's worth of data was sent, we need to send an empty packet afterwards to signal end of transfer
	if (IsEndpointFull)
	{
		while (!(Endpoint_IsINReady()))
		{
			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return;
		}
		Endpoint_ClearIN();
	}
	// Wait until the data has been sent to the host
	while (!(Endpoint_IsINReady()))
	{
		if (USB_DeviceState == DEVICE_STATE_Unattached)
		  return;
	}

	// Select the OUT endpoint
	Endpoint_SelectEndpoint(CDC_RX_EPNUM);

	// Acknowledge the command from the host
	Endpoint_ClearOUT();
}

static uint8_t FetchNextCommandByte(void)
{
	// Select the OUT endpoint so that the next data byte can be read
	Endpoint_SelectEndpoint(CDC_RX_EPNUM);
	// If OUT endpoint empty, clear it and wait for the next packet from the host
	while (!(Endpoint_IsReadWriteAllowed()))
```

```
	{
		Endpoint_ClearOUT();

		while (!(Endpoint_IsOUTReceived()))
		{
			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return 0;
		}
	}
	// Fetch the next byte from the OUT endpoint
	return Endpoint_Read_8();
}

static void WriteNextResponseByte(const uint8_t Response)
{
	// Select the IN endpoint so that the next data byte can be written
	Endpoint_SelectEndpoint(CDC_TX_EPNUM);

	// If IN endpoint full, clear it and wait until ready for the next packet to the host
	if (!(Endpoint_IsReadWriteAllowed()))
	{
		Endpoint_ClearIN();

		while (!(Endpoint_IsINReady()))
		{
			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return;
		}
	}

	// Write the next byte to the IN endpoint
	Endpoint_Write_8(Response);
}

/** Event handler for the USB_ConfigurationChanged event. This configures the device's endpoints ready
 *  to relay data to and from the attached USB host.
 */
void EVENT_USB_Device_ConfigurationChanged(void)
{
	// Setup CDC Notification, Rx and Tx Endpoints
	Endpoint_ConfigureEndpoint(CDC_NOTIFICATION_EPNUM, EP_TYPE_INTERRUPT,
				ENDPOINT_DIR_IN, CDC_NOTIFICATION_EPSIZE,
				ENDPOINT_BANK_SINGLE);

	Endpoint_ConfigureEndpoint(CDC_TX_EPNUM, EP_TYPE_BULK,
				ENDPOINT_DIR_IN, CDC_TXRX_EPSIZE,
				ENDPOINT_BANK_SINGLE);

	Endpoint_ConfigureEndpoint(CDC_RX_EPNUM, EP_TYPE_BULK,
				ENDPOINT_DIR_OUT, CDC_TXRX_EPSIZE,
				ENDPOINT_BANK_SINGLE);
}

// Event handler for the USB_ControlRequest event. This is used to catch and process control requests sent to
//  the device from the USB host before passing along unhandled control requests to the library for processing
//  internally.
//
void EVENT_USB_Device_ControlRequest(void)
```

```
{
	// Ignore any requests that aren't directed to the CDC interface
	if ((USB_ControlRequest.bmRequestType & (CONTROL_REQTYPE_TYPE |
CONTROL_REQTYPE_RECIPIENT)) !=
	   (REQTYPE_CLASS | REQREC_INTERFACE))
	{
		return;
	}

	// Process CDC specific control requests
	switch (USB_ControlRequest.bRequest)
	{
		case CDC_REQ_GetLineEncoding:
			if (USB_ControlRequest.bmRequestType == (REQDIR_DEVICETOHOST |
REQTYPE_CLASS | REQREC_INTERFACE))
			{
				Endpoint_ClearSETUP();

				// Write the line coding data to the control endpoint
				Endpoint_Write_Control_Stream_LE(&LineEncoding,
sizeof(CDC_LineEncoding_t));
				Endpoint_ClearOUT();
			}

			break;
		case CDC_REQ_SetLineEncoding:
			if (USB_ControlRequest.bmRequestType == (REQDIR_HOSTTODEVICE |
REQTYPE_CLASS | REQREC_INTERFACE))
			{
				Endpoint_ClearSETUP();

				// Read the line coding data in from the host into the global struct
				Endpoint_Read_Control_Stream_LE(&LineEncoding,
sizeof(CDC_LineEncoding_t));
				Endpoint_ClearIN();
			}
			break;
	}
}
void ApplicationStart(void)
{
	cli();
	//Сброс TIMER1 и счетчика перед запуском скетча
	TIMSK1 = 0;
	TCCR1B = 0;
	// Перенос таблицы векторов прерываний в раздел приложения
	MCUCR = (1 << IVCE);
	MCUCR = 0;
	//перейти к началу пространства приложений
	__asm__ volatile("jmp 0x0000");
}
ISR(TIMER1_COMPA_vect, ISR_BLOCK)
{
	// Reset counter
	TCNT1H = 0;
	TCNT1L = 0;
}
```

Файл Caterina.h

```
#ifndef _CDC_H_
#define _CDC_H_

	/* Includes: */
		#include <avr/io.h>
		#include <stdlib.h>
		#include <stdio.h>
		#include <math.h>
		#include <avr/wdt.h>
		#include <avr/eeprom.h>
		#include <avr/power.h>
		#include <avr/interrupt.h>
		#include <util/delay.h>
		#include <avr/boot.h>
		#include <inttypes.h>
		#include <avr/pgmspace.h>
		#include <util/atomic.h>
		#include "Descriptors.h"
		#include <LUFA/Drivers/USB/USB.h>

		#define BOOTSTARTADDR 0x3800 * 2

	/*Random stuff*/
		#define MAX_DICE 255

	/* Macros: */
		#define ABS(x) ((x) < 0 ? -(x) : (x))

		#define FLASH_PAGE_COUNT FLASHEND / SPM_PAGESIZE

		#define CPU_PRESCALE(n)	(CLKPR = 0x80, CLKPR = (n))
		#define LED_SETUP()		DDRC |= (1<<7); DDRB |= (1<<0); DDRD |= (1<<5);

		// Безопасная запись памяти EEPROM
	#define eeprom_write_byte_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_byte(addr, val); }
	#define eeprom_write_word_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_word(addr, val); }
	#define eeprom_write_dword_safe(addr, val)	{ eeprom_busy_wait(); eeprom_write_dword(addr, val); }

		#define TX_LED_OFF()	PORTD |= (1<<5)
		#define TX_LED_ON()	PORTD &= ~(1<<5)
		#define RX_LED_OFF()	PORTB |= (1<<0)
		#define RX_LED_ON()		PORTB &= ~(1<<0)

/******Main Functions*****/
int main(void);
bool ReseivePrepare(void);
void pc_mc_connection(void);
bool RequirementCMD(uint8_t cmd, uint8_t* pDataArray, uint8_t DataArraySize);// Функции для установки требуемых параметров
void MainWorkingCMD(uint8_t cmd, uint8_t* pDataArray, uint8_t DataArraySize);// Основные рабочие функции
/******Time Functions*****/
void SendPlusTimeMistake(void);		// Отправить на ПК сумму положительных накруток в секундах
void SendCurrTimeFromRegister(void);// Отправить текущее время МК на ПК
void SendMinusTimeMistake(void);	// Отправить на ПК сумму отрицательных накруток в секундах
void SendSynchroTimeFromEEPROM(void);// Отправить на ПК время синхронизации записанное в EEPROM
void TimeSynchronization(uint8_t * pDataArray, uint8_t DataArraySize);// Синхронизировать время с ПК
```

```
void ReceivedSynchroTimeToEEPROM(uint8_t * pDataArray, uint8_t DataArraySize);
void SendMaxPositive(void);// Отправить на ПК допустимую максимальную положительную накрутку
void SendMaxNegative(void);// Отправить на ПК допустимую максимальную отрицательную накрутку

/******Help Functions*****/
void Initialization(void); // инициализация системы
void get_data_from_buffer(uint8_t* pBuffer, uint8_t* pArray,int ArSize);
void check_up_down_time_mistake(uint32_t PC_time); // проверить накрутки и отформатировать если накрутки превысили максимальные значения
uint32_t JoinArray(uint8_t *mass, uint8_t size); // соединить массив байт в одно число uint32_t
void SplitArray(uint32_t timeStamp, uint8_t *mass, uint8_t size); // Разъединить массив на байты
void split_int(uint32_t num, char * c, uint8_t * size);      // Преобразовать число в массив char[]


/********Format_Flash*****/
void SetFormattingMarkAndReboot(uint8_t reason); // Записать причину форматирования в EEPROM и перезагрузить МК для форматирования в загрузочной области

/**********USART**********/
void send_package_ENCRYPT(uint32_t dataSize, uint8_t response, uint8_t * data);// подготовить зашифрованный пакет к отправке
void send_package_NO_ENCRYPT(uint32_t dataSize, uint8_t response, uint8_t * data);// подготовить открытый пакет к отправке
void CreateAndSendPackage(uint8_t response, uint32_t dataSize, uint8_t * data);// Создать пакет и передать его в функцию SendByteArray
void SendByteArray(uint8_t size, uint8_t * data);// Отправить пакет на ПК
void SendOneByte(uint8_t bt);  // отправить 1 байт
static uint8_t FetchNextCommandByte(void);
static void WriteNextResponseByte(const uint8_t Response);

/*Encryption & Decryption*/
void GenerateKey(void);// Сгенерировать новый случайный ключ
bool GenerateAndUseDefaultKey(void);// Сгенерировать ключ по умолчанию
void encryptArray(uint8_t* EncryptData, uint32_t* EncryptDataSize, uint8_t* inputData, uint32_t dataSize); // Зашифровать
void decryptArray(uint8_t* EncryptData, uint32_t EncryptDataSize, uint8_t* DecryptData, uint32_t* DecryptDataSize);// Расшифровать

/**********USB**********/
void EVENT_USB_Device_ConfigurationChanged(void);
void EVENT_USB_Device_ControlRequest(void);

void reboot(void); // Перезагрузить МК
void SetPassword(uint8_t *pswd); // установить пароль
bool CheckPassword(uint8_t *pswd);// проверить пароль
void write_serial_num(uint8_t * serialNumArr, uint8_t DataArraySize);// Записать серийный номер
void send_serial_num(void); // Отправить серийный номер на ПК
void Send_SN_forMan(void);// Отправить серийный номер в человеко понятном виде. Для терминала.
void SetEndLicenseTimeToEEPROM(uint8_t * timeArr); // Установить время окончания лицензии в EEPROM
void SendEndLicenseTimeFromEEPROM(void);// Отправить на ПК время окончания лицензии из EEPROM
void CheckLicenseTime(void);// проверить время окончания лицензии. Если закончилась, то форматирование
void Action(void);// Обследование/беседа с проверяемым
void SetActionMaxQuontity(uint8_t * qty);// Установить максимальный размер бесед в сутки. Если = 0, то без ограничений
void SetGMT_Corrector(uint8_t * corrector);// Установить корректор времени в EEPROM
void SendGMT_Corrector(void);// Отправить корректор времени, который установлен для часового пояса
void SendActionMaxQuontity(void);// Отправить на ПК максимальное количество бесед в сутки
void SendRemainCurrentQuontityAction(void);// Отправить оставшееся количество бесед за текущие сутки
```

```
bool IncrementCurrentActionCounter(void);// Увеличить счетчик бесед в EEPROM за текущий день.
uint32_t ComputeNewEndDayUTS_and_SetInEEPROM(void);// вычислить конец текущих суток и установить в EEPROM
void CheckFuseAndLock(void);// Проверить FUSE и LOCK биты. Конфигурационные и блокировочные
bool CheckRequirement(void);  // Проверить, установлены ли требуемые инициализационные настройки

////////////// База данных ////////////
void DB_ReadAll(void);// Считать всю БД
void DB_SetOneVal(uint8_t * data, uint8_t DataSize);// Записать 1 значение в БД по указанному индексу
void DB_GetOneVal(uint8_t * data, uint8_t DataSize);// Считать 1 значение из БД по указанному индексу

////////// Контроль пинов ////////////
void example(void);
void get_hist_max_min(char PortName, char PinVoltage, uint8_t HistNumb, uint8_t PointsNumb);
void get_histogram (char PortName, char PinVoltage, uint16_t* HistPoints, uint8_t PointsNumb, uint8_t PointTimeStep, uint8_t ChargeInterv);
void find_max_min(uint16_t* pHistogram, uint8_t HistNumb, uint8_t PointsNumb, uint16_t* MaxArray, uint16_t* MinArray);
uint8_t ADC_get_value(char adc_input);

//// Вычисление по формуле //////////
void compute_f_wzh(uint8_t* pDataArray, uint8_t DataArraySize); // запустить вычисления и отправить результат на ПК
double compute_wzh_Array(uint8_t * wzhArr, uint8_t fCount);  // Посчитать значения F0, F1..Fn
double computeFurmula_whz(uint32_t w, uint32_t z, uint32_t h); // Посчитать формулу

//// Проверка флагов перезагрузки ////
void CheckRSTflags(void); // проверка флагов перезагрузки

#endif
```

Файл Caterina.c

```
#define  INCLUDE_FROM_CATERINA_C
#include "Caterina.h"

static CDC_LineEncoding_t LineEncoding = { .BaudRateBPS = 0,
                                .CharFormat  = CDC_LINEENCODING_OneStopBit,
                                .ParityType  = CDC_PARITY_None,
                                .DataBits    = 8                    };

////////////////////// Ответные команды ////////////////////////////////////
static uint8_t confirmation =                   0;      // подтверждение
static uint8_t deny =                           1;      // отказ. Не верный пароль
static uint8_t unknownCommand =                 2;      // неизвестная команда
static uint8_t maxQtyActionIsNotExists =        3;      // Максимальное количество бесед в сутки не установлено
static uint8_t noGMT_Corrector =                4;      // Корректор времени для GMT не установлен
static uint8_t timeNotSync =                    5;      // Время не синхронизировано
static uint8_t badEndDay =                      6;      // Не удалось вычислить конец суток
static uint8_t maxActionQtyIsExists =   8;// Максимальное количество действий в сутки уже установлено
static uint8_t timeCorrectorExceedsMaximum = 9;// Устанавливаемый корректор времени превышает
допустимый максимум
static uint8_t timeCorrecorIsExists = 10;       // Корректор времени уже установлен. Ошибка повторной
установки при имеющемся корректоре
static uint8_t timeCorrectorNotInstalled = 11;// Не удалось установить корректор при первичной инициализации.
static uint8_t timeCorrectorIsNotExists = 12;   // Корректор времени не установлен
static uint8_t endLicTimeIsExists =     13;     // Время окончания лицензии уже установлено
static uint8_t endLicTimeIsNotExists = 14;      // Время окончания лицензии не установлено
static uint8_t passwordIsExists =       15;     // Пароль уже установлен
static uint8_t passwordIsNotSet =       16;     // Не установлен пароль в EEPROM
static uint8_t serialNumIsExists =      17;     // Серийный номер уже установлен
static uint8_t serialNumIsNotExists =   18;     // Серийный номер не установлен
static uint8_t MaxPositiveIsNotExists = 21;     // Максимально положительная накрутка не установлена
static uint8_t MaxNegativeIsNotExists =22; // Максимально отрицательная накрутка не установлена
static uint8_t tooMuchData =            23; // Слишком много данных. При команде от ПК, количество данных
оказалось больше их размера
static uint8_t dbOutOfRange =           24; // Выход за пределы размера БД.
static uint8_t dbValIsNotExists =       25; // Значение БД не существует. Ошибка при перезаписи элемента БД
static uint8_t badWZHsize =             26; // Не верное количество данных для вычисления wzh
static uint8_t serialNumBadSize =       27; // Слишком большой серийный номер. Максимум 10.
static uint8_t timeoutError =                                   0xff;// ошибка зависания
////////////////////////////////////////////////////////////////////////////
static uint8_t NO_DATA[0]; // пустой массив для команд, которые ничего не должны возвращать

////////////////////////// Счетчик несуществующих команд //////////////////////
static uint8_t UnknownCMDcounter = 0; // Счетчик
static uint8_t MaxUnknownCMDinDay = 5; // Максимальное количество не верных команд в сутки
static uint16_t UnknownCMDcounterEEPROMaddr = 40; // Адрес счетчика не верных команд

//////////////////////////////// Работа с паролем ////////////////////////////
static bool IsAuthorized = false; // Пользователь прошел авторизацию?
static uint8_t PasswordSize = 40; // Размер пароля
static uint16_t PasswordStartAddr = 100; // Стартовый адрес расположения пароля
static uint8_t NumberOfLogins = 1; // Количество попыток авторизации
static uint8_t MaxNumberOfLogins = 10; // Максимальное количество авторизаций

////////////////////////////// Причины форматирования ////////////////////////
static uint8_t bad_L_FUSE =             65;     // Не верный L - конфигурационный байт
static uint8_t bad_H_FUSE =             66;     // Не верный H - конфигурационный байт
```

```
static uint8_t bad_E_FUSE =                 67;     // Не верный E - конфигурационный байт
static uint8_t bad_LOCK_BITS =              68;     // Не верный LOCK - блокировочный байт
static uint8_t badPassword =                69;     // Не верно введен пароль N раз
static uint8_t formatCommand =              70;     // Команда форматирования от пользователя
static uint8_t endLicense =                 71;     // Лицензия истекла
static uint8_t timeHackPlus       =         72;     // Превышено число положительных накруток времени
static uint8_t timeHackMinus =              73;     // Превышено число отрицательных накруток времени
static uint8_t badCMDhack =                 74;     // Превышено число не верных команд в сутки
static uint8_t EXTRF_isSet =                75;     // Аппаратный сброс МК. Замыкание ножки RST на GND
static uint8_t BORF_isSet =                 76;     // Понижение питания МК. Срабатывание подсистемы
Brown-out Detection(BOD)

//////////////////////////////// Подсчет и ограничение бесед ////////////////////////////////
static uint16_t ActionCounterEEPROMaddr = 80;                   // Адрес счетчика бесед в EEPROM
static uint16_t ActionMaxQuontityEEPROMaddr = 81;       // Адрес максимального кол-ва бесед в EEPROM
static uint16_t GMT_CorrectorEEPROMaddr = 85;                   // адресс корректора в EEPROM
static uint16_t ActionEndDayUTS_EEPROMaddr = 90;        // Адрес расположения конца суток в секундах
static uint16_t GMT_CorrectorMaximim = 43200;       // Максимальный корректор. Минимальный = 0.
static uint32_t DayInSec = 86400;                                       // количество секунд в сутках
bool ActionQtyExhausted = true;         // Количество действий в сутки исчерпано

//////////////////////////////// Конфигурационные и блокировочные биты ///////////////////////
uint8_t RELEASE_L_FUSE = 0xFF;
uint8_t RELEASE_H_FUSE = 0xD8;
uint8_t RELEASE_E_FUSE = 0xC8;    //
uint8_t RELEASE_LOCK_BITS = 0xFC;

///// Не форматируемые данные EEPROM
static uint16_t unixTimeStampEEPROMaddr = 10;       // адрес времени синхронизации
static uint16_t serialNumEEPROMaddr = 15;           // адрес серийного номера в EEPROM
static uint16_t formattingMarkEEPROMaddr = 30;      // адрес метки форматирования

//////////////////////////////// Время ////////////////////////////////////////////////////////
bool FirstTimeSync = false;             // Флаг самой первой синхронизации времени
bool TimeSynchronized = false;          // синхронизированно ли время?
static uint32_t unixTimeStamp = 0;      // На старте МК время берем из EEPROM
static uint8_t UTSarrSize = 5;          // длина массива времени
static uint8_t UTSarr[5];               // массив времени
bool nextTick = false;                  // переменная для обозначения тика в 1 сек.
static uint16_t mc_Sec = 997;           // Количество милисекунд в секунде микроконтроллера
static uint16_t timeCounter = 1;        // счетчик милисекунд.
uint8_t correctorValue = 2;             // Величина корректировки. То есть если = N, то к каждой Nй секунде
добавляем 1 милисекунду
uint8_t timeCorrectorCounter = 1;       // Счетчик корректора времени

////////////////////////////////////////////////////////////////////////////////////////////////
// Работа с серийным номером
uint8_t sizeSerNum = 10;        // размер серийного номера

// Работа с временем жизни лицензии
static uint32_t EndLicenseTime = 0;     // Время окончания лицензии
static uint16_t LicenseTime_EEPROMaddr = 200; // Адрес окончания лицензии

//////////////// База данных ////////////////////////////////
uint16_t DataBaseEEPROMAddr = 250;
uint8_t dbSize = 32;
uint32_t g_KeyRequestCounter = 0;       // счетчик запроса ключа
```

```
const uint16_t PlusTimeMistakeEEPROM_addr = 150;
const uint16_t MinusTimeMistakeEEPROM_addr = 155;
uint32_t PlusTimeMistake = 0;  // положительная накрутка в секундах.
uint32_t MinusTimeMistake = 0;          // отрицательная накрутка в секундах.
static uint32_t MaxPositive = 0;// максимальное количество положительных накруток
static uint32_t MaxNegative = 0;          // максимальное количество отрицательных накруток
static uint8_t MaxPositivePercent = 102;
static uint8_t MaxNegativePercent = 2;
uint16_t MaxPositiveEEPROMaddr = 70;
uint16_t MaxNegativeEEPROMaddr = 74;
//////////////////////////////////////////////////////////////
// Адреса причин перезагрузки в EEPROM
static uint16_t EXTRF_EEPROMaddr = 170;              // аппаратный ресет
static uint16_t BORF_EEPROMaddr = 171;               // ресет по снижению питания
static uint16_t PORF_EEPROMaddr = 172;               // ресет по пдключению питания
bool isInit = false;                                 // флаг инициализации времени для проверки флагов

// Шифрование
uint8_t g_TimeKeySize = 32;
uint8_t g_TimeKey[32] = {  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};   // присвоение заводского ключа

bool keyIsDefault = false;
int main(void)
{
        Initialization();
        while (true)
        {
                CheckLicenseTime();
                CheckRSTflags();
                pc_mc_connection();     // Время выполнения должно быть менее 30 мс.

                USB_USBTask();// Задача USB должна вызываться не реже 30 мс в режиме устройства. Задача
должна работать всегда.
        }
        return 0;
}

bool CheckRequirement()
{
        // Не разглашается
}

/********Проверка флагов перезагрузки******************/
void CheckRSTflags()
{
        // если время еще не синхронизировали и его нет в EEPROM, то проверку на флаги не проводить
        if (!isInit) return;

        uint8_t EXTRF_value = eeprom_read_byte((uint8_t*)EXTRF_EEPROMaddr);
        if (EXTRF_value == 1)
                SetFormattingMarkAndReboot(EXTRF_isSet);

        uint8_t BORF_value = eeprom_read_byte((uint8_t*)BORF_EEPROMaddr);
        uint8_t PORF_value = eeprom_read_byte((uint8_t*)PORF_EEPROMaddr);
```

```
	if(BORF_value == 1 && PORF_value == 0)
		SetFormattingMarkAndReboot(BORF_isSet);
}

void Send_SN_forMan()
{
	uint8_t strSNsize = 15;
	char strSN[15] = "S/N: ";

	uint8_t sn = eeprom_read_byte((uint8_t*) serialNumEEPROMaddr);
	if (sn == 0xFF)
	{
		char str_noSN[14] = "SN: undefined.";
		uint8_t no_SN[14];
		for(uint8_t i = 0; i < 14; i++)
			no_SN[i] = str_noSN[i];

		SendByteArray(14, no_SN);
		return;
	}

	uint8_t sn_arr_EEPROM[sizeSerNum];
	eeprom_read_block(sn_arr_EEPROM, (uint8_t*)serialNumEEPROMaddr, sizeSerNum);

	for(uint8_t i = 0; i < sizeSerNum; i++) strSN[i + 5] = sn_arr_EEPROM[i];

	uint8_t outArr[strSNsize];
	for(uint8_t i = 0; i < strSNsize; i++)	{ outArr[i] = strSN[i]; }

	SendByteArray(strSNsize, outArr);
}

void split_int(uint32_t num, char * c, uint8_t * size)
{
	uint32_t n = num;
	uint32_t numQty = 0;

	// посчитать кол-во цифр в числе
	while (n != 0) { n /= 10; numQty++; }

	*size = numQty;

	for (uint8_t i = 0; i < numQty; i++)
	{
		c[i] = (char)((num % 10) + 48);
		num = num / 10;
	}

	char t;
	//инвертируем массив символов
	for (uint8_t i = 0; i < numQty / 2; i++)
	{
		t = c[i];
		c[i] = c[numQty - 1 - i];
		c[numQty - 1 - i] = t;
	}
}
```

```
/*********MAIN_FUNCTIONS************************************************************/
void pc_mc_connection(void)
{
	if (ReseivePrepare() != true) return;	// подготовка к приёму

	uint8_t firstByte = FetchNextCommandByte();
	if (firstByte == timeoutError) { SendOneByte(timeoutError); return; }

	uint8_t HANDSHAKE = 'Z';
	switch(firstByte)
	{
		case 'Z':
			SendOneByte(HANDSHAKE);
			break;

		case 's':
				Send_SN_forMan();	// отправить человекопонятный S/N
				return;
			break;

		default:
			return;
	}

	// НЕ шифруемая команда. 0-вой байт
	uint8_t command = FetchNextCommandByte();
	if (command == timeoutError) { SendOneByte(timeoutError); return; }

	if (command == 'Y')	// сбросить ключ на дефолтный если пароль существует
	{
		if(GenerateAndUseDefaultKey())
			send_package_ENCRYPT(0, confirmation, NO_DATA);
		else
			send_package_ENCRYPT(0, passwordIsNotSet, NO_DATA);
		return;
	}

	uint8_t dsArrSize = 3;
	uint8_t dataSizeArray[dsArrSize];

	// 1й, 2й, 3й байты - размер
	for(uint8_t i = 0; i < dsArrSize; i++)
	{
		dataSizeArray[i] = FetchNextCommandByte();
		if (dataSizeArray[i] == timeoutError) { SendOneByte(timeoutError); return;}
	}

	uint32_t dataSize = JoinArray(dataSizeArray, dsArrSize); // вычислить размер

	uint32_t EncryptDataSize = dataSize;
	uint8_t EncryptData[EncryptDataSize];

	if (EncryptDataSize != 0)
	{
		for(int i = 0; i < EncryptDataSize; i++)
			EncryptData[i] = FetchNextCommandByte();
	}
```

```
	// Игнорировать данные, которые по количеству выходят за рамки размера данных
	// Проверить. Есть ли в буфере еще входные данные
	uint8_t trashByte = FetchNextCommandByte();
	if(trashByte != 0xFF)
	{
		send_package_ENCRYPT(0, tooMuchData, NO_DATA);
		return;
	}

	uint8_t DecryptData[255];
	uint32_t DecryptDataSize = 0;
	for (uint16_t k = 0; k < 255; k++) DecryptData[k] = 0;   // ОБЯЗАТЕЛЬНО! Нужно обнулять этот массив т.к. из-за особонности avr Си он хранит предыдущий мусор.

	decryptArray(EncryptData, EncryptDataSize, DecryptData, (uint32_t*)&DecryptDataSize);
	uint8_t data[DecryptDataSize];                                  //N byte == Read received data
	if (DecryptDataSize != 0)
	{
		for(int i = 0; i < DecryptDataSize; i++)
		{
			data[i] = DecryptData[i];
		}
	}

	// функции, которые установят требуемые параметры
	if(RequirementCMD(command, data, DecryptDataSize))		// если команда из этого списка, то
		return;  // выходим, что бы не отсылать больше пакетов

	if(CheckRequirement()) // если требуемые установки сделаны, то разрешить главные команды
		MainWorkingCMD(command, data, DecryptDataSize); // основные рабочие команды. Сюда попадаем, только если требуемые настройки установленны, а иначе не проходит проверка выше и на ПК отправляется ошибка
	else
		return;
}

bool RequirementCMD(uint8_t cmd, uint8_t* pDataArray, uint8_t DataArraySize)
{
	switch(cmd)
	{
		// 1) Синхронизация времени, запрос текущего
		case '+':
				TimeSynchronization(pDataArray, DataArraySize);
			break;

		case '?':
				SendCurrTimeFromRegister();
			break;

		case 'I':
				SendSynchroTimeFromEEPROM();
			break;

		case 'F':
				SendPlusTimeMistake();
			break;

		case 'M':
```

```
                SendMinusTimeMistake();
        break;

case 'D':
                SendMaxPositive();
        break;

case 'K':
                SendMaxNegative();
        break;

// 2,3) Установка пароля. Авторизацию вынесли в главные ф-ции
case 'P':
                SetPassword(pDataArray);
        break;

// 4) Установка серийного номера
case 'W':
                write_serial_num(pDataArray, DataArraySize);
        break;

case 'S':
                send_serial_num();
        break;

// 5) Установка времени окончания лицензии
case 'T':
                SetEndLicenseTimeToEEPROM(pDataArray);
        break;

case 'E':
                SendEndLicenseTimeFromEEPROM();
        break;


// 6) Установка максимального количества бесед в сутки //////////
case 'H':
                SetActionMaxQuontity(pDataArray);
        break;

case 'G':
                SendActionMaxQuontity();
        break;

case 'R':
                SendRemainCurrentQuontityAction();
        break;

// 7) Установка корректора часового пояса
case 'Q':
                SetGMT_Corrector(pDataArray);
        break;

case 'J':
                SendGMT_Corrector();
        break;

////////////////////////////////////
```

```
		case '4':
				GenerateKey();
			break;

		case 'N': //Format memory
				SetFormattingMarkAndReboot(formatCommand);	// установить флаг форматирования перезагрузить МК для форматирования из загрузочной области
			break;

		default:
// Если попали сюда, то никакой пакет на ПК не отправился и команда не из этого списка либо не существует
// Если команда из рабочего списка (т.е. сейчас ничего не отправляли на ПК), то true и можно делать проверку
// То есть если команда из инициализационного списка, то не надо делать проверку на установку требуемых параметров
		return false;	// true обозначает что команда не из этого списка
	}

	return true;
}

void MainWorkingCMD(uint8_t cmd, uint8_t* pDataArray, uint8_t DataArraySize)
{
	switch (cmd)
	{
		case 'V':
			compute_f_wzh(pDataArray, DataArraySize);
			break;

		case 'A':
				// Авторизация. Ожидаем пароль из PasswordSize байт
				if(CheckPassword(pDataArray))
				{
					IsAuthorized = true;
					send_package_ENCRYPT(0, confirmation, NO_DATA);
				}
				else
				{
					if (NumberOfLogins == MaxNumberOfLogins)
						SetFormattingMarkAndReboot(badPassword);

					IsAuthorized = false;
					send_package_ENCRYPT(0, deny, NO_DATA);
					NumberOfLogins++;
				}

				Action();
			break;


		case 'U':
				Action();
			break;

		case '5':
				DB_ReadAll();
			break;

		case '6':
```

```
                        DB_SetOneVal(pDataArray, DataArraySize);
                    break;

                case '7':
                        DB_GetOneVal(pDataArray, DataArraySize);
                    break;

        default:
                    // Подсчет не верных команд
                    send_package_ENCRYPT(0, unknownCommand, NO_DATA);
                    UnknownCMDcounter++;
                    eeprom_write_byte_safe((uint8_t*) UnknownCMDcounterEEPROMaddr,
UnknownCMDcounter);
                    if (UnknownCMDcounter == MaxUnknownCMDinDay - 1)
                        SetFormattingMarkAndReboot(badCMDhack);

          break;
      }
}

void CheckLicenseTime()
{
        if (unixTimeStamp > EndLicenseTime)
            SetFormattingMarkAndReboot(endLicense);
}

bool ReseivePrepare()
{
        Endpoint_SelectEndpoint(CDC_RX_EPNUM);
        if (!(Endpoint_IsOUTReceived()))
          return false;

        return true;
}

/**********TIMER_FUNCTIONS**********************************************************/
void TimeSynchronization(uint8_t * pDataArray, uint8_t DataArraySize)
{
        //Received current Time to register
        //Receive data
        timeCounter = 1;        // сброс счетчика т.к. синхронизация
        cli();

        uint32_t pc_time = JoinArray(pDataArray, UTSarrSize);


        if (FirstTimeSync)
        {
            unixTimeStamp = pc_time;
            FirstTimeSync = false;
        }

        //inside this function we save received time, if it's less then current
        check_up_down_time_mistake(pc_time);

        unixTimeStamp = pc_time; // Save received time in SRAM reg
```

```
	// Запись времени в EEPROM раз в 3 часа. Что-бы не исчерпать количество перезаписи
	uint32_t UTS_eeprom = eeprom_read_dword((uint32_t*)unixTimeStampEEPROMaddr);

	if (UTS_eeprom == 0xFFFFFFFF)
		eeprom_write_dword_safe(((uint32_t*)unixTimeStampEEPROMaddr), unixTimeStamp);

	if ((unixTimeStamp - UTS_eeprom) >= 10800)
		eeprom_write_dword_safe(((uint32_t*)unixTimeStampEEPROMaddr), unixTimeStamp);

	sei();

	TimeSynchronized = true;

	send_package_ENCRYPT(0, confirmation, NO_DATA);
}

void CheckFuseAndLock()
{
	uint8_t L_FUSE;
	uint8_t H_FUSE;
	uint8_t E_FUSE;
	uint8_t LOCK_BITS;

	L_FUSE =	boot_lock_fuse_bits_get(GET_LOW_FUSE_BITS);
	H_FUSE =	boot_lock_fuse_bits_get(GET_HIGH_FUSE_BITS);
	E_FUSE =	boot_lock_fuse_bits_get(GET_EXTENDED_FUSE_BITS);
	LOCK_BITS = boot_lock_fuse_bits_get(GET_LOCK_BITS);

	if(L_FUSE != RELEASE_L_FUSE)		SetFormattingMarkAndReboot(bad_L_FUSE);
	if(H_FUSE != RELEASE_H_FUSE)		SetFormattingMarkAndReboot(bad_H_FUSE);
	if(E_FUSE != RELEASE_E_FUSE)		SetFormattingMarkAndReboot(bad_E_FUSE);
	if(LOCK_BITS != RELEASE_LOCK_BITS)	SetFormattingMarkAndReboot(bad_LOCK_BITS);
}

//////////////////////////////// Беседы и ограничение ////////////////////////////////////////
void Action()
{
	bool incResult = false;
	incResult = IncrementCurrentActionCounter();
}

bool IncrementCurrentActionCounter()
{
	uint8_t ActionMaxQuontity = eeprom_read_byte((uint8_t *)ActionMaxQuontityEEPROMaddr);
	// Если максимальное количество = 0 или 0xff, то безлимитно
	if (ActionMaxQuontity == 0 || ActionMaxQuontity == 0xff)
	{
		//send_package_ENCRYPT(0, confirmation, NO_DATA);
		return true;
	}

	// Проверить. Если находимся в новых сутках, то
	// 1) обнулить счётчик
	// 2) определить и записать границу следующих суток в EEPROM
	// Иначе ничего не делать.

	uint32_t EndDayUTS = eeprom_read_dword((uint32_t*) ActionEndDayUTS_EEPROMaddr);  // читаем
текущее время конца суток
```

```
	// если время конца суток не установлено, то вычисляем и устанавливаем в EEPROM
	if (EndDayUTS == 0xFFFFFFFF)
		EndDayUTS = ComputeNewEndDayUTS_and_SetInEEPROM();

	if (EndDayUTS == 0)
		return false;

	// сравниваем с текущим UTS
	if( unixTimeStamp > EndDayUTS)	// если уже новые сутки
	{
		// обнуляем счетчик в EEPROM
		eeprom_write_byte_safe((uint8_t *)ActionCounterEEPROMaddr, 0);
		EndDayUTS = ComputeNewEndDayUTS_and_SetInEEPROM();
	}

	uint8_t ActionCounter = eeprom_read_byte((uint8_t *)ActionCounterEEPROMaddr);
	uint8_t NewActionCounter = 0;

	// Если превысили количество бесед за сутки
	if (ActionCounter == ActionMaxQuontity)
	{
		ActionQtyExhausted = true;
		//send_package_ENCRYPT(0, ActionQtyExhausted, NO_DATA);
		return false;
	}
	else
	{
		ActionQtyExhausted = false;
	}

	// если за текущие сутки только начали считать беседы
	if(ActionCounter == 0xff)
	{
		NewActionCounter = 1;
	}
	else
		NewActionCounter = ActionCounter + 1;

	eeprom_write_byte_safe((uint8_t *)ActionCounterEEPROMaddr, NewActionCounter);

	ActionCounter = eeprom_read_byte((uint8_t *)ActionCounterEEPROMaddr);

	uint8_t currRemainingQty[1];
	currRemainingQty[0] = ActionCounter;
	if (ActionCounter == NewActionCounter)
		return true;
	else
		return false;
}

uint32_t ComputeNewEndDayUTS_and_SetInEEPROM()
{
	uint16_t unixTimeStamp_Mod_DayInSec = 0;
	uint16_t delta = 0;
	uint32_t GMT_Corrector = 0;
	uint32_t EndDayUTS = 0;
```

```
	GMT_Corrector = eeprom_read_dword((uint32_t*) GMT_CorrectorEEPROMaddr);
	if(GMT_Corrector == 0xFFFFFFFF)
	{
		send_package_ENCRYPT(0, noGMT_Corrector, NO_DATA); // корректор не установлен
		return 0;
	}

	if (unixTimeStamp < DayInSec)
	{
		send_package_ENCRYPT(0, timeNotSync, NO_DATA);		// время не синхронизировано
		return 0;
	}


	unixTimeStamp_Mod_DayInSec = unixTimeStamp % DayInSec;		// получили кол-во прошедших
секунд за текущие сутки
	delta = DayInSec - unixTimeStamp_Mod_DayInSec - GMT_Corrector; // получить дельту, сколько нужно
прибавить к текущему дню, что бы получить полночь.
	EndDayUTS = unixTimeStamp + delta;

	if (EndDayUTS == 0)
	{
		send_package_ENCRYPT(0, badEndDay, NO_DATA);
		return false;
	}

	eeprom_write_dword_safe((uint32_t*)ActionEndDayUTS_EEPROMaddr, EndDayUTS);

	return EndDayUTS;
}

void SetActionMaxQuontity(uint8_t * qty)
{
	uint8_t newQty = qty[0];

	uint8_t maxQty = eeprom_read_byte((uint8_t *)ActionMaxQuontityEEPROMaddr);
	if (maxQty == 0xFF)
	{
		eeprom_write_byte_safe((uint8_t *)ActionMaxQuontityEEPROMaddr, newQty);

		maxQty = eeprom_read_byte((uint8_t * )ActionMaxQuontityEEPROMaddr);
		if (maxQty != 0xFF)
			send_package_ENCRYPT(0, confirmation, NO_DATA);
		else
			send_package_ENCRYPT(0, maxQtyActionIsNotExists, NO_DATA);
	}
	else
		send_package_ENCRYPT(0, maxActionQtyIsExists, NO_DATA);
}

void SendActionMaxQuontity()
{
	uint8_t maxQty = eeprom_read_byte((uint8_t *)ActionMaxQuontityEEPROMaddr);
	uint8_t currentQtyArr[1];

	if(maxQty != 0xFF)
	{
		currentQtyArr[0] = maxQty;
```

```
		send_package_ENCRYPT(1, confirmation, currentQtyArr);
	}
	else
		send_package_ENCRYPT(0, maxQtyActionIsNotExists, NO_DATA);
}

void SetGMT_Corrector(uint8_t * corrector)
{
	uint32_t GMT_Corrector = 0;
	GMT_Corrector = JoinArray(corrector, 3);
	if (GMT_Corrector > GMT_CorrectorMaximim)
	{
		send_package_ENCRYPT(0, timeCorrectorExceedsMaximum, NO_DATA);
		return;
	}

	GMT_Corrector = eeprom_read_dword((uint32_t*) GMT_CorrectorEEPROMaddr);
	// Установить корректор единожды
	if(GMT_Corrector != 0xFFFFFFFF)
	{
		send_package_ENCRYPT(0, timeCorrecorIsExists, NO_DATA);
		return;
	}

	GMT_Corrector = JoinArray(corrector, 3);
	eeprom_write_dword_safe((uint32_t * )GMT_CorrectorEEPROMaddr, GMT_Corrector);

	GMT_Corrector = eeprom_read_dword((uint32_t * ) GMT_CorrectorEEPROMaddr);
	if(GMT_Corrector != 0xFFFFFFFF)
		send_package_ENCRYPT(0, confirmation, NO_DATA);
	else
		send_package_ENCRYPT(0, timeCorrectorNotInstalled, NO_DATA);
}

void SendGMT_Corrector()
{
	uint32_t GMT_Corrector = eeprom_read_dword((uint32_t * )GMT_CorrectorEEPROMaddr);

	if(GMT_Corrector != 0xFFFFFFFF)
	{
		uint8_t GMT_CorrectorArray[3];
		SplitArray(GMT_Corrector, GMT_CorrectorArray, 3);
		send_package_ENCRYPT(3, confirmation, GMT_CorrectorArray);
	}
	else
		send_package_ENCRYPT(0, timeCorrectorIsNotExists, NO_DATA);
}

void SendRemainCurrentQuontityAction()
{
	uint8_t ActionCounter = 0;
	uint8_t ActionMaxQuontity = 0;
	// Вычтем из максимального числа текущще
	if (eeprom_read_byte((uint8_t *)ActionMaxQuontityEEPROMaddr) != 0xff)
		ActionMaxQuontity = eeprom_read_byte((uint8_t *)ActionMaxQuontityEEPROMaddr);

	if (eeprom_read_byte((uint8_t *)ActionCounterEEPROMaddr) != 0xff)
		ActionCounter = eeprom_read_byte((uint8_t *)ActionCounterEEPROMaddr);
```

```
	uint8_t RemainQty = ActionMaxQuontity - ActionCounter;
	uint8_t RemainQtyArr[1];
	RemainQtyArr[0] = RemainQty;
	send_package_ENCRYPT(1, confirmation, RemainQtyArr);
}

//////////////////////////////// Время лицензии ////////////////////////////////////////////////
void SetEndLicenseTimeToEEPROM(uint8_t * timeArr) // Установить время окончания лицензии в EEPROM
{
	uint32_t licTime = eeprom_read_dword((uint32_t*) LicenseTime_EEPROMaddr);

	if (licTime == 0xFFFFFFFF)
	{
		// Записать время окончания лицензии
		licTime = JoinArray(timeArr, UTSarrSize);
		eeprom_write_dword_safe	((uint32_t*) LicenseTime_EEPROMaddr, licTime);

		uint32_t checkLicTime = eeprom_read_dword((uint32_t*) LicenseTime_EEPROMaddr);
		if(checkLicTime != licTime)
		{
			send_package_ENCRYPT(0, endLicTimeIsNotExists, NO_DATA);
			return;
		}

		EndLicenseTime = licTime;

		uint32_t licDuration = EndLicenseTime - unixTimeStamp;

		// Записать максимальное количество положительной накрутки
		MaxPositive = (licDuration / 100) * MaxPositivePercent;
		eeprom_write_dword_safe((uint32_t*)MaxPositiveEEPROMaddr, MaxPositive);

		uint32_t checkMaxPositive = eeprom_read_dword((uint32_t*)MaxPositiveEEPROMaddr);
		if(checkMaxPositive != MaxPositive)
		{
			send_package_ENCRYPT(0, MaxPositiveIsNotExists, NO_DATA);
			return;
		}

		// Записать максимальное количество отрицательной накрутки
		MaxNegative = (licDuration / 100) * MaxNegativePercent;
		eeprom_write_dword_safe((uint32_t*)MaxNegativeEEPROMaddr, MaxNegative);

		uint32_t checkMaxNegative = eeprom_read_dword((uint32_t*)MaxNegativeEEPROMaddr);
		if(checkMaxNegative != MaxNegative)
		{
			send_package_ENCRYPT(0, MaxNegativeIsNotExists, NO_DATA);
			return;
		}

		send_package_ENCRYPT(0, confirmation, NO_DATA);
	}
	else
		send_package_ENCRYPT(0, endLicTimeIsExists, NO_DATA);  // устанавливать время окончания
лицензии можно только 1 раз
}
```

```
void SendEndLicenseTimeFromEEPROM()                // Отправить на пк время окончания лицензии
{
	uint8_t endLicenseTimeArray[UTSarrSize];
	uint32_t endLicTime = eeprom_read_dword((uint32_t*) LicenseTime_EEPROMaddr);

	if (endLicTime != 0xFFFFFFFF)
	{
		SplitArray(endLicTime, endLicenseTimeArray, UTSarrSize);
		send_package_ENCRYPT(UTSarrSize, confirmation, endLicenseTimeArray);
	}
	else
		send_package_ENCRYPT(0, endLicTimeIsNotExists, NO_DATA);
}

////////////////////////////// Пароль ////////////////////////////////////////////////////////////
void SetPassword(uint8_t *pswd)
{
	uint8_t pswdSetByte = eeprom_read_byte((uint8_t *)PasswordStartAddr);
	if(pswdSetByte != 0xff) // Пароль можно установить только 1 раз
	{
		send_package_ENCRYPT(0, passwordIsExists, NO_DATA);
		return;
	}

	for(int i = 0; i < PasswordSize; i++)
		eeprom_write_byte_safe((uint8_t *)i + PasswordStartAddr, pswd[i]);

	if(CheckPassword(pswd))
	{
		send_package_ENCRYPT(0, confirmation, NO_DATA);
		GenerateAndUseDefaultKey(); // Сгенерировать ключ по умолчанию если пароль установлен
	}
	else
		send_package_ENCRYPT(0, passwordIsNotSet, NO_DATA);
}

bool CheckPassword(uint8_t *pswd)
{
	for(int i = 0; i < PasswordSize; i++)
	{
		uint8_t pswdByte = eeprom_read_byte ((uint8_t *)i + PasswordStartAddr);
		if (pswdByte != pswd[i])
			return false;
	}
	return true;
}

////////////////////////////////////////////////////////////////////////////////
void SendCurrTimeFromRegister()
{
	SplitArray(unixTimeStamp, UTSarr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, UTSarr);
}

void SendPlusTimeMistake()
{
	uint8_t PlusTimeMistakeArr[UTSarrSize];
	PlusTimeMistake = eeprom_read_dword((uint32_t*)PlusTimeMistakeEEPROM_addr);
```

```
	if (PlusTimeMistake == 0xFFFFFFFF) PlusTimeMistake = 0;

	SplitArray(PlusTimeMistake, PlusTimeMistakeArr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, PlusTimeMistakeArr);
}

void SendMinusTimeMistake()
{
	uint8_t MinusTimeMistakeArr[UTSarrSize];
	MinusTimeMistake = eeprom_read_dword((uint32_t*)MinusTimeMistakeEEPROM_addr);
	if (MinusTimeMistake == 0xFFFFFFFF) MinusTimeMistake = 0;

	SplitArray(MinusTimeMistake, MinusTimeMistakeArr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, MinusTimeMistakeArr);
}

void SendSynchroTimeFromEEPROM()
{
	uint32_t UTS_eeprom = eeprom_read_dword((uint32_t*)unixTimeStampEEPROMaddr);
	uint8_t UTS_eeprom_arr[UTSarrSize];
	SplitArray(UTS_eeprom, UTS_eeprom_arr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, UTS_eeprom_arr);
}


void SendMaxPositive()
{
	uint32_t l_maxPositive = eeprom_read_dword((uint32_t*) MaxPositiveEEPROMaddr);
	if (l_maxPositive == 0xFFFFFFFF)
	{
		send_package_ENCRYPT(0, MaxPositiveIsNotExists, NO_DATA);
		return;
	}

	uint8_t l_maxPositiveArr[UTSarrSize];
	SplitArray(l_maxPositive, l_maxPositiveArr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, l_maxPositiveArr);
}

void SendMaxNegative()
{
	uint32_t l_maxNegative = eeprom_read_dword((uint32_t*) MaxNegativeEEPROMaddr);
	if(l_maxNegative == 0xFFFFFFFF)
	{
		send_package_ENCRYPT(0, MaxNegativeIsNotExists, NO_DATA);
		return;
	}

	uint8_t l_maxNegativeArr[UTSarrSize];
	SplitArray(l_maxNegative, l_maxNegativeArr, UTSarrSize);
	send_package_ENCRYPT(UTSarrSize, confirmation, l_maxNegativeArr);
}

//////////////////// База данных ////////////////////////
void DB_ReadAll()
{
	uint8_t dataBase[dbSize];
	eeprom_read_block(dataBase, (uint8_t*)DataBaseEEPROMAddr, dbSize);
```

```
	send_package_ENCRYPT(dbSize, confirmation, dataBase);
}

void DB_SetOneVal(uint8_t * data, uint8_t DataSize)
{
	uint8_t idx = data[0];
	uint8_t val = data[1];

	if (idx < 0 || idx > dbSize)
	{
		send_package_ENCRYPT(0, dbOutOfRange, NO_DATA);
		return;
	}

	eeprom_write_byte_safe((uint8_t*)DataBaseEEPROMAddr + idx, val);
	uint8_t checkVal = eeprom_read_byte((uint8_t*)DataBaseEEPROMAddr + idx);
	if (checkVal != 0xFF)
		send_package_ENCRYPT(0, confirmation, NO_DATA);
	else
		send_package_ENCRYPT(0, dbValIsNotExists, NO_DATA);
}

void DB_GetOneVal(uint8_t * data, uint8_t DataSize)
{
	uint8_t idx = data[0];

	if (idx > dbSize)
	{
		send_package_ENCRYPT(0, dbOutOfRange, NO_DATA);
		return;
	}

	uint8_t val = eeprom_read_byte((uint8_t*)DataBaseEEPROMAddr + idx);
	if (val == 0xFF)
	{
		send_package_ENCRYPT(0, dbValIsNotExists, NO_DATA);
		return;
	}

	uint8_t valArr[1];
	valArr[0] = val;
	send_package_ENCRYPT(1, confirmation, valArr);
}


//Вычисление по формуле
void compute_f_wzh(uint8_t* wzh_array, uint8_t wzh_arraySize)
{ // Не разглашается }

double compute_wzh_Array(uint8_t * wzhArr, uint8_t fCount)
{ // Не разлглашается }

double computeFurmula_whz(uint32_t w, uint32_t z, uint32_t h)
{ // Не разлглашается }


void reboot()
{
```

```
        wdt_enable(WDTO_250MS);
    for (;;);
}

/************************HELP_FUNCTIONS**************/
void Initialization()
{
        //Start initialization
        // Reset some registers
                MCUSR = 0;                                                      //reset all reset flags
                wdt_disable();                                                  //disable watchdog timer
                clock_prescale_set(clock_div_1);        //disable clock division

        // Start led signal
                LED_SETUP();
                RX_LED_OFF();
                TX_LED_OFF();
                CPU_PRESCALE(0);

        // Timer initialization
                OCR1AH = 0;
                OCR1AL = 250;

        // Comparing with A Interruption
                TIMSK1 = (1 << OCIE1A);

        // Set 1/64 prescale
                TCCR1B = ((1 << CS11) | (1 << CS10));

        // Enable global interruption flag
                sei();


        unixTimeStamp = eeprom_read_dword((uint32_t*)unixTimeStampEEPROMaddr);
        if (unixTimeStamp == 0xFFFFFFFF)
        {
                FirstTimeSync = true;

                isInit = false;
        }
        else
                isInit = true;

        EndLicenseTime = eeprom_read_dword((uint32_t*) LicenseTime_EEPROMaddr);

        CheckFuseAndLock();

        MaxPositive = eeprom_read_dword((uint32_t*) MaxPositiveEEPROMaddr);
        MaxNegative = eeprom_read_dword((uint32_t*) MaxNegativeEEPROMaddr);

        UnknownCMDcounter = eeprom_read_byte((uint8_t*) UnknownCMDcounterEEPROMaddr);
        if (unixTimeStamp == 0xFF) UnknownCMDcounter = 0;

        // инициализировать БД нулями если там 0xFF
        for(uint8_t i = 0; i < dbSize; i++)
        {
                uint8_t checkDbByte = eeprom_read_byte((uint8_t*) DataBaseEEPROMAddr + i);
                if (checkDbByte == 0xFF)
```

```
                eeprom_write_byte_safe((uint8_t*) DataBaseEEPROMAddr + i, 0);
        }

        GenerateAndUseDefaultKey(); // Сгенерировать ключ по умолчанию если пароль установлен
}

void get_data_from_buffer(uint8_t* pBuffer, uint8_t* pArray, int ArSize)
{
        for(int i = 0; i < ArSize; i++)
        {
                *pArray = *pBuffer;
                pArray++;
                pBuffer++;
        }
}

uint32_t JoinArray(uint8_t * mass, uint8_t size)
{
        uint32_t joinTS = 0;
        uint32_t multiplier = 1;
        int i;
        int x;

        for (x = 1; x < size; x++)
    multiplier *= 100;

        for (i = 0; i < size; i++)
        {
                if (i != 0) multiplier /= 100;
                joinTS += mass[i] * multiplier;
        }

        return joinTS;
}

void SplitArray(uint32_t timeStamp, uint8_t * mass, uint8_t size)
{
        uint32_t divisor = 1;
        uint32_t subtrahend;
        uint32_t tmpTime;
        uint32_t time = timeStamp;

        for (int i = 1; i < size; i++)
                divisor *= 100;

        for (uint8_t i = 0; i < size; i++)
        {
                if (i != 0)
                {
                        subtrahend = mass[i - 1] * divisor;
                        time -= subtrahend;
                        divisor /= 100;
                }

                tmpTime = time / divisor;
                mass[i] = tmpTime;
        }
}
```

```
void check_up_down_time_mistake(uint32_t PC_time)
{
	if (PC_time > unixTimeStamp)
	{
		uint32_t sumPlusTimeMistake = eeprom_read_dword((uint32_t*)PlusTimeMistakeEEPROM_addr);
// Считать сумму положительных накруток из EEPROM
		if(sumPlusTimeMistake == 0xFFFFFFFF) sumPlusTimeMistake = 0;

		PlusTimeMistake = PC_time - unixTimeStamp; // посчитали текущую положительную накрутку
		sumPlusTimeMistake += PlusTimeMistake;

		// Запишем в EEPROM
		eeprom_write_dword_safe((uint32_t*)PlusTimeMistakeEEPROM_addr, sumPlusTimeMistake);

		if (sumPlusTimeMistake > MaxPositive)
			SetFormattingMarkAndReboot(timeHackPlus);
	}
	else
	{
		uint32_t sumMinusTimeMistake =
eeprom_read_dword((uint32_t*)MinusTimeMistakeEEPROM_addr);    //  считать сумму отрицательных накруток
из EEPROM
		if (sumMinusTimeMistake == 0xFFFFFFFF) sumMinusTimeMistake = 0;

		MinusTimeMistake = unixTimeStamp - PC_time;         // посчитали текущую отрицательную
накрутку
		sumMinusTimeMistake += MinusTimeMistake;

		// запишем в EEPROM
		eeprom_write_dword_safe((uint32_t*)MinusTimeMistakeEEPROM_addr, sumMinusTimeMistake);

		if (sumMinusTimeMistake > MaxNegative)
			SetFormattingMarkAndReboot(timeHackMinus);
	}
}

/**************Format_Flash*********************************************/
// reason - причина форматирования
void SetFormattingMarkAndReboot(uint8_t reason)
{
	uint8_t formatting_mark = 0;
	while(formatting_mark == 0)    // НА потом. Добавить отдельно Форматирование после конца
загрузчика если не удалось 1000 раз поставить флаг форматирования.
	{
		eeprom_write_byte_safe((uint8_t*) formattingMarkEEPROMaddr, reason);
		formatting_mark = eeprom_read_byte ((uint8_t*) formattingMarkEEPROMaddr);
	}
	send_package_ENCRYPT(0, confirmation, NO_DATA);
	reboot();
}

/**************First Writing of Serial number into MC*************************************/
void write_serial_num(uint8_t * serialNumArr, uint8_t DataArraySize)
{
	if(DataArraySize != sizeSerNum)
	{
		send_package_ENCRYPT(0, serialNumBadSize, NO_DATA);
```

```
		return;
	}

	uint8_t sn_firstByte = eeprom_read_byte((uint8_t*) serialNumEEPROMaddr);
	if (sn_firstByte == 0xFF)
	{
		eeprom_busy_wait();
		eeprom_write_block(serialNumArr, (uint16_t*) serialNumEEPROMaddr, sizeSerNum);

		send_package_ENCRYPT(0, confirmation, NO_DATA);
	}
	else
		send_package_ENCRYPT(0, serialNumIsExists, NO_DATA);  // ошибка т.к. серийный номер можно устанавливать только 1 раз
}

void send_serial_num()
{
	uint8_t sn_firstByte = eeprom_read_byte((uint8_t*) serialNumEEPROMaddr);
	if (sn_firstByte != 0xFF)
	{
		uint8_t sn_arr_EEPROM[sizeSerNum];
		eeprom_read_block(sn_arr_EEPROM, (uint8_t*)serialNumEEPROMaddr, sizeSerNum);
		send_package_ENCRYPT(sizeSerNum, confirmation, sn_arr_EEPROM);
	}
	else
		send_package_ENCRYPT(0, serialNumIsNotExists, NO_DATA);
}

/*********USART*************************************/
void send_package_ENCRYPT(uint32_t dataSize, uint8_t response, uint8_t * data)
{
	// Ответ и размер не шифруем.
	// Зашифровать данные.

	uint8_t EncryptData[255];
	uint32_t EncryptDataSize = 0;
	for (uint16_t k = 0; k < 255; k++) EncryptData[k] = 0;   // ОБЯЗАТЕЛЬНО! Нужно обнулять этот массив т.к. из-за особонности avr Си он хранит предыдущий мусор.

	encryptArray(EncryptData, (uint32_t*)&EncryptDataSize, data, dataSize);

	// обнуление размера зашифрованной последовательности. Т.к. из-за особенности ф-ции зашифрования при нулевом массиве данных возвращается размер 1. Что является ошибкой.
	if (dataSize == 0) EncryptDataSize = 0;

	CreateAndSendPackage(response, EncryptDataSize, EncryptData);
}

void send_package_NO_ENCRYPT(uint32_t dataSize, uint8_t response, uint8_t * data)
{
	CreateAndSendPackage(response, dataSize, data);
}

void CreateAndSendPackage(uint8_t response, uint32_t dataSize, uint8_t * data)
{
	uint8_t packageSize = dataSize + 4;	// 4 - это команда и размер
	uint8_t package[dataSize + 4];
```

```
	package[0] = response;

	uint8_t sizeArr = 3;
	uint8_t dsArr[sizeArr];
	SplitArray(dataSize, dsArr, sizeArr);

	// прикрепить размер к пакету
	for(uint8_t i = 0; i < sizeArr; i++) package[i + 1] = dsArr[i];

	// прикрепить данные. Если они есть
	if (dataSize != 0)
		for (uint8_t i = 0; i < dataSize; i++) package[i + 4] = data[i];

	SendByteArray(packageSize, package);
}

void SendOneByte(uint8_t bt)
{
	uint8_t bArr[1];
	bArr[0] = bt;
	SendByteArray(1, bArr);
}

void SendByteArray(uint8_t size, uint8_t * data)
{
	Endpoint_SelectEndpoint(CDC_TX_EPNUM);

	for(uint8_t i = 0; i < size; i++) WriteNextResponseByte(data[i]);

	// Remember if the endpoint is completely full before clearing it
	bool IsEndpointFull = !(Endpoint_IsReadWriteAllowed());

	// Send the endpoint data to the host
	Endpoint_ClearIN();

	// If a full endpoint's worth of data was sent, we need to send an empty packet afterwards to signal end of
transfer
	if (IsEndpointFull)
	{
		while (!(Endpoint_IsINReady()))
		{
			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return;
		}

		Endpoint_ClearIN();
	}

	// Wait until the data has been sent to the host
	while (!(Endpoint_IsINReady()))
	{
		if (USB_DeviceState == DEVICE_STATE_Unattached)
		  return;
	}

	// Select the OUT endpoint
	Endpoint_SelectEndpoint(CDC_RX_EPNUM);
```

```
	// Acknowledge the command from the host
	Endpoint_ClearOUT();
}

static uint8_t FetchNextCommandByte(void)
{
	uint32_t timeOutCounter = 0;

	// Select the OUT endpoint so that the next data byte can be read
	Endpoint_SelectEndpoint(CDC_RX_EPNUM);

	// If OUT endpoint empty, clear it and wait for the next packet from the host
	while (!(Endpoint_IsReadWriteAllowed()))
	{
		Endpoint_ClearOUT();

		while (!(Endpoint_IsOUTReceived()))
		{
			if (timeOutCounter > 10000) return timeoutError;

			timeOutCounter++;

			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return 0;
		}
	}

	// Fetch the next byte from the OUT endpoint
	return Endpoint_Read_8();
}

static void WriteNextResponseByte(const uint8_t Response)
{
	// Select the IN endpoint so that the next data byte can be written
	Endpoint_SelectEndpoint(CDC_TX_EPNUM);

	// If IN endpoint full, clear it and wait until ready for the next packet to the host
	if (!(Endpoint_IsReadWriteAllowed()))
	{
		Endpoint_ClearIN();

		while (!(Endpoint_IsINReady()))
		{
			if (USB_DeviceState == DEVICE_STATE_Unattached)
			  return;
		}
	}

	// Write the next byte to the IN endpoint
	Endpoint_Write_8(Response);
}

/***********Time interruption*****************************************/
ISR(TIMER1_COMPA_vect, ISR_BLOCK)
{
	// Reset counter
	TCNT1H = 0;
	TCNT1L = 0;
```

```
	if (timeCounter == mc_Sec)
	{
		nextTick = true;
		timeCounter = 1;
		unixTimeStamp++;

		if(correctorValue != 0)
		{
			if (timeCorrectorCounter == correctorValue)
			{
				timeCounter++;
				timeCorrectorCounter = 1;
			}
			else
				timeCorrectorCounter++;
		}
	}

	timeCounter++;
}

/**********Encryption & Decryption data**************************/
void GenerateKey()
{
	uint8_t newTimeKey[g_TimeKeySize];
	for(uint8_t i = 0; i < g_TimeKeySize; i++) newTimeKey[i] = 0; // очистка

	for (int i = 0; i < g_TimeKeySize; i++)
	{
		uint8_t buf = rand() / (RAND_MAX / MAX_DICE + 1);
		if(buf > 50)
			buf = 1;
		else
			buf = 0;

		newTimeKey[i] = buf;
	}

	send_package_ENCRYPT(g_TimeKeySize, confirmation, newTimeKey);

	for (uint8_t i = 0; i < g_TimeKeySize; i++)	// применить нвый ключ для для МК
		g_TimeKey[i] = newTimeKey[i];
}

bool GenerateAndUseDefaultKey()
{
	// Формирование ключа по умолчанию из первых g_TimeKeySize элементов пароля
	// Формируем только если пароль установлен

	uint8_t pswdSetByte = eeprom_read_byte((uint8_t *)PasswordStartAddr);
	if(pswdSetByte != 0xff)
	{
		uint8_t password[PasswordSize];
		uint8_t newKey[g_TimeKeySize];
		eeprom_read_block(password, (uint8_t*)PasswordStartAddr, PasswordSize);

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

```
                        newKey[i] = password[i] & 1;

                for (uint8_t i = 0; i < g_TimeKeySize; i++)        // применить нвый ключ для для МК
                        g_TimeKey[i] = newKey[i];

                return true;
        }
        else
                return false;
}

void encryptArray(uint8_t* EncryptData, uint32_t* EncryptDataSize, uint8_t* inputData, uint32_t dataSize)
{
        // Не разглашается
}

void decryptArray(uint8_t* EncryptData, uint32_t EncryptDataSize, uint8_t* DecryptData, uint32_t*
DecryptDataSize)
{
   // Не разглашается
}

/**********USB_FUNCTIONS*****************/
// Event handler for the USB_ConfigurationChanged event. This configures the device's endpoints ready
// to relay data to and from the attached USB host.

void EVENT_USB_Device_ConfigurationChanged(void)
{
        // Setup CDC Notification, Rx and Tx Endpoints
        Endpoint_ConfigureEndpoint(CDC_NOTIFICATION_EPNUM, EP_TYPE_INTERRUPT,
                                ENDPOINT_DIR_IN, CDC_NOTIFICATION_EPSIZE,
                                ENDPOINT_BANK_SINGLE);

        Endpoint_ConfigureEndpoint(CDC_TX_EPNUM, EP_TYPE_BULK,
                                ENDPOINT_DIR_IN, CDC_TXRX_EPSIZE,
                                ENDPOINT_BANK_SINGLE);

        Endpoint_ConfigureEndpoint(CDC_RX_EPNUM, EP_TYPE_BULK,
                                ENDPOINT_DIR_OUT, CDC_TXRX_EPSIZE,
                                ENDPOINT_BANK_SINGLE);
}

// Event handler for the USB_ControlRequest event. This is used to catch and process control requests sent to
//  the device from the USB host before passing along unhandled control requests to the library for processing
//  internally.
//
void EVENT_USB_Device_ControlRequest(void)
{
        // Ignore any requests that aren't directed to the CDC interface
        if ((USB_ControlRequest.bmRequestType & (CONTROL_REQTYPE_TYPE |
CONTROL_REQTYPE_RECIPIENT)) !=
           (REQTYPE_CLASS | REQREC_INTERFACE))
        {
                return;
        }

        // Process CDC specific control requests
        switch (USB_ControlRequest.bRequest)
```

```
	{
		case CDC_REQ_GetLineEncoding:
			if (USB_ControlRequest.bmRequestType == (REQDIR_DEVICETOHOST |
REQTYPE_CLASS | REQREC_INTERFACE))
			{
				Endpoint_ClearSETUP();

				// Write the line coding data to the control endpoint
				Endpoint_Write_Control_Stream_LE(&LineEncoding,
sizeof(CDC_LineEncoding_t));
				Endpoint_ClearOUT();
			}

			break;
		case CDC_REQ_SetLineEncoding:
			if (USB_ControlRequest.bmRequestType == (REQDIR_HOSTTODEVICE |
REQTYPE_CLASS | REQREC_INTERFACE))
			{
				Endpoint_ClearSETUP();

				// Read the line coding data in from the host into the global struct
				Endpoint_Read_Control_Stream_LE(&LineEncoding,
sizeof(CDC_LineEncoding_t));
				Endpoint_ClearIN();
			}
			break;
	}
}
```

Приложение 2. Листинг программы тестирования на Processing(java)

diplom_interface.pde

```
import controlP5.*;
import processing.serial.*;
import g4p_controls.*;
PrintWriter output;

ControlP5 cp5;
Serial myPort;

boolean DEBUG = true;

int ProgramCounter = 0;

void draw()
{
  background(0);
}

void setup()
{
  // автоматический выбор порта с ключом
  int portNum = 0;
  boolean result = false;
  printArray(Serial.list());
  for(int i = 0; i < Serial.list().length; i++)
  {
    myPort = new Serial(this, Serial.list()[i], 9600);
    result = SendHandshake();
    if(result)
    {
      if (!PortWait()) return;
      byte incomingByte = (byte)myPort.read();
      println("TimeOutError  = " + hex(incomingByte));
      portNum = i;
      break;
    }
  }

  if(result)
    println("Key COM port = ", Serial.list()[portNum]);
  else
  {
    println("Порт с ключом не найден!!!");
    exit();
  }

  //boolean runTest = true;
  boolean runTest = false;

  size(1000, 700);


  if (runTest)
  {
    //DEBUG = false;
    keyArr_test = GenerateDefaultKey("QWEDFLKNLDDFLKGN4K3J4G93JG0J34JNG3NRSGDF");
```

```
    int pc_utc = PC_UTS();
    int mc_ust = MC_UTS();
    println("PC time: " + pc_utc + "\nMC time: " + mc_ust);

    h_test();
  }
  else
  {
    // Интерфейс
    Controls();
  }
}

GTextField tbx_cmdLine;

GTextField tbx_password;
GTextField tbx_serial_num;
GTextField tbx_end_license_time;
GTextField tbx_max_Action_Qty;
GTextField tbx_GMT_Time_Corrector;

GLabel lbl_password;
GLabel lbl_serial_num;
GLabel lbl_end_license_time;
GLabel lbl_max_Action_Qty;
GLabel lbl_GMT_Time_Corrector;

public void Controls()
{
  tbx_cmdLine = new GTextField(this, 10, 10, 180, 20);


// Параметры, необходимые для использования рабочих функций МК
  tbx_password =                new GTextField(this, 600, 200, 390, 20);
  tbx_serial_num =              new GTextField(this, 600, 225, 390, 20);
  tbx_end_license_time =        new GTextField(this, 600, 250, 390, 20);
  tbx_max_Action_Qty =          new GTextField(this, 600, 275, 390, 20);
  tbx_GMT_Time_Corrector =      new GTextField(this, 600, 300, 390, 20);

  G4P.setGlobalColorScheme(8);
  lbl_password =                new GLabel(this, 350, 35, 250, 350, "Пароль");
  lbl_serial_num =              new GLabel(this, 350, 60, 250, 350, "Серийный номер");
  lbl_end_license_time =        new GLabel(this, 350, 85, 250, 350, "Дата окончания лицензии");
  lbl_max_Action_Qty =          new GLabel(this, 350, 110, 250, 350, "Максимальное кол-во бесед в сутки");
  lbl_GMT_Time_Corrector =      new GLabel(this, 350, 135, 250, 350, "Корректор времени для GMT");


  lbl_password.setTextAlign(GAlign.RIGHT, null);
  lbl_serial_num.setTextAlign(GAlign.RIGHT, null);
  lbl_end_license_time.setTextAlign(GAlign.RIGHT, null);
  lbl_max_Action_Qty.setTextAlign(GAlign.RIGHT, null);
  lbl_GMT_Time_Corrector.setTextAlign(GAlign.RIGHT, null);

  tbx_password.setText("qwedflknlddflkgn4k3j4g93jg0j34jng3nrsgdf".toUpperCase());
  tbx_serial_num.setText("0705160102");
  tbx_end_license_time.setText(Integer.toString(PC_UTS() + 2678400 * 1));  // кол-во месяцев 2678400 * 1
  tbx_max_Action_Qty.setText("40");
  tbx_GMT_Time_Corrector.setText("10800");
```

```
  cp5 = new ControlP5(this);
  PFont font = createFont("arial", 50);

// Инициализировать
cp5.addBang("btn_init")                    .setPosition(850, 325).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Поздороваться
  cp5.addBang("btn_handshake_test_z")          .setPosition(10, 50).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// тест
  cp5.addBang("btn_Send_Command_Line")         .setPosition(200, 10).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Работа с паролем
  cp5.addBang("btn_Send_true_Password_A")      .setPosition(10, 100).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_Send_bad_Password_A")       .setPosition(10, 125).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_set_password_P")            .setPosition(10, 150).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_use_default_key_Y")         .setPosition(10, 200).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


// Работа со временем
  cp5.addBang("btn_Timing_i")                  .setPosition(200, 100).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_time_test_t")               .setPosition(200, 125).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_get_mc_time")               .setPosition(200, 150).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_GetSynchroTimeFromEEPROM_i")  .setPosition(200, 175).setSize(170,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_GetPlusTimeMistake_F")      .setPosition(200, 225).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_SendMinusTimeMistake_m")    .setPosition(200, 275).setSize(170,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_SendMaxPositive_d")         .setPosition(200, 325).setSize(170,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("byte_SendMaxNegative_k")        .setPosition(200, 350).setSize(170,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Работа с флэш
  cp5.addBang("btn_FLASH_FORMAT_F")            .setPosition(400, 100).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_get_formatting_reason_r")   .setPosition(400, 125).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Серийный номер
  cp5.addBang("btn_set_serialNum_w")           .setPosition(600, 100).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_get_serialNum_s")           .setPosition(600, 125).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Время лицензии
```

```
  cp5.addBang("btn_set_endLicenseTime_T")         .setPosition(800, 100).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_get_endLicenseTime_e")         .setPosition(800, 125).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


// Action. Обследование/беседа с проверяемым
  cp5.addBang("btn_Action_u")                     .setPosition(10, 300).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_SetActionMaxQuontity_h")       .setPosition(10, 325).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_GETActionMaxQuontity_g")       .setPosition(10, 350).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_SetActionTimeCorrector_q")     .setPosition(10, 375).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_GetActionTimeCorrector_j")     .setPosition(10, 400).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_RemainCurrentQtyAction_r")     .setPosition(10, 425).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


// Шифрование
  cp5.addBang("btn_GenerateKey_4")                .setPosition(400, 425).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_encryption_test")              .setPosition(400, 475).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


// База данных
  cp5.addBang("btn_DB_Read_All_5")                .setPosition(600, 350).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_DB_SetOneVal_6")               .setPosition(600, 375).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
  cp5.addBang("btn_DB_GetOneVal_7")               .setPosition(600, 400).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

// Посчитать по формуле
  cp5.addBang("btn_compute_f_wzh")                .setPosition(850, 450).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


  cp5.addBang("btn_heavy_test")                   .setPosition(850, 600).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

  cp5.addBang("btn_test")                         .setPosition(200, 50).setSize(140,
20).getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

  textFont(font);
  text("word", 10, 50);
}

public void handleTextEvents(GEditableTextControl textControl, GEvent event){}
```

DataTransfer.pde

```
public class packageMC //<>//
{
  public char Size;
  public char Response;
  public char[] Data;
}

// Команда, данные
public packageMC send_package(byte command, char data[], String info)
{
  d_println("");
  d_println("");
  d_println(info);
  String currentDateTime = day() + "." + month() + "." + year() + " " + hour() + ":" + minute() + ":" + second();
  d_println("ProgramCounter: " + ProgramCounter + " _____ " + currentDateTime + " _____unix TimeStamp: " +
System.currentTimeMillis()/1000);
  ProgramCounter++;
  // Шифруемый пакет
  // не шифруется,   не шифруется     шифруется
  // команда(1байт), размер(3байта),  шифрованная последовательность по размеру
  // Размер данных
  byte dataSize = 0;
  if (data != null) dataSize = (byte)data.length;

  d_println("     outCmd: " + (int)command);
  d_println("outDataSize: " + (int)dataSize);
  if (data != null)
  {
    d_print("    outData: ");
    printCharArray(data);
  }
  // зашифрование
  char encryptDataPackageSize = 0;
  char[] encryptDataPackage = new char[0];
  if (data != null && dataSize != 0)  // шифруем данные если они есть
  {
    d_print("           CurrentKey: ");
    printCharArray(keyArr_test);
    encryptionData encryptedData = encrypt(data, keyArr_test);
    d_println("    encryptedData.Size: " + (int)encryptedData.Size);
    d_print("    encryptedData.Data: ");
    printCharArray(encryptedData.Data);

    encryptDataPackageSize = (char)(encryptedData.Size);
    encryptDataPackage = new char[encryptDataPackageSize];

    d_println("encryptDataPackageSize: " + (int)encryptDataPackageSize);

    for (int i = 0; i < encryptedData.Size; i++)
      encryptDataPackage[i] = encryptedData.Data[i];

    d_print("    encryptDataPackage: ");
    printCharArray(encryptDataPackage);
  }

  int PacketSize = encryptDataPackageSize + 4; // Размер всего пакета. 1байт команда, 1й, 2й, 3й размер, данные
по размеру
```

```
  char[] packageOut = new char[PacketSize];
  packageOut[0] = (char)command;

  // дописать размер
  char[] dsArray = new char[3];
  dsArray = SplitArray(encryptDataPackageSize, 3);
  for (int i = 0; i < 3; i++)
    packageOut[i+1] = dsArray[i];

  // дописать зашифрованные данные
  for (int i = 0; i < encryptDataPackageSize; i++)
    packageOut[i + 4] = encryptDataPackage[i];


  d_print("packageOut: ");
  printCharArray(packageOut);

  if (SendHandshake()) d_println("ok handshake");
  else {
    d_println("bad handshake");
    return null;
  }

  d_println("_____________________________________________________________________________");

  myPort.write(byte(packageOut));  // отправка

  // ждем ответа.    // Если вышли по таймауту
  if (!PortWait()) {
    d_println("Команда '" + command +"'. Ответ не получен по истечению таймаута");
    return null;
  }
  packageMC pack  = recieve();
  return pack;
}

public packageMC recieve()
{
  // 0й байт овтет
  byte response = -1;
  if (myPort.available() > 0) response = (byte)(myPort.read());

  // 1й, 2й, 3й - размер
  char[] dsArray = new char[3];
  for (char i = 0; i < 3; i++)
    dsArray[i] = (char)myPort.read();

  d_print("inDataSize: ");
    printCharArray(dsArray);

  int inEncryptedDataSize = JoinArray(dsArray, (byte)3);

  // данные по размеру
  char[] inDataEncrypted = new char[inEncryptedDataSize];
  for (int i = 0; i < inEncryptedDataSize; i++) inDataEncrypted[i] = (char)myPort.read();

  if (inEncryptedDataSize != 0)
  {
```

```
    d_println("inEncryptedDataSize: " + (int)inEncryptedDataSize);
    d_print("    inDataEncrypted: ");
    printCharArray(inDataEncrypted);

    d_print("        inByteArray: ");
    for (int i =0; i< inDataEncrypted.length; i++) d_print(String.format("%8s",
Integer.toBinaryString(inDataEncrypted[i])).replace(' ', '0') + " ");
    d_println("");
   }

   encryptionData mcEnc__Data = new encryptionData();

   mcEnc__Data.Data = inDataEncrypted;
   mcEnc__Data.Size = (char)inDataEncrypted.length;

   encryptionData decrData = decrypt(mcEnc__Data, keyArr_test);

   if (decrData.Size != 0)
   {
    d_print("    inDataDecrypted: ");
    printCharArray(decrData.Data);
   }

   d_println("  response: " + (int)response);
   d_println("inDataSize: " + (int)decrData.Size);
   d_print("    inData: ");
   if (decrData.Size != 0)
    printCharArray(decrData.Data);
   else
    d_print("NO_DATA");

   d_println("");
   d_println("");

   packageMC pack = new  packageMC();
   pack.Response = (char)response;
   pack.Size = decrData.Size;
   pack.Data = decrData.Data;

   return pack;
}

// Ожидание ответа от МК. Если не дождались ответа false
public boolean PortWait()
{
  // ждем ответа
  int waitCounter = 0;
  boolean wait = true;
  while (myPort.available() == 0 && wait)
  {
    delay(10);
    waitCounter++;
    if (waitCounter > 100) wait = false;
  }
  return wait;
}
```

btn_release.pde

```
////////////////////////////
////////// Общие функции ////
////////////////////////////
// ascii коды >= 91

byte handshake =                    'Z';      // поздороваться

// Action. Обследование/беседа с проверяемым
byte action =                       'U';  // провести беседу
byte SetActionMaxQuontity =         'H';  // Установить максимально количество бесед в сутки
byte GetActionMaxQuontity =         'G';  // Получить максимально количество бесед в сутки
byte SetActionTimeCorrector =       'Q';  // Установить корректор времени для часового пояса
byte GetActionTimeCorrector =       'J';  // Получить корректор времени для часового пояса
byte GetRemainCurrentQtyAction =    'R';  // Получить текущее оставшееся количество бесед

// Время лицензии
byte set_endLicenseTime =           'T';  // Установить время окончания лицензии в EEPROM
byte get_endLicenseTime =           'E';  // Получить время окончания лицензии из EEPROM

// Серийный номер
byte writeSerNum =                  'W';  // Записать серийный номер в EEPROM
byte getSerNum =                    'S';  // Получить серийный номер из EEPROM

// Пароль
byte set_pasword =                  'P';  // Установить пароль
byte send_password =                'A';  // Авторизация
byte use_default_key =              'Y';  // Использовать ключ по умолчанию.

// Время
byte timing =                       '+';  // синхронизировать время МК с ПК
byte get_time =                     '?';  // получить время МК
byte GetPlusTimeMistake =           'F';  // получить положительную накрутку
byte SendMinusTimeMistake =         'M';  // Отправить на ПК сумму отрицательных накруток в секундах
byte GetSynchroTimeFromEEPROM =     'I';  // получить время синхронизации из EEPROM
byte SendMaxPositive =              'D';  // получить максимально возможную положительную накрутку
byte SendMaxNegative =              'K';  // получить максимально возможную отрицательную накрутку

// Шифрование
byte GenerateKey =                  '4';  // получить сгенерированный рандомный ключ

// Флэш
byte FORMAT_FLASH =                 'N';  // Отформатировать flash

// База данных
byte DB_Read_All =                  '5';  // Получить всю БД
byte DB_SetOneVal =                 '6';  // Записать 1 байт по ключу
byte DB_GetOneVal =                 '7';  // Прочитать 1 байт по ключу

// Посчитать по формуле
byte compute_f_wzh =                'V';  //

////////////////////////////////////
//////// Ответы от МК//////////////
////////////////////////////////////
// uint8_t confirmation = 0x00;    // всё успешно
// uint8_t deny = 0x01;            // отказ при выполнении административной команды из-за того что пользователь
не авторизован командой 'A'
```

```
// uint8_t unknownCommand = 0x02;  // неизвестная команда

//_____________________________________  Кнопки
public void btn_handshake_test_z()
{
  boolean result = SendHandshake();
  if (result)
    println("ok");
  else
    println("bad");

  if (!PortWait()) return;

  byte incomingByte = (byte)myPort.read();
  println("TimeOutError  = " + hex(incomingByte));
}


////////////// работа с паролем
public void btn_set_password_P()
{
  String pswd = tbx_password.getText();
  char[] pswdArr = new char[pswd.length()];

  for (int i = 0; i < pswd.length(); i++)
  {
    String oneChar = pswd.substring(i, i+1);
    pswdArr[i] = (char)oneChar.charAt(0);
  }

  send_package(set_pasword, pswdArr, "Установка пароля");

  keyArr_test = GenerateDefaultKey(tbx_password.getText());
}

// Проверить пароль в МК
public void btn_Send_true_Password_A()
{
  String pswd = tbx_password.getText();
  char[] pswdArr = new char[pswd.length()];

  for (int i = 0; i < pswd.length(); i++)
  {
    String oneChar = pswd.substring(i, i+1);
    pswdArr[i] = (char)oneChar.charAt(0);
  }

  printCharArray(send_package(send_password, pswdArr, "Авторизация").Data);
}



public void btn_Send_bad_Password_A()
{
  char AdminPassword[] = {1, 71, 12, 58, 33, 11, 22, 12, 25, 8,
    23, 98, 11, 41, 12, 2, 33, 85, 12, 12,
    12, 15, 1, 23, 17, 14, 78, 2, 78, 12,
    11, 33, 34, 12, 28, 23, 42, 42, 21, 2};
```

```
  printCharArray(send_package(send_password, AdminPassword, "Авторизация").Data);
}

public void btn_use_default_key_Y()
{
  send_package(use_default_key, null, "Сброс секретного ключа на дефолтный");
  keyArr_test = GenerateDefaultKey(tbx_password.getText());

  println("Используем дефолтный ключ");
}

// Работа с флеш

public void btn_FLASH_FORMAT_F()
{
  printCharArray(send_package(FORMAT_FLASH, null, "Форматирование").Data);
}

////////////////////// Время ////////////////////////
public void btn_Timing_i()
{
  int pc_uts = (int)(System.currentTimeMillis()/1000);
  char[] USTarr = SplitArray(pc_uts, 5);
  send_package(timing, USTarr, "Синхронизация времени");
}

public void btn_get_mc_time()
{
  int pc_utc = PC_UTS();
  int mc_ust = MC_UTS();
  println("PC time: " + pc_utc + "\nMC time: " + mc_ust);
}

public void btn_GetSynchroTimeFromEEPROM_i()
{
  char[] timeFromEEPROM = send_package(GetSynchroTimeFromEEPROM, null, "Получить время
синхронизации из EEPROM").Data;
  /*if (timeFromEEPROM != null)
    printCharArray(timeFromEEPROM);*/
}

public void btn_SendMinusTimeMistake_m()
{
  char[] MinusTimeMistakeArr = send_package(SendMinusTimeMistake, null, "Отрицательные накрутки
времени").Data;

  int mc_time = JoinArray(MinusTimeMistakeArr, (byte)MinusTimeMistakeArr.length);
  println(mc_time);
}

public void btn_GetPlusTimeMistake_F()
{
  char[] PlusTimeMistakeArr =  send_package(GetPlusTimeMistake, null, "Положительные накрутки
времени").Data;
  int mc_time = JoinArray(PlusTimeMistakeArr, (byte)PlusTimeMistakeArr.length);
  println(mc_time);
}
```

```
public void btn_SendMaxPositive_d()
{
  packageMC maxPositivePack = send_package(SendMaxPositive, null, "Получить максимально возможную положительную накрутку");
  if (maxPositivePack.Data != null)
  {
    int maxPositive = JoinArray(maxPositivePack.Data, (byte)maxPositivePack.Data.length);
    println(maxPositive);
  }
}

public void byte_SendMaxNegative_k()
{
  packageMC maxNegativePack = send_package(SendMaxNegative, null, "Получить максимально возможную отрицательную накрутку");
  if (maxNegativePack.Data != null)
  {
    int maxNegative = JoinArray(maxNegativePack.Data, (byte)maxNegativePack.Data.length);
    println(maxNegative);
  }
}

////////////////////// Шифрование /////////////////
public void btn_GenerateKey_4()
{
  char[] keyArr = send_package(GenerateKey, null, "Получить ключ шифрования").Data;
  if(keyArr.length != 0)
  {
    printCharArray(keyArr);
    for (int i = 0; i < keyArr_test.length; i++)
     keyArr_test[i] = keyArr[i];
  }
}

public void btn_get_formatting_reason_r()
{
  byte get_formatting_reason = 'r';
  send_package(get_formatting_reason, null, "Получить причину форматирования");
}

public void btn_time_test_t()
{
  output = createWriter("log" + PC_UTS() + ".txt");   // файл
  boolean isSync = false;
  int counter = 0;

  output.println("start: " + PC_UTS());
  output.flush();

  // sync while PCtime != MCtime
  while(!isSync)
  {
    println("     sync " + counter);
    btn_Timing_i();

    println("PC time" + PC_UTS());
    println("MC time" + MC_UTS());
```

```
    // check MCtime == PCtime
    // if MCtime != PCtime then sync
    if (MC_UTS() != PC_UTS())
      isSync = false;
    else
      isSync = true;

    delay(1000);
    counter++;
  }

  println("              syncEND ");

  while(true)
  {
    // if MCtime < PCtime then mc_Sec++;
    // if MCtime > PCtime then mc_Sec--;
    int pc_utc = PC_UTS();
    int mc_ust = MC_UTS();
    println("PC time_____________________: " + pc_utc + "\nMC time_____________________: " + mc_ust);
    println();

    output.println("pc_uts: " + pc_utc + " mc_uts: " + mc_ust + " result (pc_uts - mc_time): " + (pc_utc - mc_ust) );
    output.flush();

    delay(10000);
  }
}

// Отправить группу команд без даннх
public void btn_Send_Command_Line()
{
  String command;
  command = tbx_cmdLine.getText();
  for (int i = 0; i < command.length(); i++)
  {
    String oneCommand = command.substring(i, i+1);
    char[] res = send_package((byte)oneCommand.charAt(0), null, "").Data;

    for(int j = 0; j < res.length; j++)
      print(res[j]);

  }
}

// Серийный номер
  public void btn_set_serialNum_w()
  {
    char serNumArr[] = tbx_serial_num.getText().toCharArray();
    if (serNumArr.length == 0)
    {
      println("Серийный номер не заполнен.");
      return;
    }

    send_package(writeSerNum, serNumArr, "Установка серийного номера");
  }
```

```
  public void btn_get_serialNum_s()
  {
     char[] sNum = send_package(getSerNum, null, "Получить серийный номер").Data;

     if (sNum != null && sNum.length > 1)
     {
      for(int i = 0; i < 10; i++)
      {
       print(sNum[i]);
      }
     }
  }

// Время лицензии
  public void btn_set_endLicenseTime_T()
  {
    int endLicenseTime = Integer.parseInt(tbx_end_license_time.getText());
    char[] licTimeArr = SplitArray_new(endLicenseTime);
    packageMC licTimePack = send_package(set_endLicenseTime, licTimeArr, "Установка даты окончания
лицензии " + endLicenseTime);
    if (licTimePack.Response != 0) println("Время окончания лицензии не установлено!!!!!!!!");
  }

  public void btn_get_endLicenseTime_e()
  {
     char[] licTimeArr = send_package(get_endLicenseTime, null, "Получить время окончания лицензии").Data;

     if (licTimeArr != null && licTimeArr.length > 1)
     {
      int res = JoinArray(licTimeArr,  (byte)licTimeArr.length);
      println(res);
     }
  }

public void btn_Action_u()
{
  char[] resultArr = send_package(action, null, "Действие").Data;
  if (resultArr != null)
    printCharArray(resultArr);

}

public void btn_SetActionMaxQuontity_h()
{
  char[] MaxQuontity = new char[1];
  MaxQuontity[0] = (char)40;
  //printCharArray(MaxQuontity);
  send_package(SetActionMaxQuontity, MaxQuontity, "Установить максимальное число действий в сутки");
}

public void btn_GETActionMaxQuontity_g()
{
  printCharArray(send_package(GetActionMaxQuontity, null, "Получить максимальное число действий в
сутки").Data);
}

public void btn_SetActionTimeCorrector_q()
{
```

```
  int timeCorrector = 14400;  // 14400 +4   часа. GMT+3 для Москвы
  char[] timeCorrectorArr = SplitArray_new(timeCorrector);
  send_package(SetActionTimeCorrector, timeCorrectorArr, "Установить корректор времени");
}

public void btn_GetActionTimeCorrector_j()
{
  char[] TimeCorrectorArr = send_package(GetActionTimeCorrector, null, "Получить корректор времени").Data;
  int TimeCorrector = JoinArray(TimeCorrectorArr, (byte)TimeCorrectorArr.length);
  println(TimeCorrector);
}

public void btn_RemainCurrentQtyAction_r()
{
  printCharArray(send_package(GetRemainCurrentQtyAction, null, "Получить оставшееся количество действий на
текущие сутки").Data);
}

// База данных
public void btn_DB_Read_All_5()
{
  char[] dataBase = send_package(DB_Read_All, null, "Считать всю БД").Data;
  if (dataBase != null)
    printCharArray(dataBase);
}

public void btn_DB_SetOneVal_6()
{
  char[] outKeyVal = {1, 241};
  send_package(DB_SetOneVal, outKeyVal, "Записать значение в БД");
}

public void btn_DB_GetOneVal_7()
{
  char[] outKeyVal = {1};
  char[] inKeyVal = send_package(DB_GetOneVal, outKeyVal, "Считать значение из БД").Data;
  if (inKeyVal != null)
    printCharArray(inKeyVal);
}

////////////// Посчитать по формуле  ////////////////////////////

public void btn_compute_f_wzh()
{ // Не разлашается }

char[] prepareArray(char[][] outArr,  char fCount)
{ // Не разглашается }

////////////////////////////////////////////////////////////

// Тест
public void btn_test()
{
  String sn_str = tbx_serial_num.getText();
  println(sn_str);
  char sn_char_array[] = sn_str.toCharArray();
  printCharArray(sn_char_array);
}
```

```
char[] split_int(int n)
{
   char[] c = new char[10];
   int v = 0; //количество цифр в числе n
   //разбиваем на отдельные символы число n
   while (n > 9)
   {
      c[v++] = (char) ((n % 10) + '0');
      n = n / 10;
   }
   c[v++] = (char) (n + '0');
   c[v] = '\0';
   char t;
   //инвертируем массив символов
   for (int i = 0; i < v / 2; i++)
   {
      t = c[i];
      c[i] = c[v - 1 - i];
      c[v - 1 - i] = t;
   }
   v = 0;

 return c;
}
```

encryption.pde
Не разглашается

functions.pde

```
//_____________________________________  Функции

char[] SplitArray(int timeStamp, int size)
{
  int divisor = 1;
  int subtrahend;
  int tmpTime;
  int time = timeStamp;
  char mass[] = new char[size];

  for (int i = 1; i < size; i++)
    divisor *= 100;

  /*println();
   println("divisor " + divisor);
   println("size " + size);*/

  for (int i = 0; i < size; i++)
  {
    if (i != 0)
    {
      subtrahend = mass[i - 1] * divisor;
      time -= subtrahend;
      divisor /= 100;
    }

    tmpTime = time / divisor;
    mass[i] = (char)tmpTime;
  }

  return mass;
}

char[] SplitArray_new(int timeStamp)
{
  int divisor = 1;
  int multiplier = 1;
  int prevTime = 0;
  int item;
  int time = timeStamp;

  divisor = 100;
  multiplier = 100;

  int size = 0;
  int tmpTime = timeStamp;
  while (tmpTime != 0)
  {
    tmpTime /= divisor;
    size++;
  }

  char[] mass = new char[size];

  for (int i = size - 1; i >= 0; i--)
  {
    if (i == size - 1)
```

```
      prevTime = timeStamp;
    else
      prevTime = prevTime / divisor;


    time = time / divisor;
    item = prevTime - time * multiplier;
    mass[i] = (char)item;
  }

  return mass;
}

int JoinArray(char[] mass, byte size)
{
  if (mass != null && mass.length != 0 )
  {
    int joinTS = 0;
    int multiplier = 1;
    int i;

    for (int x = 1; x < size; x++)
      multiplier *= 100;

    for (i = 0; i < size; i++)
    {
      if (i != 0) multiplier /= 100;
      joinTS += mass[i] * multiplier;
    }

    return joinTS;
  }
  return 0;
}

public void printCharArray(char[] array)
{
  if (array != null)
  {
    for (int i = 0; i < array.length; i++)
    {
      d_print((int)array[i] + " ");
    }
    d_println("");
  }
}

public void printHexArray(byte[] array)
{
  if (array != null)
  {
    for (int i = 0; i < array.length; i++)
    {
      d_print(hex(array[i]) + " ");
    }
  }
}
```

```
public void printCharArrayReverse(byte[] array)
{
  if (array != null)
  {
    for (int i = array.length - 1; i >= 0; i--)
    {
      d_print(array[i] + " ");
    }
  }
}

void printCharArrayInBit(char[] arr)
{
  for(int i = 0; i < arr.length; i++)
  {
    d_print(String.format("%8s", Integer.toBinaryString(arr[i])).replace(' ', '0') + " ");
  }
}

public int PC_UTS()
{
  return (int)(System.currentTimeMillis()/1000);
}

public int MC_UTS()
{
  char[] syncTimeArr = send_package(get_time, null, "Получить текущее время МК").Data;
  int mc_time = JoinArray(syncTimeArr, (byte)syncTimeArr.length);
  return mc_time;
}

// Поздороваться с МК. Вернуть true если МК поздоровался в ответ
public boolean SendHandshake()
{
  // отправить handshake
  myPort.write(handshake);

  if (!PortWait()) // ждем ответа.
    return false;  // если не дождались
  byte incomingHandShake = (byte)myPort.read();

  if (incomingHandShake == handshake)
    return true;
  else
    return false;
}

void d_print(String str)
{
  if (DEBUG)
    print(str);
}

void d_println(String str)
{
  if (DEBUG)
    println(str);
}
```