

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность систем
и технологий»

Специальность 10.05.03 –

Информационная безопасность
автоматизированных систем

Специализация -

Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ
IP-ТРАФИКА

Студент

Исаков Дмитрий Сергеевич

Руководитель

Агафонов П.В.

Нормоконтролёр

Фатеев А.Г.

Рецензент

Редьяев А.В.

Работа допущена к защите (протокол заседания кафедры от 26.06.18 № 16)

Заведующий кафедрой

С.Л. Зефиоров

Работа защищена с отметкой хорошо (протокол заседания ГЭК
от 27.06.18 № 4)

Секретарь ГЭК

А.П. Иванов

Подпись П.В. Агафонов
инженер 1 кат. ВКР



Проверка ВКР - Юбатов
25.06.18

Оглавление

Введение.....	5
1 Описание аппаратной платформы устройства криптографической защиты IP-трафика.....	6
2 Описание архитектуры микроконтроллера TI MSP, асинхронного интерфейса приемопередачи данных - UART.....	10
2.1 Описание архитектуры микроконтроллера TI MSP.....	10
2.2 Описание асинхронного интерфейса приемопередачи данных - UART.....	23
3 Описание программных модулей устройства криптографической защиты IP-трафика.....	27
4 Разработка программного модуля инициализации микроконтроллера.....	34
5 Разработка программных модулей приема и передачи данных....	36
6 Разработка программного криптографического модуля.....	39
7 Отладка и тестирование разработанного программного обеспечения на макетной плате ЕК-ТМ4С1294ХL.....	41
8 Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....	55
Заключение.....	59
Список используемых источников.....	60
Приложение А Архитектура МК ТМ4С1294NCPDT.....	61
Приложение Б Блок-схема алгоритма функционирования модуля инициализации микроконтроллера.....	62
Приложение В Блок-схема алгоритма функционирования модуля приема данных.....	63
Приложение Г Блок-схема алгоритма функционирования модуля передачи данных.....	64

Обозначения и сокращения

В настоящем отчете применяют следующие обозначения и сокращения:

АЦП	—	Аналого-цифровое преобразование;
ГОСТ	—	Государственный стандарт;
МК	—	Микроконтроллер;
УКЗ	—	Устройство криптографической защиты;
ПЭВМ	—	Персональная электронно-вычислительная машина;
ШИМ	—	Широтно-импульсная модуляция.

Введение

Информация — это одна из самых ценных вещей в современной жизни. Появление глобальных компьютерных сетей сделало простым получение доступа к информации. Легкость и скорость доступа к данным с помощью компьютерных сетей, таких как Интернет, сделали значительными такие угрозы безопасности данных как неавторизованный доступ к информации, неавторизованное изменение информации, неавторизованный доступ к сетям и другим сервисам.

На сегодняшний день для защиты информации от несанкционированного доступа применяются программные, аппаратные и программно-аппаратные средства.

Программная реализация системы защиты информации имеет низкую стоимость, но качество защиты данных невысокое. Аппаратная реализация обеспечивает более высокое качество защиты информации, но цена таких технических решений возрастает. Программно-аппаратные системы для защиты информации предоставляют пользователю гибкость настройки и высокую защищенность данных. Программно-аппаратная криптосистема состоит из электронного устройства, которое подключается к персональному компьютеру и программного обеспечения для работы с устройством. В таких системах выполнение функций, некритичных к скорости работы и безопасности, перекладывается на программное обеспечение, что способствует снижению их стоимости.

Поэтому дипломный проект связан с наиболее актуальным и перспективным способом защиты информации от несанкционированного доступа — программно-аппаратные системы. В данном проекте будет разработано программное обеспечение для защиты IP-трафика на основе макетной платы EK-TM4C1294XL, которая выполнена на базе специализированного микроконтроллера TM4C1294NCPDT.

1 Описание аппаратной платформы устройства криптографической защиты IP-трафика

В качестве аппаратной платформы устройства криптографической защиты IP-трафика выступает плата ЕК-TM4C1294XL — макетная плата на базе мощного ядра ARM® Cortex-M4 микроконтроллера TM4C1294NCPDT.

На рисунке 1 приведена фотография платы ЕК-TM4C1294XL с указанием основных ее функций.

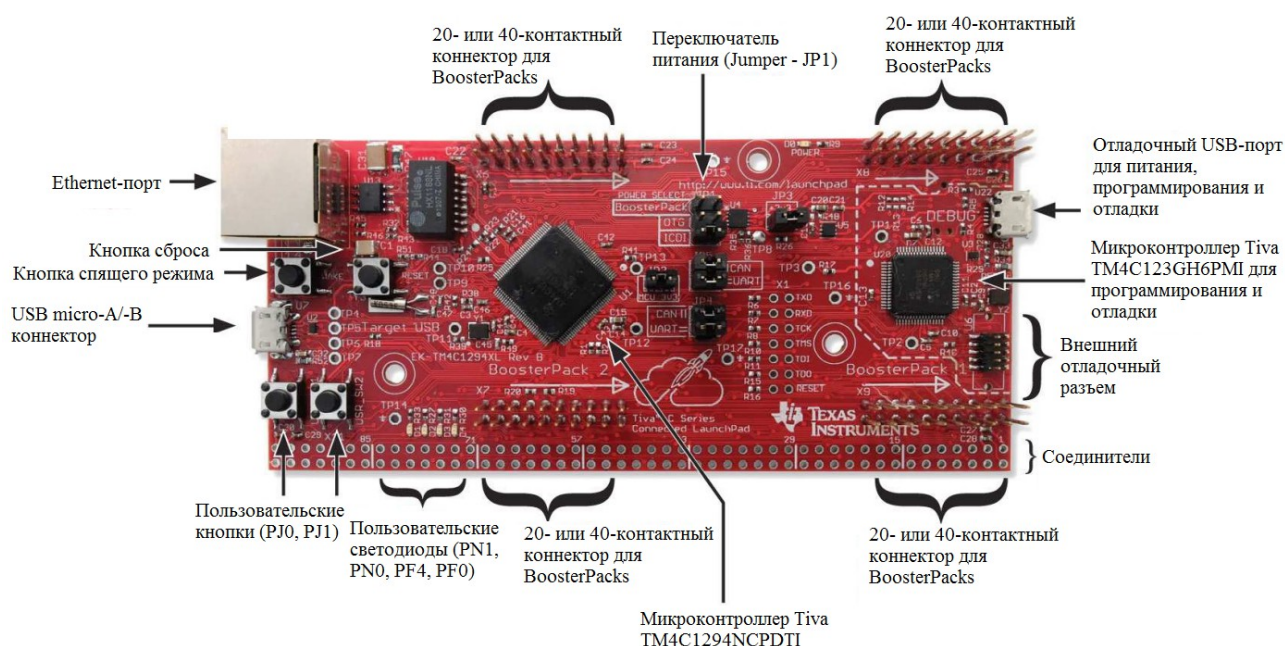


Рисунок 1 — Фотография платы с указанием основных ее функций

К основным преимуществам данной платы можно отнести следующие:

- производительный микроконтроллер TM4C1294NCPDT с интегрированным MAC и PHY и встроенным широким набором периферии;
- недорогая цена;
- наличие пользовательских кнопок и программируемых светодиодов;
- встроенный интерфейс для программирования и отладки ICDI;
- большое коммутационное поле для подключения различных устройств (98 входов/выходов общего назначения, 2 40-контактных коннектора для подключения BoosterPack);

— наличие переключки питания, позволяющей регулировать питание платы: от источника питания через USB-разъем, через отладочный USB-разъем, от BoosterPack.

На рисунке 2 приведена архитектура аппаратной платформы устройства криптографической защиты IP-трафика.

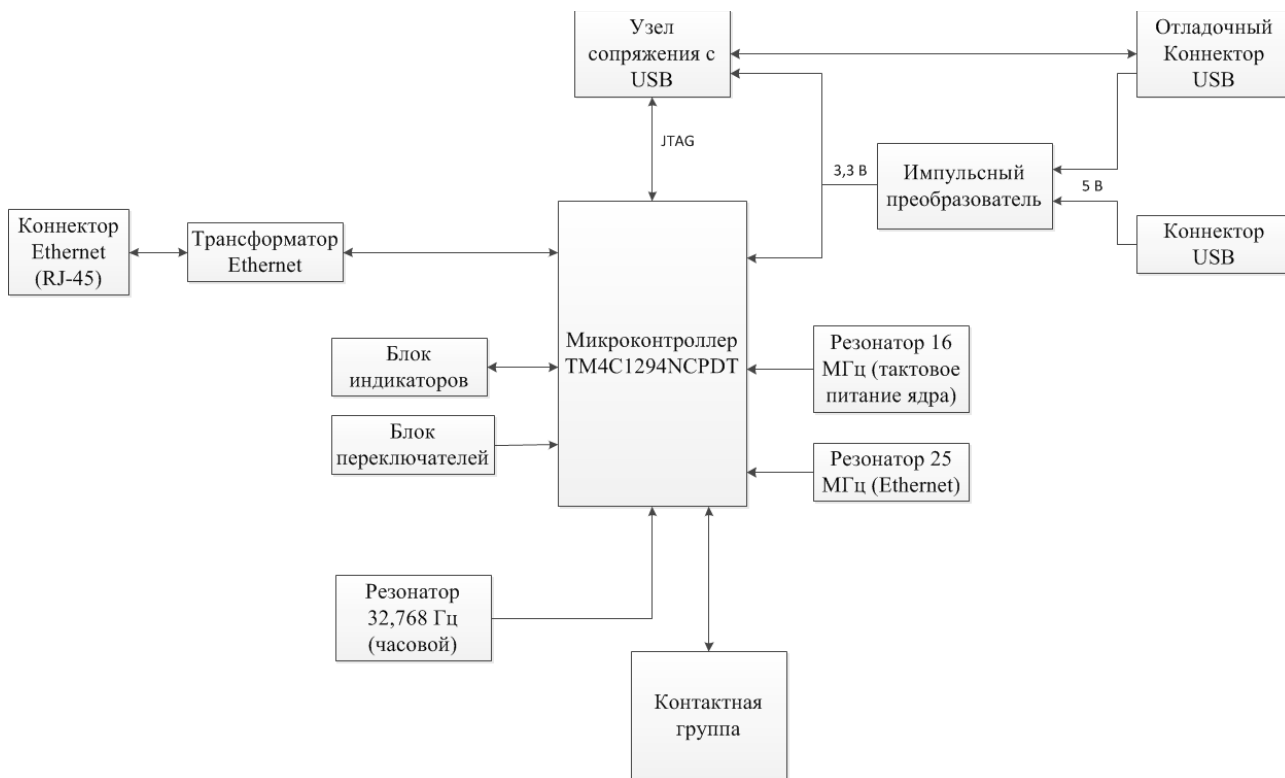


Рисунок 2 — Архитектура аппаратной платформы устройства криптографической защиты IP-трафика

Основа платы это микроконтроллер TM4C1294NCPDTI — 32-разрядный микроконтроллер ARM Cortex-M4F с флэш-памятью 1024 КБ, 256 КБ SRAM памяти, 6 КБ EEPROM и с частотой тактирования 120 МГц. Микроконтроллер имеет интегрированный 10/100 Ethernet MAC и PHY, соединение USB 2.0, модуль спящего режима, множество последовательных соединений (Ethernet, UART, USB и другие), а также широкий спектр других периферийных устройств.

Микроконтроллер имеет 3 резонатора. Резонатор с частотой 16 МГц предназначен для тактового питания ядра микроконтроллера. Резонатор с

частотой 25 МГц предназначен для тактирования MAC и PHY Ethernet-блоков микроконтроллера. Резонатор 32,768 Гц предназначен для работы микроконтроллера в режиме пониженного энергопотребления (в спящем режиме), а также для реализации функций вычисления времени.

Контактная группа необходима для подключения устройств к плате с помощью программируемых контактов общего назначения, что обеспечивает гибкость для различных соединений. Контактная группа поддерживает от 0 до 98 программируемых входных/выходных контактов. Количество доступных контактов зависит от используемых периферийных устройств.

Предусмотрены два пользовательских переключателя для ввода и управления программным обеспечением TM4C1294NCPDTI. Переключатели подключены к контактам GPIO PJ0 и PJ1. Также предусмотрен переключатель сброса и переключатель спящего режима. Переключатель сброса предназначен для сброс системы микроконтроллера. Переключатель спящего режима - это один из способов вывести устройство из спящего режима.

На плате предусмотрено четыре пользовательских светодиода. Светодиоды D1 и D2 подключены к контактам GPIO PN1 и PN0. Эти светодиоды предназначены для использования приложениями. Светодиоды D3 и D4 подключены к контактам GPIO PF4 и PF0, которые могут управляться программным обеспечением пользователя или встроенным модулем Ethernet микроконтроллера. Также предусмотрен светодиод питания, чтобы указать, что на плате присутствует напряжение 3,3 В.

Плата способна подключаться к сети Ethernet с использованием разъема типа RJ45. Микроконтроллер содержит полностью интегрированный Ethernet MAC и PHY.

Трансформатор Ethernet предназначен для согласования PHY микроконтроллера с линией стандарта 10/100 base-TX (Ethernet).

Плата питается от источника питания, подключенный через разъем USB (RJ-45). Импульсный преобразователь необходим для преобразования входного

сигнала с напряжением 5 В в выходное напряжение 3,3 В для питания элементов платы EK-TM4C1294XL.

Узел сопряжения с USB предназначен для подключения микроконтроллера к ПЭВМ через отладочный USB-порт в режиме отладки и программирования, а также в обычном рабочем режиме через USB-порт. Узел сопряжения с USB реализован на базе микроконтроллера TM4C123GH6PMI.

2 Описание архитектуры микроконтроллера TI MSP, асинхронного интерфейса приемопередачи данных — UART

2.1 Описание архитектуры микроконтроллера TI MSP

Микросхема TM4C1294NCPDT относится к новому семейству микроконтроллеров Tiva™ C ARM Cortex-M4 фирмы TI. Серия характеризуется максимальной производительностью и широким набором периферийных узлов при низкой стоимости. Микроконтроллеры Tiva™ позиционируются в качестве экономичного приложения при решении задач обработки данных, управления и сопряжения по различным типам интерфейсов для применения в таких продуктах как:

- промышленное коммуникационное оборудование;
- сетевые устройства, шлюзы и адаптеры;
- оборудование мониторинга и контроля жилых и коммерческих объектов;
- системы безопасности и доступа;
- панели управления HMI;
- технические средства управления автоматизацией производства;
- испытательное и измерительное оборудование;
- системы пожарной и охранной сигнализации;
- медицинское оборудование;
- игровое оборудование;
- электронные индикаторы продажи (POS);
- системы интеллектуального управления освещением;
- оборудование отслеживания транспортных средств.

Микроконтроллеры Серии Tiva™ объединяют в себе множество различных коммуникационных функций, позволяющих решать большое

количество задач по обмену информацией и обработке данных в реальном времени, при этом соблюдается оптимальный баланс между производительностью и потребляемой мощностью.

Помимо широкого набора интерфейсных модулей микроконтроллеры серии Tiva™ имеют встроенные периферийные устройства для аналоговой и цифровой обработки сигналов, которые совместно с гибкой системой управления потреблением мощности и другими интегрированными функциональными блоками, обеспечивают основу для применения при изготовлении и разработке электронного оборудования систем широкого спектра применений.

Микроконтроллер TM4C1294NCPDT серии Tiva™ обладает большим набором периферии, в том числе и встроенными модулями Ethernet PHY и MAC для прямого (с использованием согласующего трансформатора) подключения к линиям связи стандарта 100 BASE-TX, что позволяет применять микросхемы этого типа при разработке различных сетевых устройств с низкой стоимостью. Основные характеристики микроконтроллера TM4C1294NCPDT приведены в таблице 1.

Таблица 1 — Общие характеристики микроконтроллера TM4C1294NCPDT

Характеристика	Описание
Производительность	
Ядро	ARM Cortex-M4F
Максимальная частота ядра	120 МГц
FLASH (ППЗУ программ и данных)	1024 КБайт
SRAM (ОЗУ данных)	256 Кбайт
EEPROM (ЭСППЗУ)	6 КБайт
Внутренний ROM	Внутренний ROM, загруженный с помощью программного обеспечения TivaWare™ C Series
Внешний периферийный интерфейс (EPI)	8- /16-/ 32-разрядный интерфейс для подключения внешних периферийных устройств

Продолжение таблицы 1

Характеристика	Описание
Безопасность	
Циклический избыточный код (CRC)	16- / 32-битная функция хеширования, которая поддерживает четыре формы CRC
Прерыватель (Tamper)	Поддерживает четыре входных сигнала и конфигурируемый отклик на произошедшее событие
Интерфейсы	
Универсальный асинхронный приемопередатчик (UART)	Восемь модулей UART
Четырехканальный синхронный последовательный порт (QSSI)	Четыре модуля SSI с поддержкой Bi-, Quad- и продвинутой поддержкой SSI
Интерфейс I ² C	Десять модулей I ² C с четырьмя скоростями передачи, включая высокоскоростные
Интерфейс CAN	Два контроллера CAN 2.0 A/B
Ethernet MAC	10/100 Ethernet MAC контроллер
Ethernet PHY	Устройство PHY с поддержкой IEEE 1588 PTP
Интерфейс USB	USB 2.0 OTG/хост/устройство
Системная интеграция	
Контроллер прямого доступа к памяти (μDMA)	32-канальный конфигурируемый контроллер μDMA
Универсальный таймер (GPTM)	Восемь 16/32-битных таймеров
Контрольный таймер (WDT)	Два модуля WDT
Модуль спящего режима (HIB)	Спящий режим с батарейным питанием
Интерфейс ввода/вывода общего назначения (GPIO)	15 физических блоков GPIO
Расширенное управление движением	
Широтно-импульсная модуляция (PWM)	Один модуль PWM с четырьмя блоками генератора ШИМ и блок контроля. 8 выходов PWM.

Продолжение таблицы 1

Характеристика	Описание
Квадратурный декодер (QEI)	Один модуль QEI
Аналоговая периферия	
Аналого-цифровой преобразователь (ADC)	Два 12-битных модуля АЦП
Аналоговый контроллер компаратора	Три независимых интегральных аналоговых компаратора
Цифровой компаратор	16 цифровых компараторов
Интерфейс отладки и программирования JTAG и отладчик SWD	Один отладочный модуль JTAG со встроенным ARM SWD
Информация о корпусе	
Количество входов/выходов	128
Рабочая температура	Промышленный (от минус 40 ° С до 85 ° С) диапазон температур Расширенный (от минус 40 ° С до 105 ° С) диапазон температур

В приложении А приведена архитектура микроконтроллера TM4C1294NCPDT. Микроконтроллер имеет две встроенные шины для связи ядра с периферийными блоками: расширенная периферийная шина (APB); расширенная шина с повышенной производительностью (AHB).

Все микроконтроллеры серии Tiva™ C, включая микроконтроллер TM4C1294NCPDT, разработаны на базе ядра процессора ARM Cortex-M. Ядро процессора ARM Cortex-M обеспечивает высокопроизводительную и недорогую платформу, которая отвечает потребностям минимальной реализации памяти, сокращенного количества выводов и низкого энергопотребления, а также выдающейся вычислительной производительности и исключительной реакции на прерывания.

ARM Cortex-M4F включает в себя интегрированный системный 24-битный таймер SysTick с гибким механизмом управления. Счетчик может быть использован различными способами, например:

- таймер RTOS, который срабатывает при программируемой скорости (например, 100 Гц) и вызывает SysTick;
- таймер высокоскоростного сигнала с помощью системных часов;
- аварийный сигнал с переменной скоростью;
- простой счетчик, используемый для измерения времени до завершения и используемого времени.

Микроконтроллер TM4C1294NCPDT включает в себя ARM контроллер приоритетных векторных прерываний (NVIC). При возникновении определенного заранее события контроллер прерываний автоматически останавливает выполнение основной программы и обращается к соответствующей функции обработки прерываний. После выхода из функции обработки прерываний программа продолжает выполнение с того места, где возникло прерывание.

Интегрированная макроячейка трассировки (ETM) значительно увеличивает возможности отладки, позволяя наблюдать за потоком инструкций и данных внутри ядра CPU в режиме реального времени.

MPU делит карту памяти на регионы, и определяет положение, размер, разрешение на доступ и атрибуты памяти для каждого из них. Поддерживается:

- независимая установка атрибута для каждого региона;
- наложение (перекрытие) регионов;
- экспортирование атрибутов памяти в систему, которые влияют на доступ к памяти в регионе.

Блок вычислений с плавающей точкой (FPU) полностью поддерживает широкий спектр математических операций над вещественными числами, такие как сложение, вычитание, умножение, деление, умножение с накоплением, квадратный корень. Блок FPU также обеспечивает преобразование между данными с фиксированной точкой и данными с плавающей точкой.

Микроконтроллер TM4C1294NCPDT обеспечивает 256 КБ одноканальное

статическое ОЗУ (SRAM). SRAM функционирует с помощью четырех 32-битных отдельных массивов, которые позволяют увеличить скорость доступа к памяти. Память SRAM обеспечивает пропускную способность около 2 ГБ/с при тактовой частоте 120 МГц.

Микроконтроллер TM4C1294NCPDT обеспечивает 1024 Кбайт флэш-памяти. Флэш-память выполнена в виде четырех массивов (4x256 КБ общего объема). Блоки памяти могут быть помечены как «только для чтения» или «только для выполнения», обеспечивая различные уровни защиты кода. Блоки с пометкой «только для чтения» не могут быть стерты или запрограммированы, защищая содержимое этих блоков от изменения. Блоки с пометкой «только для выполнения» не могут быть стерты или запрограммированы, они могут только считываться контроллером выборки инструкций, защищая содержимое этих блоков от чтения контроллером или отладчиком. Микроконтроллер TM4C1294NCPDT обеспечивает повышенную производительность и энергосбережение благодаря реализации двух наборов инструкций предвыборки буферов. Каждый буфер предварительной выборки составляет 2x256 бит, также они могут быть объединены в один 4x256-битный буфер.

Постоянная память (ROM) микроконтроллера TM4C1294NCPDT запрограммирована с помощью следующего программного обеспечения:

- библиотека драйверов периферии TivaWare. Она является встроенным программным обеспечением для управления периферийными устройствами с возможностью использования загрузчика. Библиотека выполняет как инициализацию периферийных устройств, так и функции управления;

- загрузчик TivaWare. Он используется для приложений, требующих программируемости в полевых условиях с поддержкой обновлений программного обеспечения;

- расширенный стандарт шифрования (AES). AES является публичным стандартом шифрования. AES является мощным методом

шифрования с достаточной производительностью и стойкостью. Кроме того, он быстр в аппаратном и программном обеспечении, достаточно легко реализуем, и требует мало памяти. Пакет шифрования Texas Instruments поставляется с полным исходным кодом;

- циклический избыточный код (CRC) с функцией обнаружения ошибок. CRC представляет собой метод для проверки правильного получения сообщений.

Микроконтроллер TM4C1294NCPDT включает в себя EEPROM с памятью в 6 Кбайт, доступной как 1536 32-битных слов.

Внешний периферийный интерфейс (EPI) обеспечивает доступ к внешним устройствам с использованием параллельной шины. В отличие от периферийных устройств связи (SSI, UART и I²C) EPI функционирует подобно шине для внешних периферийных устройств и памяти. EPI имеет следующие особенности:

- 8/16/32-битная выделенная параллельная шина для внешних периферийных устройств и памяти;

- интерфейс памяти поддерживает непрерывный доступ к памяти независимо от размера шины данных таким образом, что позволяет выполнение кода непосредственно из SDRAM, SRAM и флэш-памяти;

- эффективный обмен данными с помощью контроллера доступа μ DMA (отдельные каналы для чтения и записи).

Микроконтроллер TM4C1294NCPDT включает в себя модуль циклический избыточный код (CRC), который служит для правильной передачи сообщений и системы безопасности.

Контроллер TM4C1294NCPDT включает следующие последовательные периферийные интерфейсы:

- 10/100 Ethernet MAC, интегрированный PHY;
- два контроллера CAN 2.0 A/B;

- контроллер USB 2.0 OTG/хост/устройство с дополнительной высокой скоростью, используя внешний PHY;
- восемь UART с IrDA;
- десять модулей I²C с четырьмя скоростями передачи;
- модули четырехканального синхронного последовательного интерфейса (QSSI) с поддержкой двух- и четырехъядерными SSI.

Контроллер Ethernet микроконтроллера TM4C1294NCPDT состоит из полностью интегрированного контроллера управления доступом к среде (MAC) и сетевого физического интерфейса (PHY) со следующими основными характеристиками:

- поддержка 10/100 Мбит скорости передачи данных;
- поддержка полного дуплекса и полудуплекса;
- поддержка четырех фильтров MAC-адресов и программируемого 64-битного хэш-фильтра для фильтрации многоадресных адресов;
- проверка IP-заголовка и аппаратное обеспечение контрольной суммы (IPv4, IPv6, TCP / UDP / ICMP);
- эффективная передача с использованием интегрированного прямого доступа к памяти (DMA).

В микроконтроллере реализовано два независимых контроллера интерфейса CAN. Они являются полнофункциональными CAN-узлами, отвечающими требованиям к активным и пассивным устройствам CAN 2.0A и 2.0B и поддерживающими передачу данных на скорости не более 1 Мбит/сек.

Стандарт универсальной последовательной шины (USB) предназначен для подключения периферийных устройств и отключения с использованием стандартизованного интерфейса без перезагрузки системы. Микроконтроллер TM4C1294NCPDT имеет один контроллер USB, который соответствует стандарту USB 2.0 с функцией высокой скорости. Контроллер USB может иметь три конфигурации: USB-устройство, USB Host и USB On-The-Go.

Универсальный асинхронный приемопередатчик (UART) является интегральной схемой, которая используется для последовательной связи устройств, содержащая передатчик (параллельно-последовательный преобразователь) и приемник (последовательно-параллельный преобразователь), а так же каждый по отдельности содержит генератор тактовой частотой. Микроконтроллер TM4C1294NCPDT включает в себя восемь полностью программируемых UART.

Интерфейс I²C представляет собой шину, обеспечивающую двунаправленную передачу данных через двухпроводную схему (Последовательная линия передачи данных SDA и последовательная линия синхронизации SCL). Шина I²C подключается к внешним устройствам I²C, к таким как память (ОЗУ и ПЗУ), сетевые устройства, ЖК-дисплеи, генераторы тонов и т.д. Шина I²C также может быть использована для тестирования и диагностики системы на момент разработки и производства продукта. I²C имеет четыре скорости передачи: стандартный (100 Кбит/с), быстрый режим (400 Кбит/с), быстрый режим плюс (1 Мбит/с), режим высокой скорости (3,33 Мбит/с).

Четырехканальный синхронный последовательный интерфейс (QSSI) представляет собой двунаправленный интерфейс связи, который преобразует данные между параллельным и последовательным режимами работы. Модуль QSSI выполняет последовательно-параллельное преобразование данных, полученных от периферийного устройства, и параллельно-последовательного преобразование данных, передаваемых на периферийное устройство.

Микроконтроллер TM4C1294NCPDT обеспечивает множество стандартных функций системной интеграции в устройство, в том числе:

- контроллер прямого доступа к памяти (DMA);
- системное управление и часы, включающий встроенный высокоточный генератор 16 МГц;

- восемь 32-разрядных таймера (каждый из которых может быть выполнен в виде двух 16-битных таймеров);
- спящий режим с батарейным питанием;
- часы реального времени в модуле спящего режима;
- два контрольных таймера, один из которых работает от основного генератора, а другой — от прецизионного внутреннего генератора;
- до 90 GPIO, в зависимости от конфигурации.

Микроконтроллер TM4C1294NCPDT включает в себя контроллер прямого доступа к памяти (DMA), известный как микро-DMA (μ DMA). Контроллер μ DMA обеспечивает решение задач передачи данных без использования ресурсов Cortex-M4F, что позволяет более эффективно использовать процессор, повысить производительность. Контроллер μ DMA может выполнять передачу данных между памятью и периферийными устройствами. В нем есть выделенные каналы для каждого поддерживаемого встроенного модуля, которые могут быть запрограммированы автоматически выполнять переходы между периферийными устройствами и памятью.

Модуль системного управления определяет общую работу устройства. В нем содержится информация об устройстве (версия, номер, размер SRAM, размер флэш-памяти, и т.д.). Он управляет функциями энергосбережения, управляет синхронизацией устройства и отдельных периферийных устройств, управляет питанием, ведет отчетность. Модуль также содержит несколько источников синхронизации для системного времени микроконтроллеров. Микроконтроллер TM4C1294NCPDT синхронизируется системными часами (SYSCLK), которые распределяются между процессором и интегрированными периферийными устройствами. В микроконтроллере TM4C1294NCPDT предоставляются следующие источники синхронизации:

- прецизионный генератор с частотой 16 МГц (PIOSC);
- главный генератор (MOSC);

- низкочастотный внутренний генератор (LFIOSC);
- генератор спящего режима RTC (RTCOSC).

Программируемые таймеры могут быть использованы как таймеры или для счетчики внешних событий, которые управляют входными контактами таймера. Каждый 16/32-битный GPTM блок обеспечивает два 16-битных таймера/счетчика, которые могут быть сконфигурированы для работы независимо друг от друга, как таймеры или счетчики событий, или сконфигурированы для работы в качестве одного из 32-битного таймера или одного 32-битного RTC таймера (часы реального времени). Таймеры могут быть также использованы для запуска аналого-цифровой преобразователь (АЦП).

Модуль таймера (GPTM) содержит восемь 16/32-битных GPTM блоков, имеющих следующие режимы работы:

- 16- или 32-битный программируемый одnorазовый таймер;
- 16- или 32-битный программируемый периодический таймер;
- 16-битный универсальный таймер с 8-битным делителем;
- 32-битный в режиме реального времени (RTC) при использовании внешнего 32,768 кГц времени в качестве входного;
- 16-битный режим отсчета времени или времени ввода с 8-разрядным делителем;
- 16-битный режим PWM с 8-битным делителем и программно-программируемой инверсией ввода PWM-сигнала;
- системные часы.

Модуль спящего режима (HIB) обеспечивает логику для отключения питания основного процессора и периферийных устройств и для включения питания на внешних или временных событиях.

Контрольный таймер (Watchdog Timer) используется для восстановления контроля, когда система потерпела неудачу из-за ошибки программного обеспечения или отказа внешнего устройства реагировать ожидаемым образом.

Контрольный таймер может генерировать прерывание, не маскируемое прерывание или сброс, когда достигается значение тайм-аута. Кроме того, контрольный таймер может быть настроен для генерации прерываний микроконтроллера при его первом тайм-ауте и генерировать сигнал сброса на второй тайм-аут. После того, как контрольный таймер настроен, регистр блокировки может быть активирован для предотвращения от непреднамеренного изменения конфигурации таймера. Микроконтроллер TM4C1294NCPDT имеет два модуля контрольного таймера: первый использует системные часы для настройки своего таймера, а второй — использует PIOSC.

Программируемый интерфейс ввода/вывода общего назначения (GPIO) обеспечивают гибкость для различных соединений. Модуль GPIO состоит из 15 физических блоков GPIO, каждому из которых соответствует отдельный порт GPIO. Модуль GPIO поддерживает от 0 до 90 программируемых входных/выходных контактов. Количество доступных GPIO зависит от используемых периферийных устройств.

Микроконтроллер TM4C1294NCPDT обеспечивает продвинутый контроль за потоком (Advanced Motion Control) с помощью функций, интегрированных в устройство, включая:

- восемь продвинутых PWM-выходов для управления потоком;
- четыре входа с ошибками для предотвращения выключений с малой задержкой;
- один вход квадратурного декодера (QEI).

Широтно-импульсная модуляция (PWM) является надежным методом цифрового кодирования уровней аналогового сигнала. Микроконтроллер TM4C1294NCPDT содержит один PWM-модуль с четырьмя блоками генератора PWM и блоком управления, в общей сложности 8 выходов PWM. Каждый блок генератора PWM содержит один таймер (16-разрядный счетчик), два компаратора, генератор сигнала PWM, генератор мертвой зоны и переключатель

прерывания/АЦП. Каждый блок генератора PWM производит два сигнала PWM, которые могут быть либо независимыми сигналами, либо одной парой дополнительных сигналов с задержками в мертвой полосе.

Квадратурный декодер (QE1), также известный как двухканальный инкрементный декодер, преобразует линейное перемещение в импульсный сигнал. Контролируя число импульсов и относительные фазы двух сигналов могут быть отслежены положение, направление вращения, и скорость. Кроме того, третий канал может быть использован для сброса счетчика положения. Микроконтроллер TM4C1294NCPDT включает в себя один модуль QE1, обеспечивающий управление одним двигателем.

Микроконтроллер TM4C1294NCPDT обеспечивает аналоговые функции, встроенные в устройство, в том числе:

- два 12-битных аналого-цифровых преобразователя (АЦП);
- три аналоговых компаратора;
- регулятор напряжения на чипе.

Аналого-цифровой преобразователь (АЦП) является периферийным устройством, которое преобразует непрерывное аналоговое напряжение в дискретное цифровое. Модуль АЦП имеет 12-битное разрешение преобразования и поддерживает 20 входных каналов, а также внутренний датчик температуры. Четыре контроллера последовательностей обеспечивают быструю обработку до 20 аналоговых входных источников без вмешательства контроллера. Каждый контроллер последовательности обеспечивает гибкое программирование с полностью настраиваемым источником входного сигнала, триггером события, генерацией прерываний и приоритетом контроллера последовательностей. Каждый модуль АЦП имеет функцию цифрового компаратора, которая позволяет перевести значение отклонения в блок сравнения.

Аналоговый компаратор является периферийным устройством, которое

сравнивает два аналоговых напряжения и обеспечивает логический выход, который сигнализирует результат сравнения. Микроконтроллер TM4C1294NCPDT обеспечивает три независимых интегрированных аналоговых компаратора, которые могут быть сконфигурированы для управления выходом или генерацией прерывания или событий АЦП.

JTAG и SWD обеспечивают все стандартные функции отладки и тестирования, а также доступ к системной памяти в режиме реального времени, не останавливая ядро или не требуя какого-либо целевого кода.

2.2 Описание асинхронного интерфейса приемопередачи данных - UART

Универсальный асинхронный приёмопередатчик (UART) представляет собой узел вычислительных устройств, который предназначен для организации канала связи между цифровыми устройствами. Для передачи данные преобразуются в последовательный вид таким образом, чтобы было возможно передать их по одной физической цифровой линии другому подобному устройству. Способ преобразования данных для передачи стандартизован и широко применяется в компьютерной технике.

Узел представляет собой логическую схему, с одной стороны подключённую к шине вычислительного устройства, а с другой имеющую два или более выводов для внешнего соединения. Для подключения двух устройств по интерфейсу UART необходимо соединить передатчик первого устройства с приемником второго устройства и приемник первого устройства с передатчиком второго устройства. На рисунке 3 приведена схема подключения двух устройств по интерфейсу UART, где Tx — передатчик, Rx — приемник, GND — земля.

Передача данных по интерфейсу UART осуществляется последовательно по одному биту в равные промежутки времени. Этот промежуток времени, измеряемый в секундах, зависит от скорости UART обратно пропорциональной зависимостью и измеряется в бодах (бит в секунду).

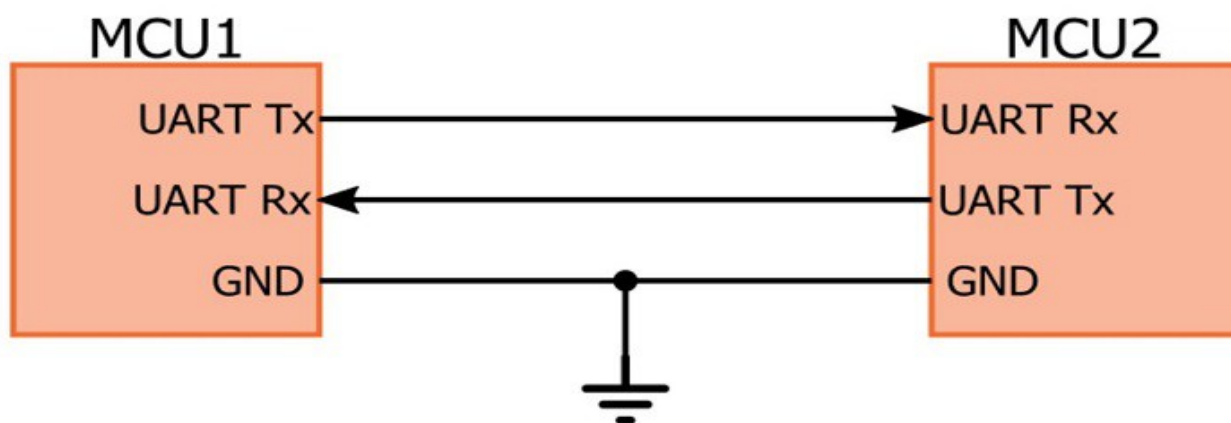


Рисунок 3 — Схема подключения двух устройств по интерфейсу UART

В потоке данных кроме информационных бит, также присутствуют биты синхронизации — стартовый и стоповый биты. При приеме биты синхронизации удаляются из потока данных. В большинстве случаев стартовый и стоповый биты обрамляют восемь бит полезной информации (1 байт), при этом младший информационный бит передается в первую очередь, сразу после стартового. Минимальной посылкой называются информационные биты, обрамленные стартовым и стоповым битами. В отдельных случаях для более надежной синхронизации приемника и передатчика при передаче большого объема информации некоторые реализации UART используют два стоповых бита. Приемник воспринимает второй стоповый бит как короткую паузу в линии.

В момент отсутствия потока данных состояния входа и выхода UART принимают значения логической 1. Стартовый бит всегда принимает значение логического 0, поэтому приёмник UART ждёт перепада из логической 1 в логический 0 и отсчитывает от него промежуток времени в половину длительности бита (середина передачи стартового бита). Если в этот момент на входе UART по-прежнему логический 0, то начинается процесс приема минимальной посылки. Для этого приёмник отсчитывает 9 последовательно идущих битовых длительностей и в каждый момент фиксирует состояние входа. Первые 8 бит являются принятыми информационными битами (минимальной посылкой), последний бит проверочный (стоповый бит). Стоповый бит всегда

принимает значение логической 1, если принятое значение другое, UART фиксирует ошибку.

На рисунке 4 приведен пример передачи данных по интерфейсу UART. В данном примере передается байт 11101101. В момент перехода с логической 1 в логический 0 UART начинает прием информационных данных, а также стопового бита. Завершает прием данных, как только принимается стоповый бит.

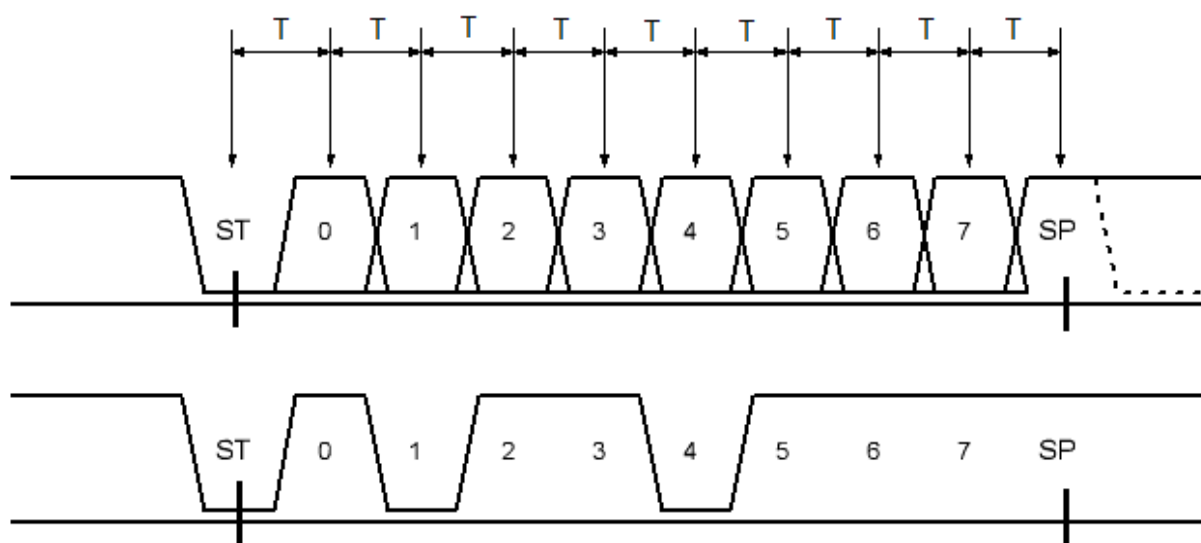


Рисунок 4 — Пример передачи данных по интерфейсу UART

Каждый приемник и передатчик UART для формирования временных интервалов имеют источник тактирования. Точность этого источника должна быть такой, чтобы сумма погрешностей (приёмника и передатчика UART) установки временного интервала от начала стартового бита до середины стопового бита не превышала половины битового интервала. Для 8-битной посылки погрешность равна $0,5/9,5$, что составляет 5% (в реальности не более 3%).

Результирующая пропускная способность UART в действительности меньше скорости соединения, так как биты синхронизации тоже занимают часть битового потока данных. Например, для 8-битных посылок формата биты синхронизации составляют 20% от общего потока данных, что при физической

скорости линии 115200 бит/с в действительности полезную скорость передачи данных составляет 92160 бит/с.

Большинство реализаций UART имеют встроенную возможность автоматически контролировать целостность потока данных с помощью контроля битовой четности. В этом случае последний бит данных в минимальной посылке называется битом четности и контролируется логикой UART. В нем хранятся данные о четности количества бит в минимальной посылке. Если сумма количества бит в минимальной посылке является четной, то это принято называть «контроль на четность», и если сумма не четна, то — «контроль на нечетность». При приеме такой посылки UART автоматически проверяет бит четности и сообщает о наличии или отсутствии ошибки при приеме.

3 Описание программных модулей устройства криптографической защиты IP-трафика

Программное обеспечение для криптографической защиты IP-трафика состоит из четырех программных модулей: модуль инициализации микроконтроллера, модуль приема данных, криптографический модуль и модуль передачи данных.

Модуль инициализации микроконтроллера обеспечивает:

- конфигурирование регистров и периферийных узлов микроконтроллера TM4C1294NCPDT, обеспечивающих прием и передачу сообщений;
- отправку модулю приема данных сообщения о готовности микроконтроллера к использованию.

Модуль приема данных обеспечивает:

- прием от модуля инициализации микроконтроллера сообщения о готовности микроконтроллера к использованию;
- прием от узла локальной сети IP-пакетов с данными, подлежащими зашифрованию, по интерфейсу Ethernet (100 Base-TX), и передачу этих данных в криптографический модуль для дальнейшего зашифрования;
- прием сообщений с зашифрованными данными из канала связи в соответствии с интерфейсом UART с одним стоповым битом без контроля четности со скоростью не менее 5 Мб/с и передачу зашифрованных данных в криптографический модуль для дальнейшего расшифрования.

Криптографический модуль обеспечивает:

- прием данных, подлежащих зашифрованию, от модуля приема данных, зашифрование этих данных в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и передачу зашифрованных данных в модуль передачи данных;

- прием зашифрованных данных от модуля приема данных, расшифрование этих данных в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и передачу расшифрованных данных в модуль передачи данных;
- хранение ключевой информации в области flash-памяти микроконтроллера TM4C1294NCPDT в виде массива данных.

Модуль передачи данных обеспечивает:

- прием зашифрованных данных от криптографического модуля, формирование сообщений с зашифрованными данными и передачу этих сообщений в канал связи в соответствии с интерфейсом UART с одним стоповым битом без контроля четности со скоростью не менее 5 Мб/с;
- прием расшифрованных данных от криптографического модуля, формирование IP-пакетов с расшифрованными данными и передачу этих IP-пакетов по интерфейсу Ethernet (100 Base-TX) узлу локальной сети.

Структура программного обеспечения криптографической защиты IP-трафика приведена на рисунке 5.

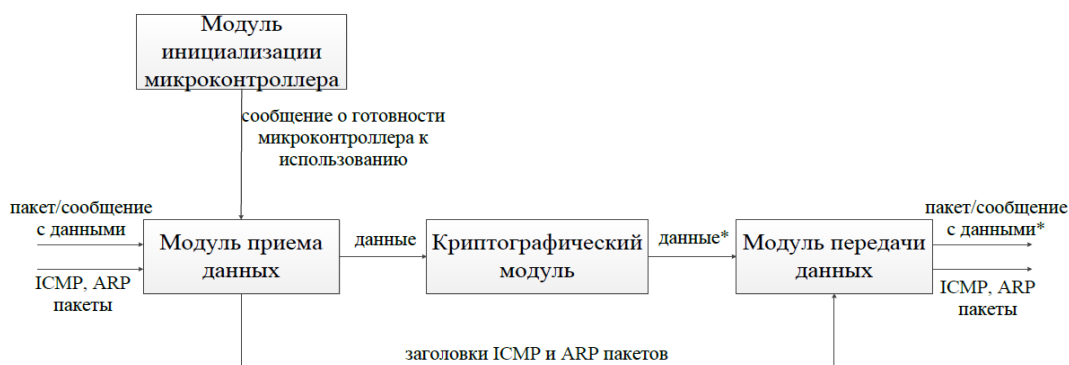


Рисунок 5 — Структура программного обеспечения для криптографической защиты IP-трафика

Модуль инициализации микроконтроллера передает сообщение о готовности микроконтроллера к использованию. Если модуль приема данных принимает пакет/сообщение с данными, то он передает данные в криптографический модуль, если ICMP, ARP пакеты, то он передает заголовки ICMP, ARP пакетов в модуль передачи данных. Криптографический модуль

принимает данные, зашифровывает их и передает в модуль передачи данных. Модуль передачи данных формирует пакет/сообщение с зашифрованными данными или ICMP, ARP пакеты и передает дальше.

После написания программного обеспечения криптографической защиты IP-трафика необходимо продемонстрировать его работоспособность и схему применения, как показано на рисунке 6.

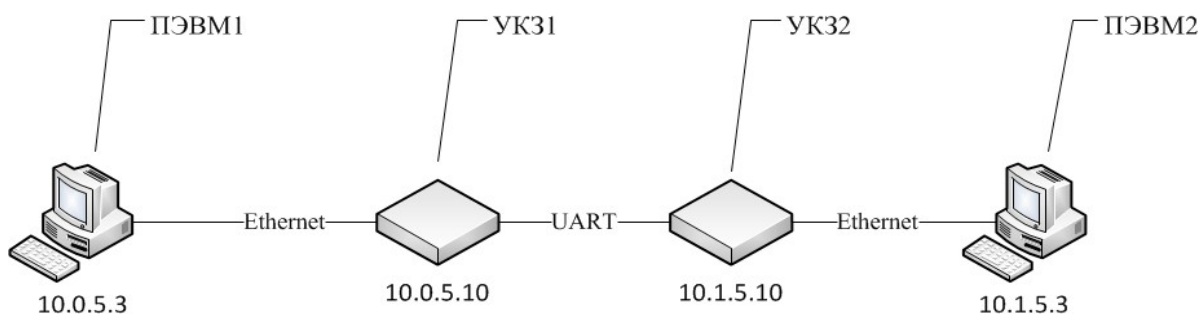


Рисунок 6 — Применение программного обеспечения криптографической защиты IP-трафика

Ethernet-кабель подключен к ПЭВМ1, УК31, УК32, ПЭВМ2 через разъем Ethernet (RJ-45). Для соединения УК31 и УК32 между собой были припаяны три провода в контактах общего назначения, за которыми закреплены функции UART (X1:74, X1:76, X1:36), как показано на рисунке 7. Контакт X1:74 — это передатчик UART. Контакт X1:76 — это приемник UART. Контакт X1:36 — земля. Для организации связи по интерфейсу UART были соединены приемник одного устройства с передатчиком другого устройства и передатчик одного устройства с приемником другого устройства, а также организовано общее заземление.

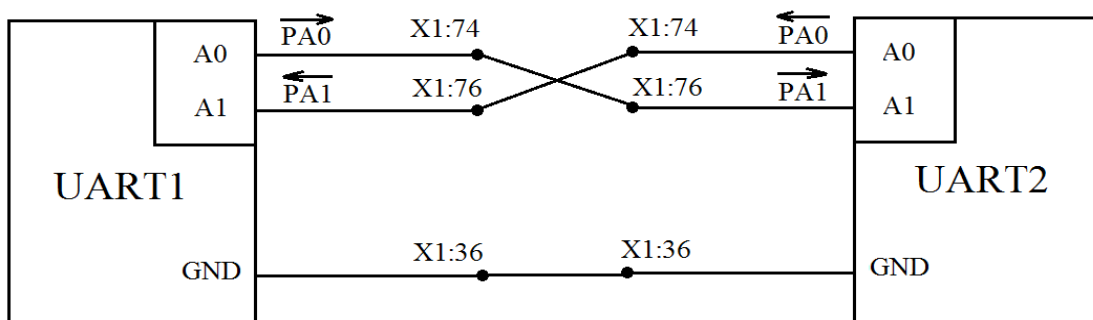


Рисунок 7 — Схема соединения УК31 и УК32 по интерфейсу UART

Как видно из рисунка 7 схема применения состоит из четырех элементов:

- ПЭВМ1, ПЭВМ2 — 2 персональных ЭВМ;
- УКЗ1, УКЗ2 — 2 устройства криптографической защиты (платы, прошитые созданным программным обеспечением).

ПЭВМ1 и УКЗ1 находятся в одной локальной сети с IP-адресом 10.0.5.0. ПЭВМ2 и УКЗ2 находятся в другой локальной сети с IP-адресом 10.1.5.0.

ПЭВМ выполняют функцию отправки и просмотра содержимого пакетов. УКЗ выполняют функции по приему, зашифрованию или расшифрованию данных и передачи пакетов следующему устройству в цепочке.

В данной схеме между УКЗ1 и УКЗ2 организуется защищенный канал связи между двумя устройствами различных локальных сетей, то есть данные в котором передаются только в зашифрованном виде. Секретность данных обеспечивается секретным ключом, который хранится в области flash-памяти микроконтроллера на УКЗ1 и УКЗ2.

Подразумевается, что на ПЭВМ1 будет сформирован IPv4-пакет, с вложенным в него UDP-пакетом и отправлен по интерфейсу Ethernet на УКЗ1. УКЗ1 осуществляет прием пакета по интерфейсу Ethernet, зашифрование полезных данных, формирование нового сообщения с зашифрованными данными и передачу его по интерфейсу UART на УКЗ2. УКЗ2 осуществляет прием сообщения с зашифрованными данными, расшифрование полезных данных, формирование IPv4-пакета с вложенным в него UDP-пакетом и передачу его по интерфейсу Ethernet на ПЭВМ2.

Каждый программный модуль получает одни входные данные, а передает дальше по цепочке другие, то есть каждый программный модуль работает со своим форматом данных. Преобразование форматов данных в программных модулях приведено на рисунке 8.

Модуль приема данных на УКЗ1 принимает IPv4-пакет от ПЭВМ1 по интерфейсу Ethernet (100 Base-TX), с вложенными в него UDP-пакетом и данными, предназначенными для зашифрования. Принятый пакет содержит

заголовки Ethernet, IPv4 и UDP. Содержимое заголовков принятого пакета приведено ниже.

Процедура принятия пакета от ПЭВМ2 на УК32 аналогичная. Содержимое заголовков точно такое же, за исключением полей MAC-получателя, MAC-отправителя IP-адрес отправителя, IP-адрес получателя и значения контрольных сумм в заголовках UDP и IPv4.

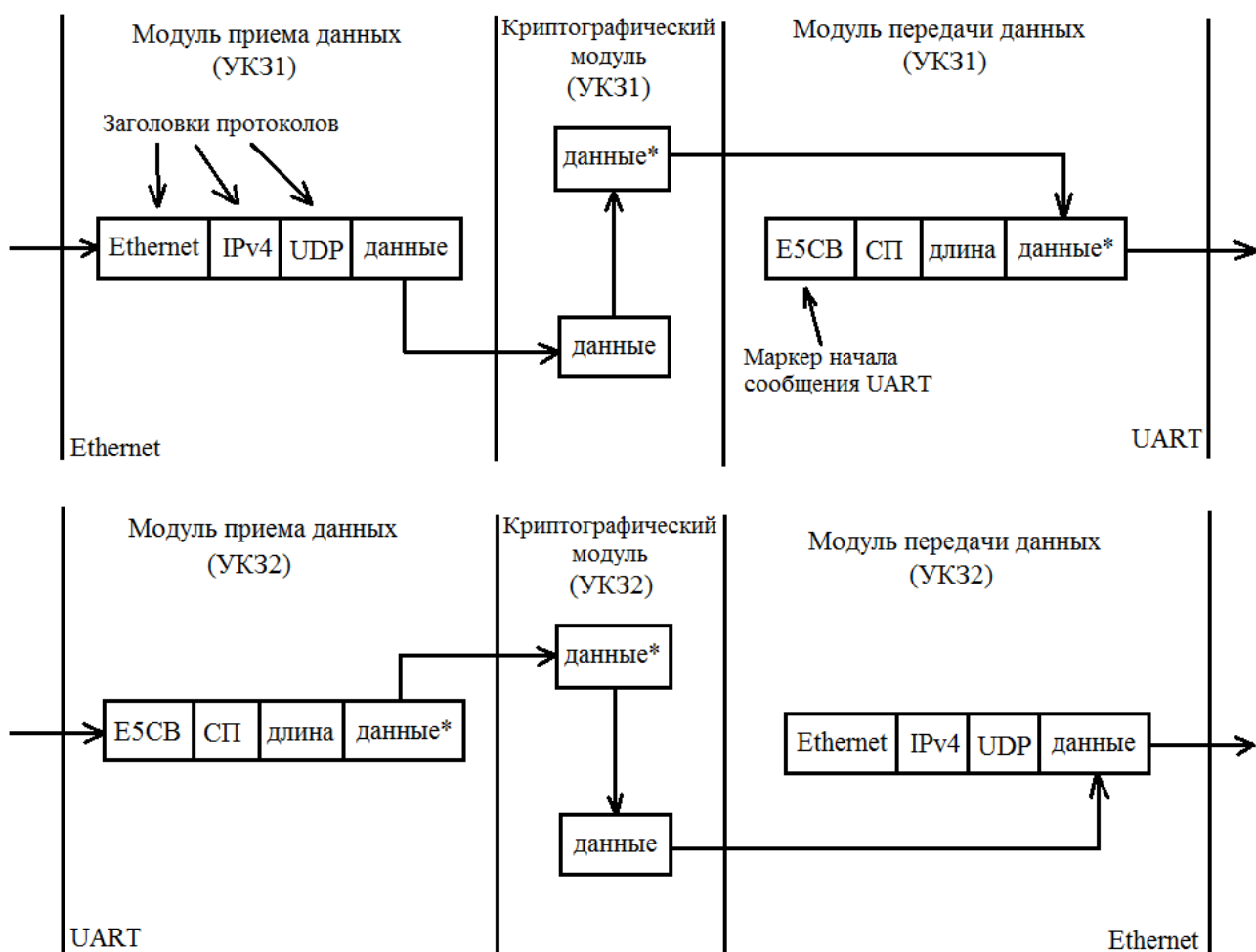


Рисунок 8 — Преобразование форматов данных в программных модулях
Заголовок Ethernet содержит следующие поля:

- преамбула (7 байт). По факту не является частью заголовка. Определяет начала Ethernet-пакета;
- MAC-получателя (6 байт);
- MAC-отправителя (6 байт);
- Ethernet-тип (2 байта). Содержит код протокола верхнего уровня.

ARP – 0x08, 0x06. IPv4 — 0x08, 0x00.

Заголовок IPv4 содержит следующие поля:

- версия протокола (4 бита). Для IPv4 это 0x4;
- размер заголовка (4 бита). Измеряется в 32-битных словах;
- тип службы (1 байт);
- общий размер пакета (2 байта);
- идентификатор (2 байта). Используется для идентификации

фрагментов пакета;

- флаги, связанные с фрагментацией (3 бита);
- смещение фрагмента (13 бит);
- время жизни пакета (1 байт);
- код протокола верхнего уровня (1 байт) . UDP – 17 (0x11);
- контрольная сумма (2 байта);
- IP-адрес отправителя (4 байта);
- IP-адрес получателя (4 байта).

Заголовок UDP содержит следующие поля:

- порт отправителя (2 байта);
- порт получателя (2 байта);
- длина датаграммы (2 байта);
- контрольная сумма (2 байта).

Далее модуль приема данных на УКЗ1 извлекает данные из принятого пакета и передает их криптографическому модулю на УКЗ1 для дальнейшего зашифрования.

Криптографический модуль на УКЗ1 принимает данные, зашифровывает их в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и

передает зашифрованные данные модулю передачи данных на УК31.

Модуль передачи данных на УК31 принимает зашифрованные данные от криптографического модуля и формирует сообщение для передачи по интерфейсу UART. Сформированное сообщение состоит из следующих полей:

- маркер начала сообщения UART (4 байта);
- синхропосылка (8 байт);
- длина сообщения (2 байта);
- зашифрованные данные (1440 байт).

Далее сформированное сообщение передается по интерфейсу UART на УК32 в модуль приема данных.

Модуль приема данных на УК32 принимает сообщение по интерфейсу UART, извлекает зашифрованные данные из него и передает в криптографический модуль на УК32.

Криптографический модуль на УК32 принимает данные, расшифровывает их в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и передает расшифрованные данные модулю передачи данных на УК32.

Модуль передачи данных на УК32 принимает расшифрованные данные, формирует IPv4-пакет, с вложенными в него UDP-пакетом и расшифрованными данными. После чего передает сформированный IPv4-пакет по интерфейсу Ethernet (100 Base-TX) на ПЭВМ2.

4 Разработка программного модуля инициализации микроконтроллера

Был разработан программный модуль инициализации микроконтроллера, основная цель которого — конфигурирование регистров и периферийных узлов микроконтроллера TM4C1294NCPDT, обеспечивающих прием и передачу сообщений.

В приложении Б приведена блок-схема алгоритма функционирования модуля инициализации микроконтроллера.

Весь алгоритм сводится к тому, чтобы сконфигурировать необходимые настройки контроллеров и периферийных узлов.

Вначале устанавливается тактовая частота 120 МГц, которая задается с помощью опорной частоты 25МГц, главного внутреннего генератора и умножителя частоты PLL. Осуществляется сброс всех выводов (пинов) Ethernet и USB. Включается светодиод D1 (PN1), который используется для оповещения пользователя о пришедшем пакете на УКЗ. Конфигурируется Ethernet на работу с внутренним модулем RHY в полнодуплексном режиме 100base-TX. Устанавливается MAC-адрес и IP-адрес для УКЗ для возможности принятия и отправки пакетов между УКЗ и ПЭВМ. Включаются и перезагружаются модули Ethernet (MAC и RHY).

Модуль MAC необходим обеспечения адресации и механизмов управления доступом к каналам, что позволяет нескольким терминалам или точкам доступа общаться между собой в многоточечной сети [4].

Трансивер физического уровня (RHY) — это устройство физического уровня, которое обеспечивает кодирование данных, поступающих от MAC-подуровня для передачи их по транспортной среде, синхронизацию передаваемых данных, а также прием и декодирование данных [4].

Некоторое время ожидается пока MAC-модуль будет готов к использованию. После это конфигурируется внутренний RHY-модуль для его использования. Конфигурируется и перезагружается MAC-модуль, а именно

работа в полнодуплексном режиме, используя 7-байтную преамбулу, с заданным MAC-адресом. Инициализируется контроллер DMA Ethernet, буфер которого будет осуществлять прием и передачу пакетов на ПЭВМ. Устанавливаются параметры фильтрации, то есть чтобы на УКЗ принимались только пакеты, адресованные данному УКЗ, а также широковещательные. Очищаются все ожидающие прерывания, чтобы не возникло случайных прерываний во время функционирования УКЗ. Включаются передатчик и приемник Ethernet MAC. Включается прерывание Ethernet. Включение источника прерываний для Ethernet RX Packet. После этого отправляется сообщение о готовности микроконтроллера к использованию.

5 Разработка программных модулей приема и передачи данных

Был разработан программный модуль приема данных, основными целями которого является:

- прием от узла локальной сети IP-пакетов с данными, подлежащими зашифрованию, по интерфейсу Ethernet (100 Base-TX), и передачу этих данных в криптографический модуль для дальнейшего зашифрования;
- прием сообщений с зашифрованными данными из канала связи в соответствии с интерфейсом UART с одним стоповым битом без контроля четности со скоростью не менее 5 Мб/с и передачу зашифрованных данных в криптографический модуль для дальнейшего расшифрования.

Также был разработан программный модуль передачи данных, основными целями которого являются:

- прием зашифрованных данных от криптографического модуля, формирование сообщений с зашифрованными данными и передачу этих сообщений в канал связи в соответствии с интерфейсом UART с одним стоповым битом без контроля четности со скоростью не менее 5 Мб/с;
- прием расшифрованных данных от криптографического модуля, формирование IP-пакетов с расшифрованными данными и передачу этих IP-пакетов по интерфейсу Ethernet (100 Base-TX) узлу локальной сети.

В приложениях В и Г приведены блок-схемы алгоритмов функционирования модулей приема и передачи данных соответственно. В блок-схемах сначала указаны данные для УКЗ1, а в скобках — для УКЗ2.

Модули приема и передачи обрабатывают 4 вида пакетов/сообщений: ARP-пакеты, IPv4-пакеты, UDP-пакеты, UART-сообщения.

ARP-пакеты обрабатываются в целях определения MAC-адреса, имея IP-адрес получателя. УКЗ отвечает на ARP-запросы, а ARP-ответы игнорирует. IPv4-пакеты в основном используются для проверки соединения. Если пришел ping-запрос, то УКЗ обрабатывает его, формируя ответ. Если пришел ping-ответ,

то УКЗ игнорирует пакет. UDP-пакеты используются для обмена данными. Принятые UDP-пакеты и UART-сообщения обрабатываются в модуле приема данных, в криптографическом модуле и модуле передачи данных, по описанным алгоритмам.

Процесс приема пакета начинается только после того, как принято сообщение о готовности микроконтроллера к использованию. После этого инициализируются проверки криптографического модуля на тестовых данных. Если криптографический модуль не прошел хотя бы одну проверку, то программа попадает в бесконечный цикл, тем самым заканчивая работу программы. Если обе проверки прошли успешно, то происходит инициализация UART и ожидается прерывание принятого пакета. Если принятого пакета нет, то выключается светодиод D1 и ожидается пакет. Если пакет принят, то включается светодиод D1 и очищается флаг события принятого пакета, чтобы после обработки пакета, можно было принять следующий. Если было принято сообщение по интерфейсу UART от УКЗ2 (УКЗ1), то вызывается функция приема сообщения в буфер UART, затем из сообщения извлекаются данные, подлежащие расшифрованию и передаются криптографическому модулю. Если не было принято сообщения по интерфейсу UART, значит был принят пакет от ПЭВМ1 (ПЭВМ2). Далее последовательно определяется какой именно пакет был принят. Если это ARP-запрос или ping-запрос, в котором адрес получателя совпадает с IP-адресом УКЗ1 (УКЗ2), то дальше передается управление модулю передачи данных. Если это UDP-пакет, в котором адрес получателя совпадает с IP-адресом УКЗ1 (УКЗ2), то из пакета извлекаются данные, подлежащие зашифрованию и передаются криптографическому модулю. Во всех остальных случаях ожидается следующий входящий пакет.

Модуль передачи данных зависит от принятого пакета.

Если принятый пакет — это ICMP-пакет (ping-запрос), то формируется ping-ответ. Меняются местами MAC-адрес отправителя на MAC-адрес получателя, IP-адрес получателя и IP-адрес отправителя. Поля тип и код ICMP-

заголовка обнуляются (эхо-ответ). Высчитывается контрольная сумма. Остальные поля заполняются аналогично ping-запросу. После чего вызывается функция отправки пакета (ICMP-ответа) в буфер DMA, который отправит его на ПЭВМ1 (ПЭВМ2).

Если принятый пакет — это ARP-запрос, то формируется ARP-ответ. Меняются местами MAC-адрес отправителя на MAC-адрес получателя, IP-адрес получателя и IP-адрес получателя. Поле Ethernet-тип заполняется значением 0x0806 (протокол верхнего уровня – ARP). Поле HTYPE заголовка ARP заполняется значением 0x0001 (канальный протокол передачи данных — Ethernet). Поле RTYPE заголовка ARP заполняется значением 0x0800 (код сетевого уровня — IPv4). Длина физического адреса HLEN заполняется значением 0x06 (6 байт), длина логического адреса PLEN — 0x04 (4 байта). Код операции opcode заполняется значением 0x0002, так как это ARP-ответ. После чего вызывается функция отправки пакета (ARP-ответа) в буфер DMA, который отправит его на ПЭВМ1 (ПЭВМ2).

Если принятый пакет — это UDP-пакет, то принимаются зашифрованные (расшифрованные) данные от криптографического модуля. Формируется сообщение с зашифрованными данными. Сообщение содержит маркер начала сообщения UART, равный 0xe5cb; синхропосылку, равную 0x0000000012345678; размер данных, равный 0x05a8; зашифрованные данные. Затем вызывается функция отправки сообщения с зашифрованными данными по интерфейсу UART на УК32 (УК31).

Если принятый пакет — UART-сообщение, то формируется UDP-пакет с расшифрованными данными. В UDP-пакете меняются местами MAC-адрес отправителя на MAC-адрес получателя, IP-адрес получателя и IP-адрес получателя. В поле данных помещаются расшифрованные данные. Затем вызывается функция отправки пакета (UDP-пакета) с зашифрованными данными по интерфейсу Ethernet через буфер DMA на ПЭВМ1 (ПЭВМ2).

6 Разработка программного криптографического модуля

Был разработан программный криптографический модуль, основными целями которого являются:

- прием данных, подлежащих зашифрованию, от модуля приема данных, зашифрование этих данных в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и передачу зашифрованных данных в модуль передачи данных;
- прием зашифрованных данных от модуля приема данных, расшифрование этих данных в соответствии с алгоритмом ГОСТ 34.12-2015 [2] с длиной блока n равной 64 бит, в соответствии с ГОСТ 34.13-2015 [3] в режиме гаммирования и передачу расшифрованных данных в модуль передачи данных; хранение ключевой информации в области flash-памяти микроконтроллера TM4C1294NCPDT в виде массива данных.

Блок-схема алгоритма функционирования криптографического модуля приведена на рисунке 9.

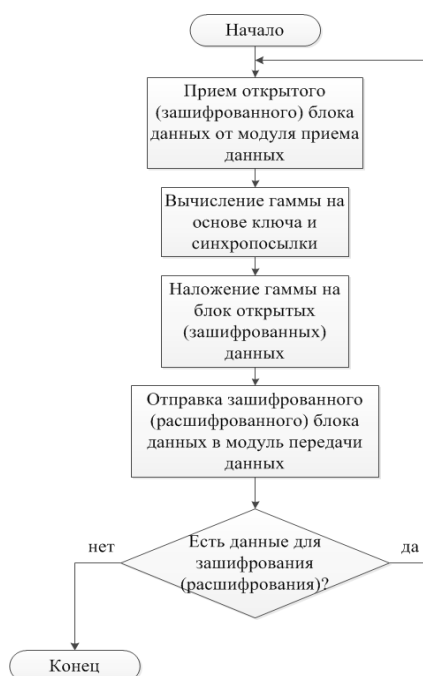


Рисунок 9 — Блок-схема алгоритма функционирования криптографического модуля

Последовательно от модуля приема данных принимаются открытые (зашифрованные) блоки данных размером 64 бит. На основе ключа и синхропосылки вычисляется гамма. Полученная гамма складывается по модулю два с блоком открытого текста, в результате чего получается блок шифртекста. После этого полученный блок шифртекста отправляется в модуль передачи данных. Так продолжается до тех пор пока есть открытые блоки текста, подлежащие зашифрованию.

7 Отладка и тестирование разработанного программного обеспечения на макетной плате ЕК-ТМ4С1294ХЛ

После того, как были разработаны программные модули инициализации микроконтроллера, приема данных, передачи данных и криптографический модуль необходимо провести отладку и тестирование разработанного программного обеспечения на макетной плате ЕК-ТМ4С1294ХЛ.

Для отладки и тестирования использовался Keil uVision5 – среда разработки, которая имеет широкий спектр утилит для осуществления полного комплекса мероприятий по написанию программного обеспечения для микроконтроллеров. Keil uVision5 включает в себя:

- базу данных микроконтроллеров, то есть их конфигурационные данные, а также дополнительную техническую документацию по микроконтроллерам;
- менеджер проектов, используемый для объединения однотипных объектов в группы;
- удобный встроенный редактор;
- средства компиляции, компоновки проекта, ассемблирования, предназначенные для создания загрузочного (исполняемого) файла программы;
- отладчик-симулятор, выполняющий отладку проекта на виртуальной модели микроконтроллера;
- иные утилиты для облегчения выполнения разноплановых задач (Find in File, Source Browser, PC-Lint, Tools Menu, Flash tool).

Был создан проект с подключением микроконтроллера ТМ4С1294НCPDT, в который были добавлены файлы с исходным кодом программных модулей, стандартная библиотека для семейства процессоров ARM Cortex, а также сторонние библиотеки для функционирования программного обеспечения, как показано на рисунках 10, 11 и 12.

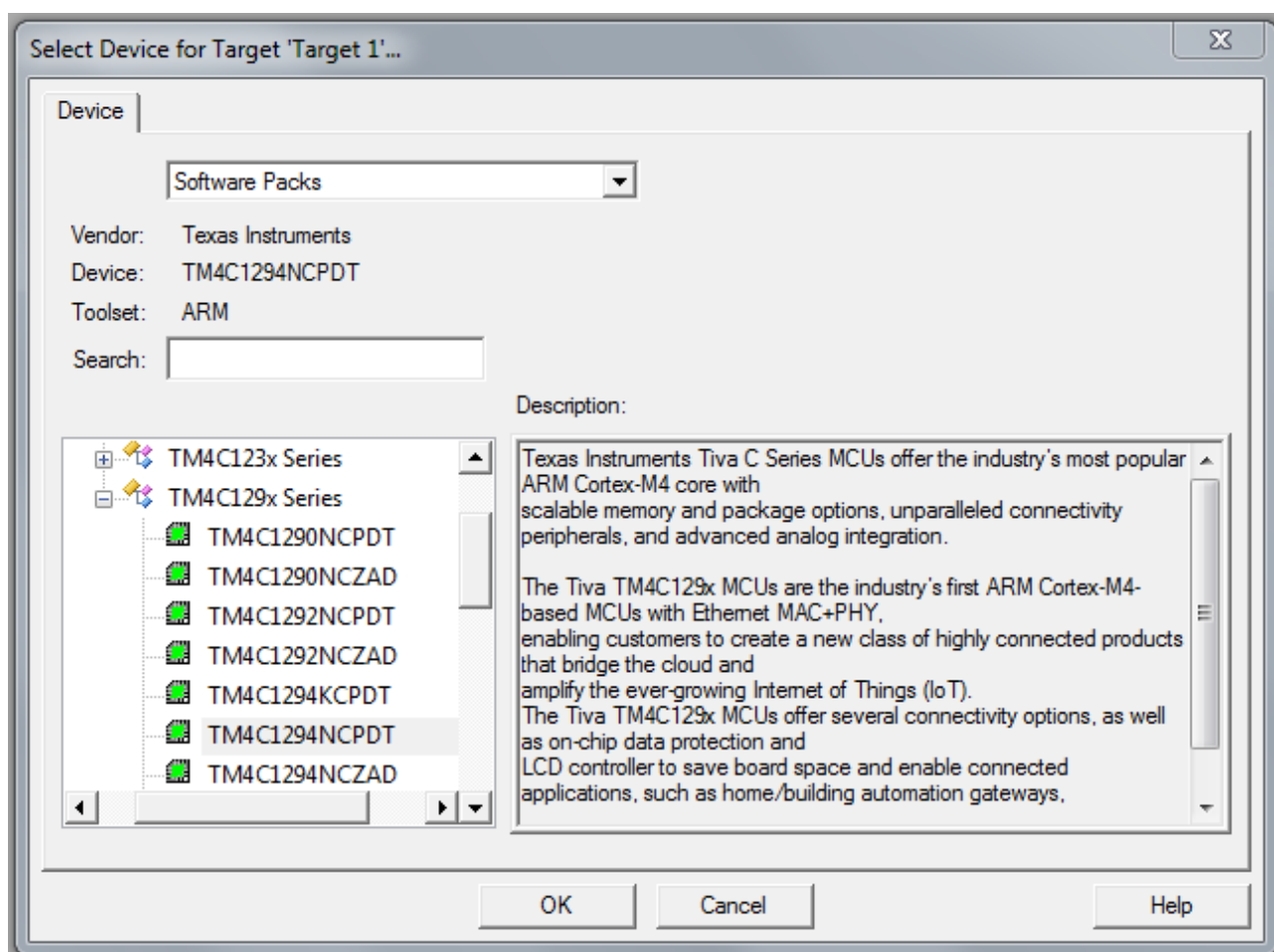


Рисунок 10 — Подключение микроконтроллера TM4C1294NCPDT

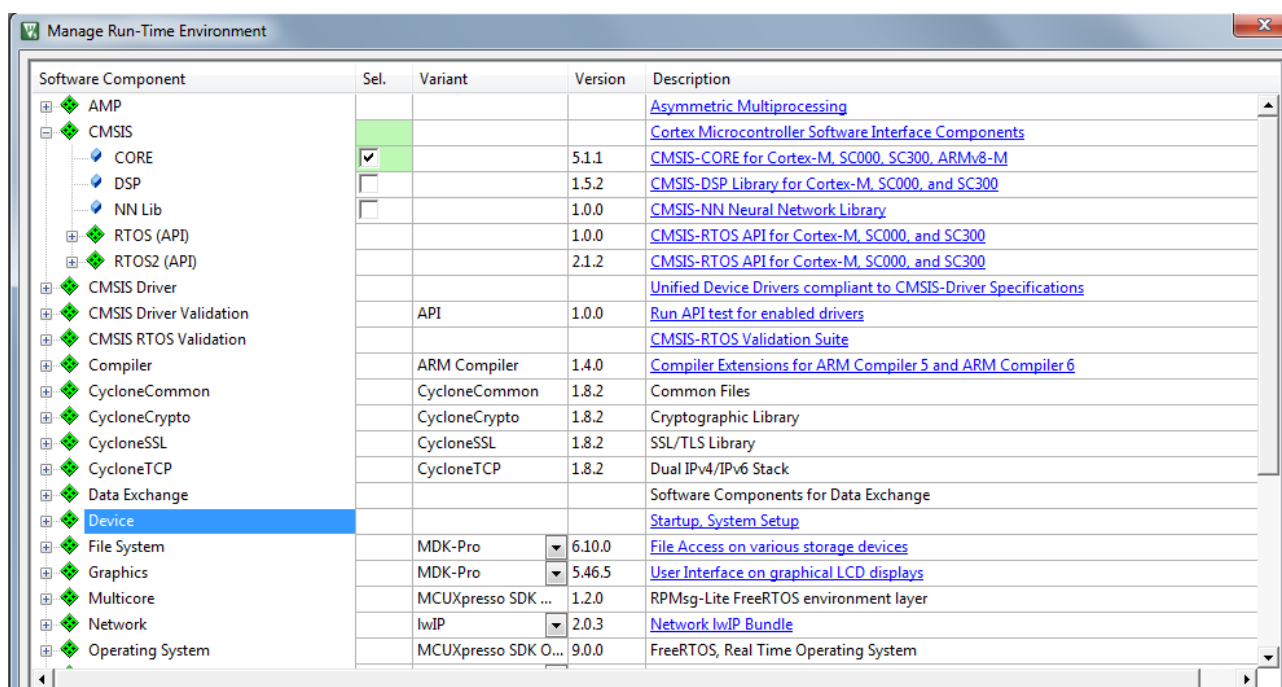


Рисунок 11 — Добавление стандартной библиотеки для семейства процессоров ARM Cortex

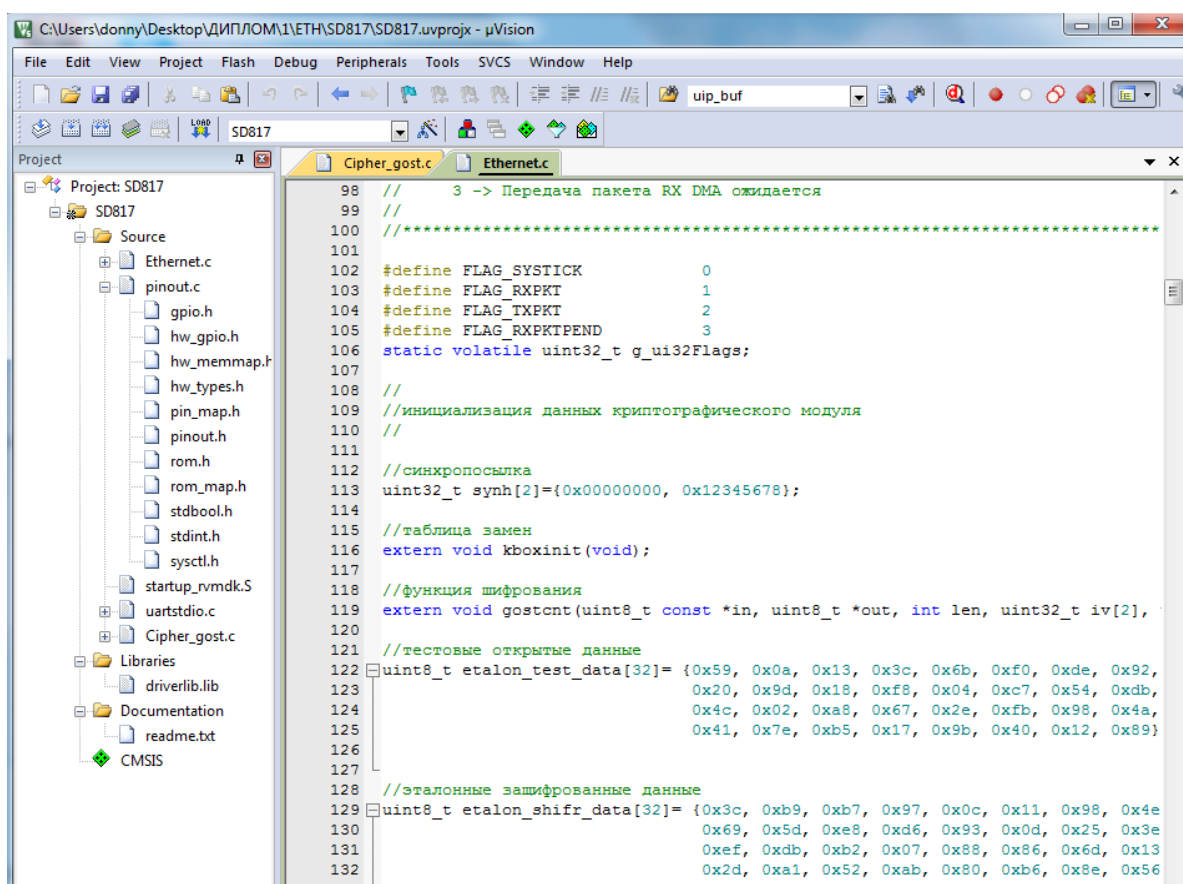


Рисунок 12 — Добавление сторонних библиотеки для функционирования программного обеспечения

В созданном проекте файл «Cipher_gost» содержит криптографический программный модуль, а файл «Ethernet» — программный модуль инициализации микроконтроллера, программный модуль приема данных и программный модуль передачи данных.

Макетная плата EK-TM4C1294XL была подключена к ПЭВМ через разъемы Ethernet (RJ-45) для обменом IPv4 пакетами и microUSB для выполнения отладки проекта, как показано на рисунке 13.

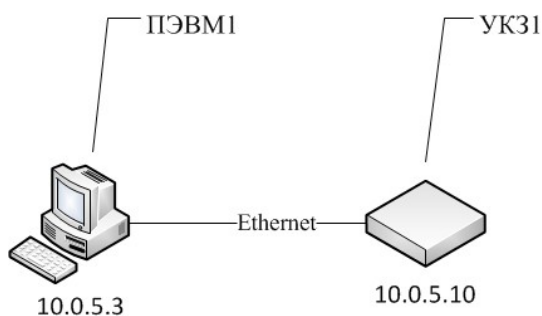


Рисунок 13 — Схема подключения для отладки проекта

С помощью отладки были обнаружены, локализованы и устранены ошибки и предупреждения в проекте. После чего была осуществлена компиляция проекта, результат чего приведен на рисунке 14.

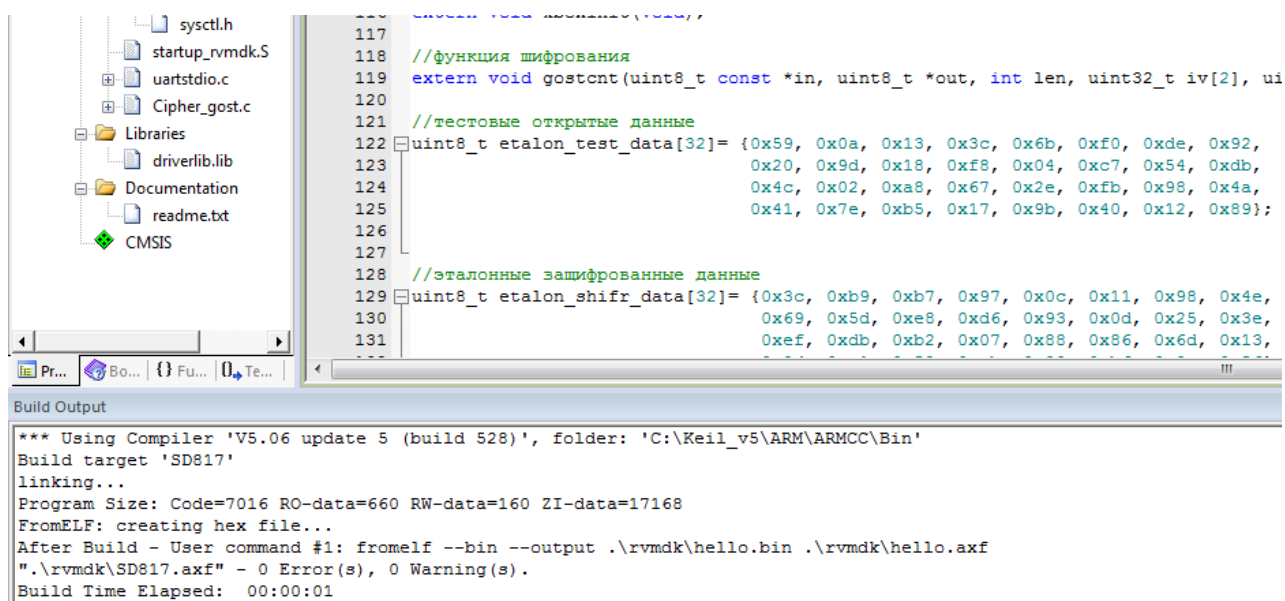


Рисунок 14 — Результат компиляции проекта

Для того, чтобы протестировать программный криптографический модуль на правильность работы, в файле «Ethernet» были созданы контрольные массивы входных данных `test_data` и выходных данных `shifr_data`, в которые будут записываться данные в результате тестирования, а также массивы, содержащие тестовые данные из контрольного примера ГОСТ 34.13-2015 [2], а именно:

- тестовая синхропосылка `synh`: 0x00000000, 0x12345678;
- тестовые входные данные `etalon_test_data`: 0x92, 0xde, 0xf0, 0x6b, 0x3c, 0x13, 0x0a, 0x59, 0xdb, 0x54, 0xc7, 0x04, 0xf8, 0x18, 0x9d, 0x20, 0x4a, 0x98, 0xfb, 0x2e, 0x67, 0xa8, 0x02, 0x4c, 0x89, 0x12, 0x40, 0x9b, 0x17, 0xb5, 0x7e, 0x41;
- тестовые выходные данные `etalon_shifr_data`: 0x4e, 0x98, 0x11, 0xc, 0x97, 0xb7, 0xb9, 0x3c, 0x3e, 0x25, 0x0d, 0x93, 0xd6, 0xe8, 0x5d, 0x69, 0x13, 0x6d, 0x86, 0x88, 0x07, 0xb2, 0xdb, 0xef, 0x56, 0x8e, 0xb6, 0x80, 0xab, 0x52, 0xa1, 0x2d;
- тестовый ключ `test_key`: 0xffeeddcc, 0xbbaa9988, 0x77665544,

0x33221100, 0xf0f1f2f3, 0xf4f5f6f7, 0xf8f9fafb, 0xfcfdfeff.

Созданные контрольные и тестовые массивы данных для тестирования криптографического программного модуля приведены на рисунке 15. Так как способ размещения байтов в системе обратный (little endian), то есть старший байт хранится в памяти по наименьшему адресу, а младший — по наибольшему, то тестовые входные данные `etalon_test_data` и тестовые выходные данные `etalon_shifr_data` вводились в обратном порядке.

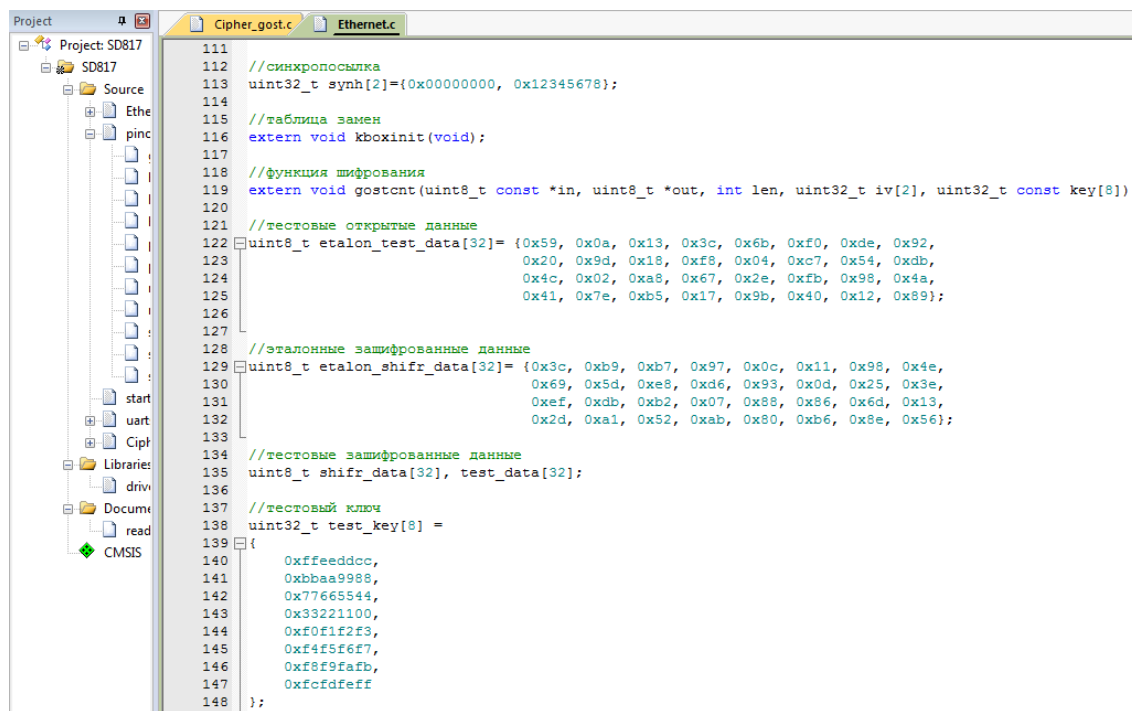


Рисунок 15 — Контрольные и тестовые массивы данных для тестирования криптографического программного модуля

Программный криптографический модуль состоит из функции зашифрования и функции расшифрования, каждую из которых необходимо проверить. Для этого в проект были добавлены две контрольные проверки.

Первая проверка — зашифрование тестовых входных данных с использованием тестовой синхропосылки и тестового ключа и сравнение полученных контрольных выходных данных с тестовыми выходными данными, которые приведены выше. Если полученные контрольные выходные данные `shifr_data` совпадают с тестовыми выходными данными `etalon_shifr_data`, то функция зашифрования в программном криптографическом модуле работает

правильно. Если же полученные контрольные выходные данные `shifr_data` отличаются от тестовых выходных данных `etalon_shifr_data`, то выработанная и эталонная гаммы не совпали, а это говорит о том, что функция зашифрования в программном криптографическом модуле работает некорректно.

Если первая проверка пройдена успешно, то программа перейдет ко второй проверке. Перед второй проверкой, необходимо еще раз проинициализировать синхропосылку `synh` в соответствии с тестовым примером, так как она была изменена в процессе первой проверки. Вторая проверка — расшифрование полученных выходных данных, которые совпадают с тестовыми выходными данными с использованием тестовой синхропосылки и тестового ключа и сравнение полученных контрольных входных данных с тестовыми входными данными, которые приведены выше. Если полученные контрольные входные данные `test_data` совпадают с тестовыми входными данными `etalon_test_data`, то функция расшифрования в программном криптографическом модуле работает правильно. Если же полученные контрольные входные данные `test_data` отличаются от тестовых входных данных `etalon_test_data`, то выработанная и эталонная гаммы не совпали, а это говорит о том, что функция зашифрования в программном криптографическом модуле работает некорректно.

В местах, где циклически осуществляется проверка функций зашифрования и расшифрования в программном криптографическом модуле, были установлены точки останова, с целью сравнения контрольных выходных данных `shifr_data` с тестовыми выходными данными `etalon_shifr_data` и контрольных входных данных `test_data` с тестовыми входными данными `etalon_test_data`. На рисунках 16 и 17 приведены результаты сравнения. Как видно из рисунков 16 и 17 контрольные выходные данные `shifr_data` совпадают с тестовыми выходными данными `etalon_shifr_data` и контрольные входные данные `test_data` также совпадают с тестовыми входными данными `etalon_test_data`. Это говорит о том, что функции зашифрования и

расшифрования работают правильно, а следовательно криптографический программный модуль тоже.

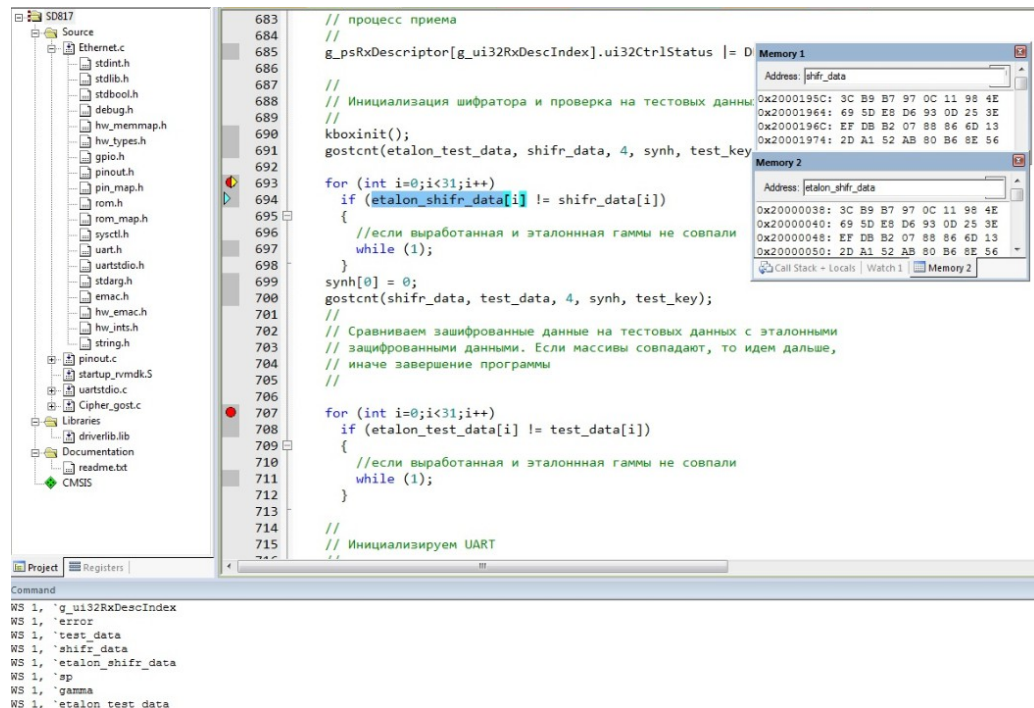


Рисунок 16 — Проверка функции зашифрования в криптографическом программном модуле

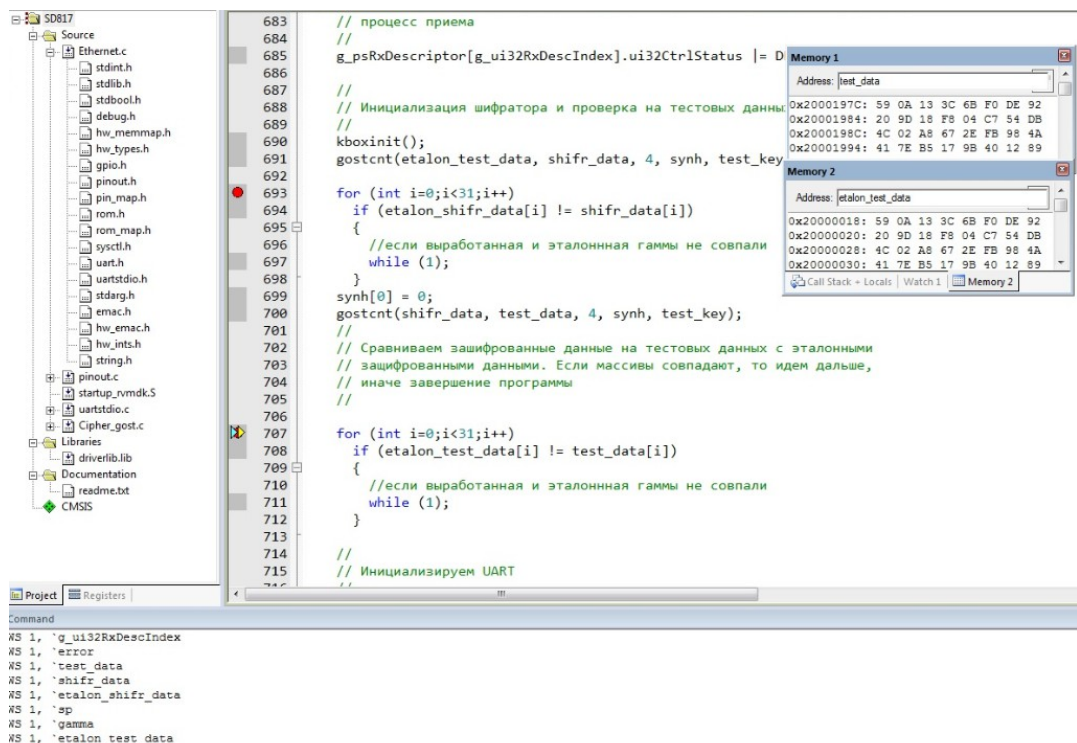


Рисунок 17 — Проверка функции расшифрования в криптографическом программном модуле

Данные проверки были оставлены в проекте с целью недопущения выхода из строя криптографического программного модуля во время работы устройства криптографической защиты IP-трафика. В случае, если проверки прошли успешно, то программа выполняется дальше. В противном случае, программа попадает в бесконечный цикл, что останавливает выполнение программы.

Для тестирования программного обеспечения, то есть всех программных модулей в совокупности на ПЭВМ1 был сформирован IPv4-пакет, с вложенным в него UDP-пакетом и отправлен по интерфейсу Ethernet на УКЗ1. УКЗ1 принял IPv4 пакет, зашифровал содержимое поля данных, сформировал новый IPv4-пакет, с вложенным в него UDP-пакетом и зашифрованными данными и отправил обратно на ПЭВМ1. Корректность отправки, обработки и принятия пакетов проверялась с помощью приложения Wireshark версии 1.10.2. На рисунке 18 приведен сетевой трафик на ПЭВМ1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
2	0.000022000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
3	0.999990000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
4	1.000015000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
5	3.090058000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
6	3.090141000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
7	3.999968000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
8	3.999991000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
9	4.999965000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
10	4.999988000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
11	6.091291000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.5.10? Tell 10.0.5.3
12	6.092121000	TexasIns_02:a2:07	AsustekC_b4:78:c3	ARP	64	10.0.5.10 is at 00:1a:b6:02:a2:07 [ETHER]
13	6.092138000	10.0.5.3	10.0.5.10	TAPV	1482	Tunnel - V=0, T=type 0
14	6.097867000	10.0.5.10	10.0.5.3	UDP	1482	Source port: complex-main Destination
15	7.090301000	fe80::9c27:8a5e:541f::ff02::1:3		LLMNR	89	Standard query 0xad3 A NPI65A53A
16	7.090477000	10.0.5.3	224.0.0.252	LLMNR	69	Standard query 0xad3 A NPI65A53A
17	7.190049000	fe80::9c27:8a5e:541f::ff02::1:3		LLMNR	89	Standard query 0xad3 A NPI65A53A
18	7.190121000	10.0.5.3	224.0.0.252	LLMNR	69	Standard query 0xad3 A NPI65A53A
19	7.390339000	10.0.5.3	10.255.255.255	NBNS	92	Name query NB NPI65A53A<00>
20	8.140014000	10.0.5.3	10.255.255.255	NBNS	92	Name query NB NPI65A53A<00>
21	8.890012000	10.0.5.3	10.255.255.255	NBNS	92	Name query NB NPI65A53A<00>
22	9.641680000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
23	10.499931000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3
24	10.640042000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.200? Tell 10.0.5.3
25	11.499921000	AsustekC_b4:78:c3	Broadcast	ARP	42	who has 10.0.0.203? Tell 10.0.5.3

Рисунок 18 — Сетевой трафик на ПЭВМ1

Как видно из рисунка 18 сформированный пакет на ПЭВМ1 с IP-адресом 10.0.5.3 действительно был отправлен на УКЗ1 с IP-адресом 10.0.5.10, после

чего УКЗ1 принял пакет, обработал его согласно разработанному программному обеспечению, сформировал новый пакет и отправил его обратно.

Содержимое отправленного пакета с данными, подлежащими зашифрованию, от ПЭВМ1 на УКЗ1 и принятого пакета от УКЗ1 на ПЭВМ1 с зашифрованными данными приведено на рисунке 19. Содержимое отправленного пакета от ПЭВМ1 на УКЗ1 с данными, подлежащими зашифрованию, находится на рисунке сверху, а содержимое принятого пакета от УКЗ1 на ПЭВМ1 с зашифрованными данными — снизу.

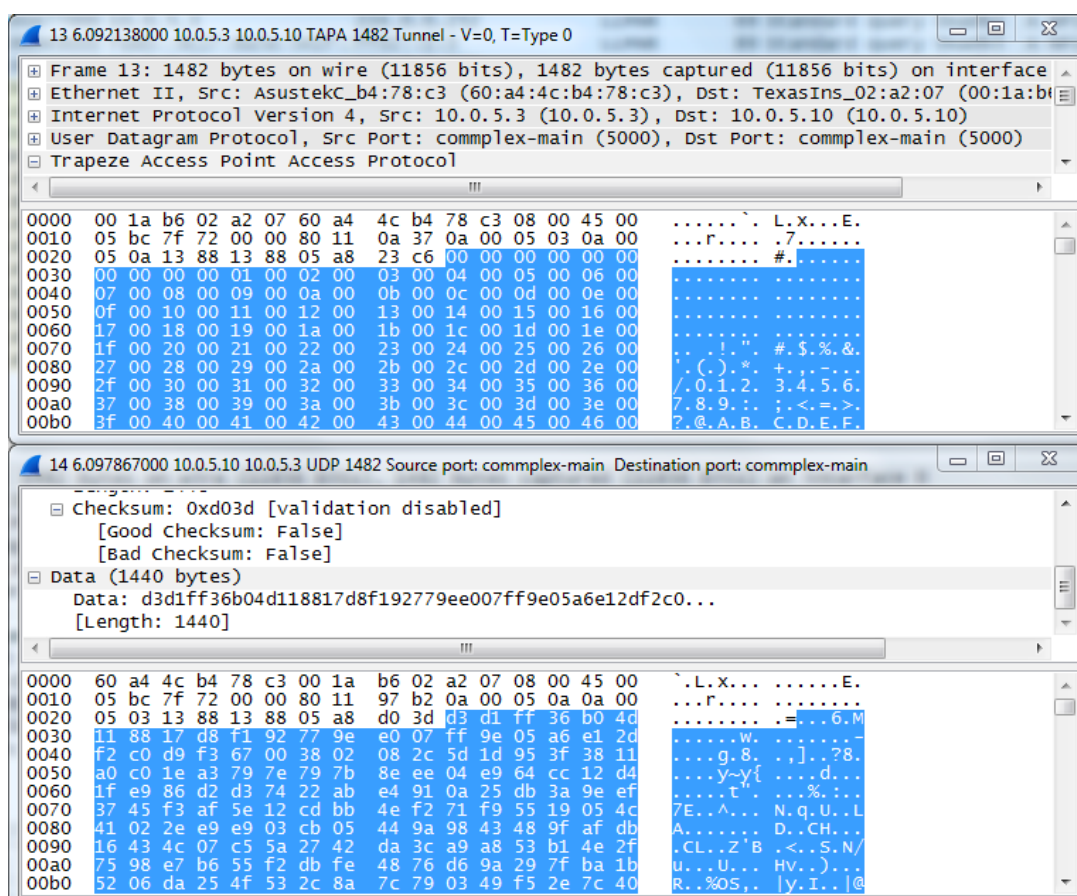


Рисунок 19 — Содержимое отправленного и принятого пакетов на ПЭВМ1 для функции зашифрования

Формируемый пакет состоит из трех заголовков: Ethernet, IPv4, UDP.

Заголовок Ethernet содержит 14 байт, заголовок IPv4 — 20 байт, и заголовок UDP — 8 байт, остальные 1440 байт — данные. Ниже по порядку следования байт расписано содержимое каждого заголовка отправленного пакета с ПЭВМ1 на УКЗ1 и принятого пакета с УКЗ2 на ПЭВМ2.

Содержимое заголовка Ethernet отправленного пакета с ПЭВМ1 на УКЗ1:

- MAC-получателя (6 байт): 0x00, 0x1a, 0xb6, 0x02, 0xa2, 0x07;
- MAC-отправителя (6 байт): 0x60, 0xa4, 0x4c, 0xb4, 0x78, 0xc3;
- Ethernet-тип (2 байта): 0x08, 0x00.

Содержимое заголовка IPv4 отправленного пакета с ПЭВМ1 на УКЗ1:

- версия протокола (4 бита): 0x4;
- размер заголовка (4 бита): 0x5;
- тип службы (1 байт): 0x00;
- общий размер пакета (2 байта): 0x05 0xbc;
- идентификатор (2 байта): 0x7f72;
- флаги (3 бита): 000;
- смещение фрагмента (13 бит): 000, 00000000;
- время жизни пакета (1 байт): 0x80;
- код протокола верхнего уровня (1 байт): 0x11;
- контрольная сумма (2 байта): 0x0a, 0x37;
- IP-адрес отправителя (4 байта): 0x0a, 0x00, 0x05, 0x03;
- IP-адрес получателя (4 байта): 0x0a, 0x00, 0x05, 0x0a.

Содержимое заголовка UDP отправленного пакета с ПЭВМ1 на УКЗ1:

- порт отправителя (2 байта): 0x13, 0x88;
- порт получателя (2 байта): 0x13, 0x88;
- длина датаграммы (2 байта): 0x05, 0xa8;
- контрольная сумма (2 байта): 0x23, 0xc6;
- данные (1440 байт): 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 (указаны первые 6 байт).

Содержимое заголовка Ethernet принятого пакета с УКЗ1 на ПЭВМ1:

- MAC-получателя (6 байт): 0x60, 0xa4, 0x4c, 0xb4, 0x78, 0xc3;

- MAC-отправителя (6 байт): 0x00, 0x1a, 0xb6, 0x02, 0xa2, 0x07;
- Ethernet-тип (2 байта): 0x08, 0x00.

Содержимое заголовка IPv4 принятого пакета с УКЗ1 на ПЭВМ1:

- версия протокола (4 бита): 0x4;
- размер заголовка (4 бита): 0x5;
- тип службы (1 байт): 0x00;
- общий размер пакета (2 байта): 0x05, 0xbc;
- идентификатор (2 байта): 0x7f72;
- флаги (3 бита): 000;
- смещение фрагмента (13 бит): 000, 00000000;
- время жизни пакета (1 байт): 0x80;
- код протокола верхнего уровня (1 байт): 0x11;
- контрольная сумма (2 байта): 0x97, 0xb2;
- IP-адрес отправителя (4 байта): 0x0a, 0x00, 0x05, 0x0a;
- IP-адрес получателя (4 байта): 0x0a, 0x00, 0x05, 0x03.

Содержимое заголовка UDP принятого пакета с УКЗ1 на ПЭВМ1:

- порт отправителя (2 байта): 0x13, 0x88;
- порт получателя (2 байта): 0x13, 0x88;
- длина датаграммы (2 байта): 0x05, 0xa8;
- контрольная сумма (2 байта): 0xd0, 0x3d;
- данные (1440 байт): 0xd3, 0xd1, 0xff, 0x36, 0xb0, 0x4d, 0x11

(указаны первые 6 байт).

После этого на ПЭВМ1 был сформирован пакет с содержимым принятого пакета, который содержит зашифрованные данные, и отправлен на УКЗ1 с целью их расшифрования.

Содержимое отправленного пакета с данными, подлежащими

расшифрованию, от ПЭВМ1 на УКЗ1 и принятого пакета от УКЗ1 на ПЭВМ1 с расшифрованными данными приведено на рисунке 20. Содержимое отправленного пакета от ПЭВМ1 на УКЗ1 с данными, подлежащими расшифрованию, находится на рисунке сверху, а содержимое принятого пакета от УКЗ1 на ПЭВМ1 с расшифрованными данными — снизу.

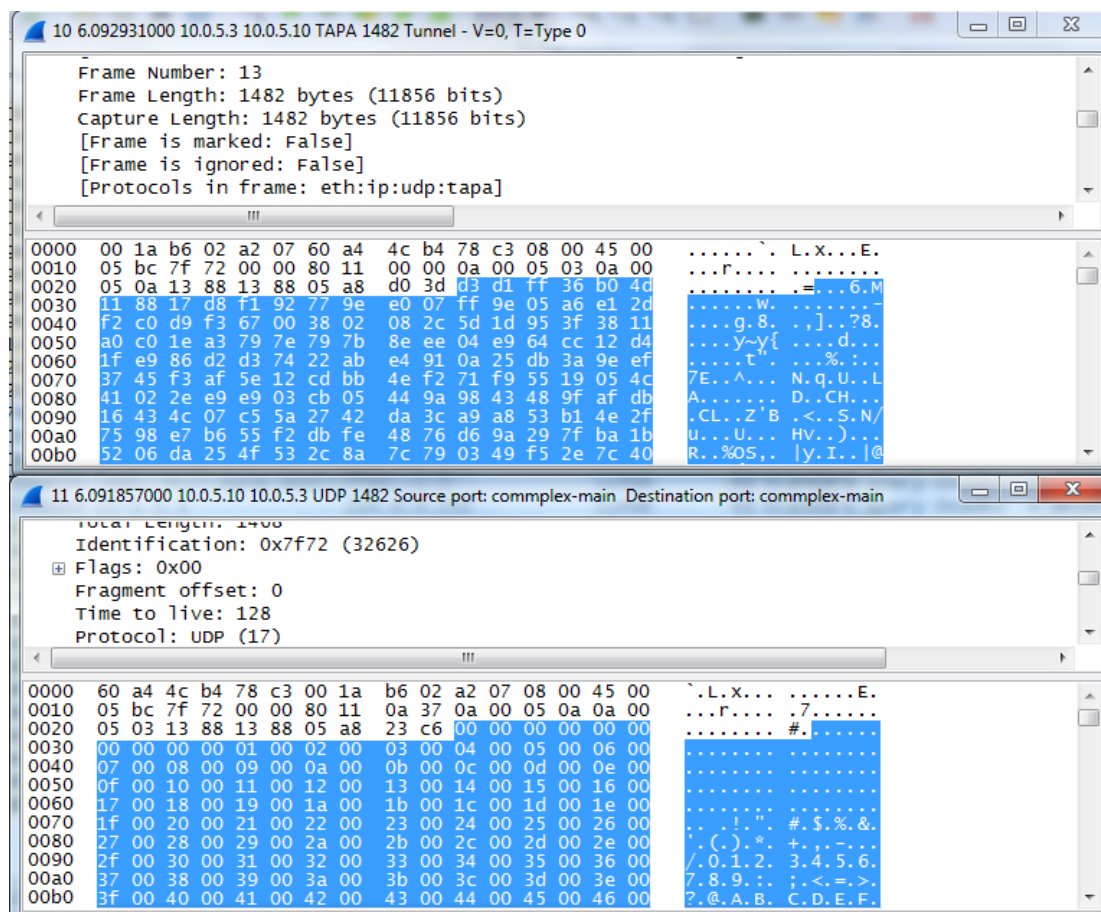


Рисунок 20 — Содержимое отправленного и принятого пакетов на ПЭВМ1 для функции расшифрования

Как видно из рисунков 19 и 20 процедурой расшифрования зашифрованных данных были получены первоначальные открытые данные. Это свидетельствует о правильности работы программного обеспечения криптографической защиты IP-трафика.

После этого была собрана схема в рабочем режиме, предназначенная для демонстрации применения программного обеспечения криптографической защиты IP-трафика, которая приведена в разделе 3 на рисунке 6.

Предварительно на ПЭВМ1 и ПЭВМ2 были запущены приложения Wireshark версии 1.10.2, чтобы отследить среди сетевого трафика наши отправленные и принятые пакеты.

Снова на ПЭВМ1 был сформирован IPv4-пакет с вложенным в него UDP-пакетом и данными, подлежащими зашифрованию, и отправлен на УК31. УК31 принял IPv4-пакет с вложенным в него UDP-пакетом и данными, подлежащими зашифрованию по интерфейсу Ethernet, зашифровал эти данные, сформировал сообщение и отправил его по интерфейсу UART на УК32. УК32 принял сообщение с данными, подлежащими расшифрованию, по интерфейсу UART, расшифровал эти данные, сформировал IPv4-пакет, с вложенным в него UDP-пакетом и расшифрованными данными и отправил его по интерфейсу Ethernet на ПЭВМ2.

На рисунках 21 и 22 приведено содержимое отправленного от ПЭВМ1 и принятого на ПЭВМ2 IPv4-пакетов соответственно.

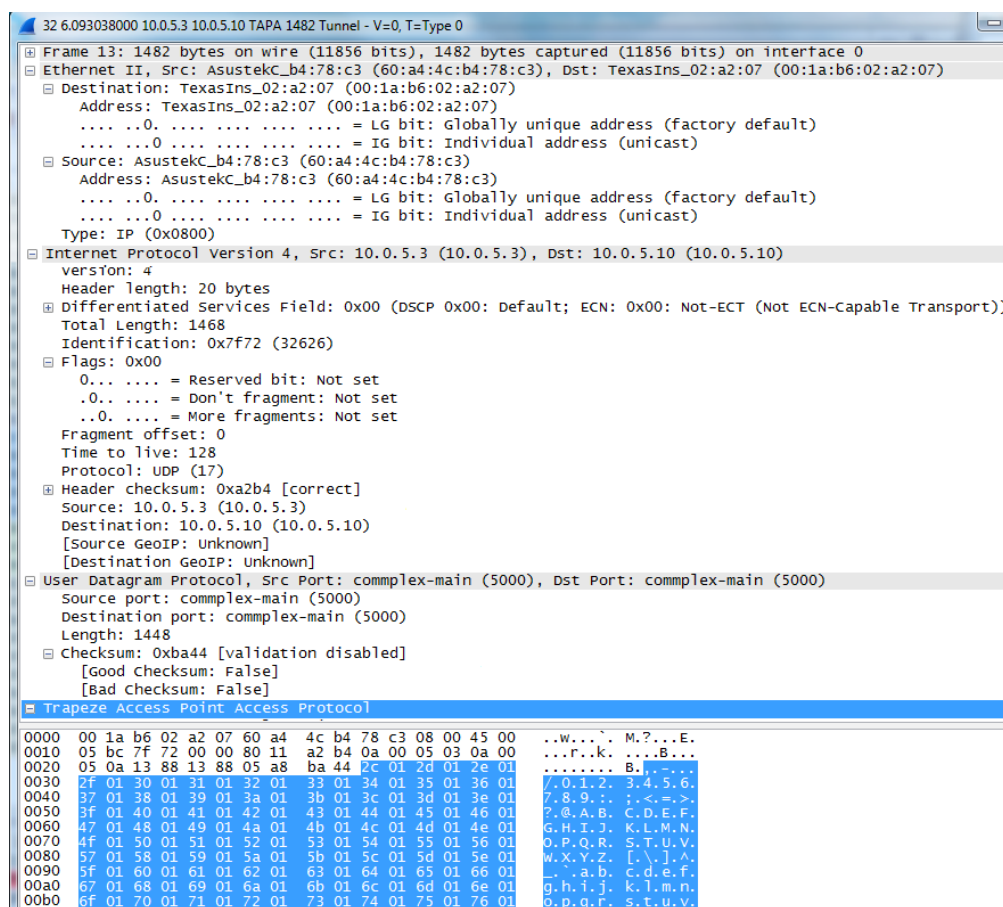


Рисунок 21 — Содержимое отправленного от ПЭВМ1 IPv4-пакета

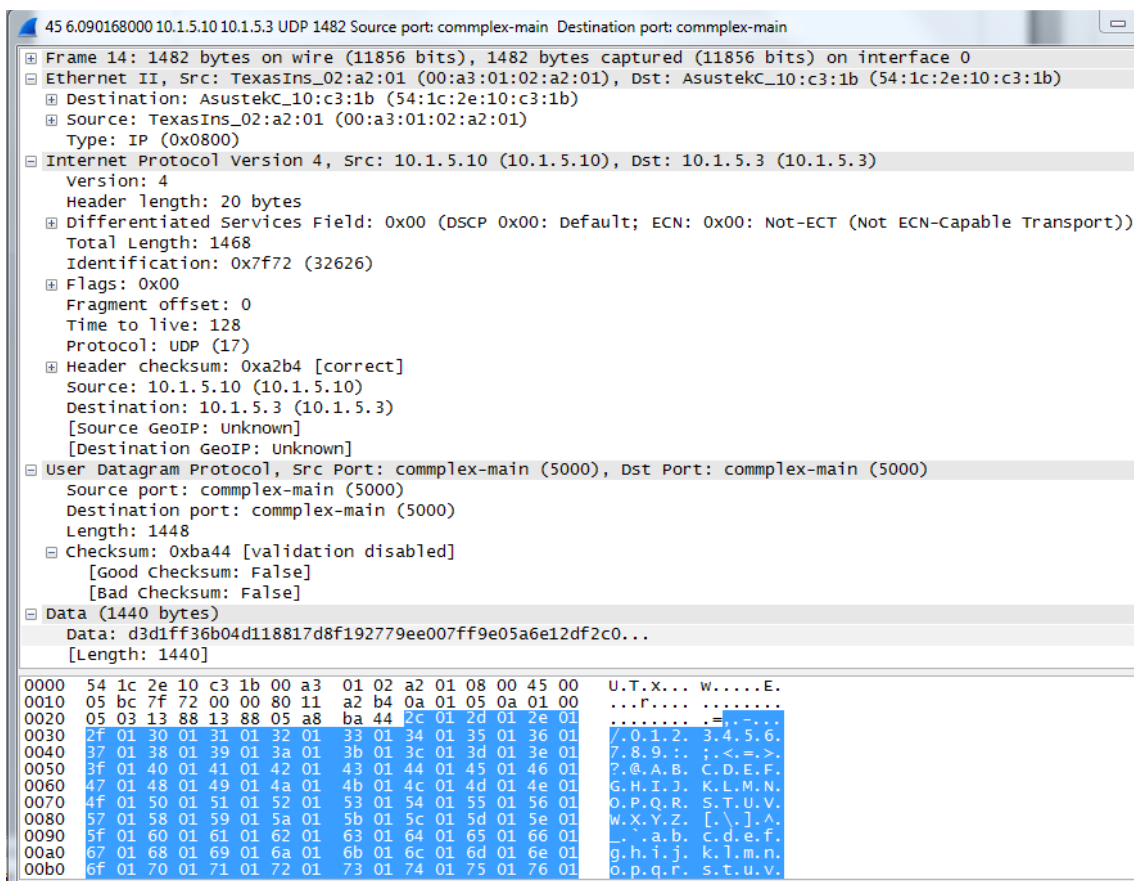


Рисунок 22 — Содержимое принятого на ПЭВМ2 IPv4-пакета

Как видно из рисунков 21 и 22 содержимое поля данных совпадает, это свидетельствует о правильности работы программного обеспечения криптографической защиты IP-трафика.

Таким образом, между УК31 и УК32 был организован защищенный канал связи между двумя устройствами различных локальных сетей, то есть данные в котором передаются только в зашифрованном виде. Секретность данных обеспечивается секретным ключом, который хранится в области flash-памяти микроконтроллера на УК31 и УК32.

Листинг разработанного программного обеспечения приведен в приложении Д.

8 Вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Специалисту по защите информации по роду своей деятельности приходится, как никому другому, выполнять свои должностные обязанности долгое время в сидячем положении, работая за компьютером. В связи с этим, такой малоподвижный образ жизни негативно влияет на здоровье и жизнедеятельность человека. Могут возникать и развиваться заболевания кистей рук и локтевых суставов, позвоночника, нервной системы, кожи, органов дыхания и пищеварения, прямой кишки, заболевания, связанные с нарушением зрения. Но одними из самых опасных и распространенных являются заболевания сердечно-сосудистой системы.

К вредным психофизиологическим факторам, влияющим на сердечно-сосудистую систему специалиста по защите информации относят:

— малоподвижный образ жизни. Это вызывает проблемы с кровообращением, которые могут привести к повышенному тромбообразованию. Эритроциты сворачиваются, тем самым нарушая насыщение тканей кислородом, в том числе сердечной мышцы. Постепенно замедляется обмен веществ и растет лишний вес, что вынуждает сердечную мышцу работать более усиленно;

— постоянное нахождение в сидячем положении. Это приводит к застою крови в нижних конечностях и повышенному растяжению клапанов вен. Из-за чего стенки сосудов растягиваются и становятся менее эластичными, что вызывает развитие варикозной болезни;

— электромагнитное излучение. Постоянное воздействие электромагнитного излучения негативно влияет на форменные элементы крови, клетки. Они имеют электрические потенциалы, положительные и отрицательные заряды. Воздействие излучения происходит на тонком уровне,

поэтому развитие заболевания невозможно предугадать и обнаружить. Также излучение может влиять на селезенку, спинной мозг, ухудшать процесс кроветворения. Постепенно угнетается гормональная система, которая вырабатывает большое количество адреналина в ответ на длительное воздействие излучения. Это негативно влияет на артериальное давление и затрудняет работу сердечной мышцы. В редких случаях излучение является причиной аритмий и может негативно сказываться на проводимости миокарда;

- нагрузка на кисти рук. Также в процессе деятельности человек, работающий за компьютером, совершает однообразные движения кистями рук, что усиливает приток крови к работающим частям тела. Это может вызвать защемление нервов и негативно сказаться на кровообращении человека.

Чтобы снизить негативное воздействие психофизиологических факторов на здоровье специалиста по защите информации необходимо, в первую очередь, соблюдать профилактические меры. К ним относятся:

- ограничение непрерывной работы за компьютером. По санитарным нормам человеку допускается работать за компьютером не более 6 часов;

- уход за зрением. Необходимо выполнять регулярную разминку глаз, при необходимости принимать витамины и минералы для зрения и использовать капли от сухости глаз;

- периодические перерывы во время работы для соблюдения режима питания;

- выбор качественного жидкокристаллического монитора. Монитор не должен обладать эффектом мерцания. Также при необходимости можно использовать специальный защитный экран;

- правильное положение тела за рабочим местом. На рисунке 23 показано правильное положение тела за рабочим местом. Монитор должен находиться от глаз на расстоянии вытянутой руки желательно под прямым углом. Необходимо держать спину ровно, плечи расправить. Руки

рекомендуется держать в согнутом положении под углом 90 градусов. Не следует опираться локтями на стол. Сидя на стуле ноги у человека должны быть в согнутом положении под углом чуть больше 90 градусов, ступни должны находиться на полу или на подставке;

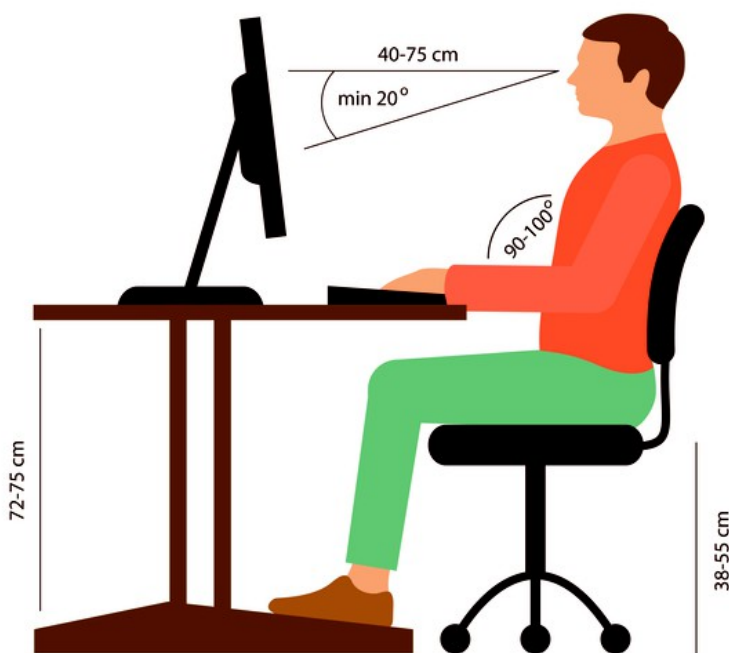


Рисунок 23 — Правильное положение тела за рабочим местом

- использование комфортной и подходящей по анатомическим признакам мебели;
- регулярная уборка рабочего места и помещения от пыли с помощью антибактериальных средств. Также следует периодически проветривать помещение;
- регулярное препровождение личного времени на свежем воздухе.

Помимо вышеперечисленного регулярно необходимо уделять время физической культуре, а именно выполнять упражнения, которые снижают негативное воздействие психофизиологических факторов на сердечно-сосудистую систему специалиста по защите информации. К таким

упражнениям относятся:

— упражнение 1. Сидя за стулом, необходимо завести руки за спинку стула, ноги перекрестить и поднять от пола в горизонтальное положение. Также можно передвигать ноги поочередно то влево, то вправо, при этом напрягая мышцы ног;

— упражнение 2. Необходимо взять лист бумаги, положить его на пальцы обеих рук, поднять его вверх и удерживаться в таком положении несколько секунд;

— упражнение 3. Необходимо встать возле рабочего места, ноги расположить на ширине плеч и, держась руками о стул или стол, выполнить несколько приседаний;

— упражнение 4. Необходимо встать прямо и сцепить руки в замок за спиной. После этого нужно приподняться на носки, при этом расцепив руки и вытянув их вверх. Также можно при подъеме выполнять повороты рук поочередно влево и вправо, одновременно выполняя повороты головой. Данное упражнение уменьшает напряжение в мышцах ног, спины и шеи;

— упражнение 5. Необходимо встать ровно и расслабиться. Затем поочередно выполнять глубокий вдох, задержку дыхания и выдох.

Физическая зарядка за рабочим местом очень полезна. Она уменьшает риск заболеваний связанных с сердечно-сосудистой системой, повышает продуктивность и работоспособность.

Заключение

В результате проделанной работы были описаны: аппаратная платформа устройства криптографической защиты IP-трафика, архитектура микроконтроллера TI MSP, асинхронный интерфейс приемопередачи данных – UART, программные модули устройства криптографической защиты IP-трафика. Были разработаны программные модули инициализации микроконтроллера, приема и передачи данных, а также криптографический модуль. Была проведена отладка и тестирование разработанного программного обеспечения на макетной плате EK-TM4C1294XL. Были рассмотрены вредные психофизиологические факторы, влияющие на сердечно-сосудистую систему специалиста по защите информации, методы и средства физической культуры, снижающие их воздействие.

В результате выполнения дипломного проекта было разработано программное обеспечение для криптографической защиты IP-трафика на базе специализированного микроконтроллера TM4C1294NCPDT.

Таким образом, все требования технического задания были выполнены в полном объеме.

Список используемых источников

- 1 Texas Instruments [Электронный ресурс]: Tiva™ TM4C1294NCPDT Microcontroller. DATA SHEET. Режим доступа: <http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf>, свободный, дата доступа: 07.06.18;
- 2 ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры»;
- 3 ГОСТ Р 34.13-2015 «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров»;
- 4 Компэл [Электронный ресурс]: Промышленный Ethernet. Режим доступа: <https://www.compel.ru/lib/ne/2010/12/6-promyshlennyiy-ethernet-ot-st-vsyo-i-srazu>, свободный, дата доступа: 07.06.18.

Приложение А
(обязательное)
Архитектура МК TM4C1294NCPDT

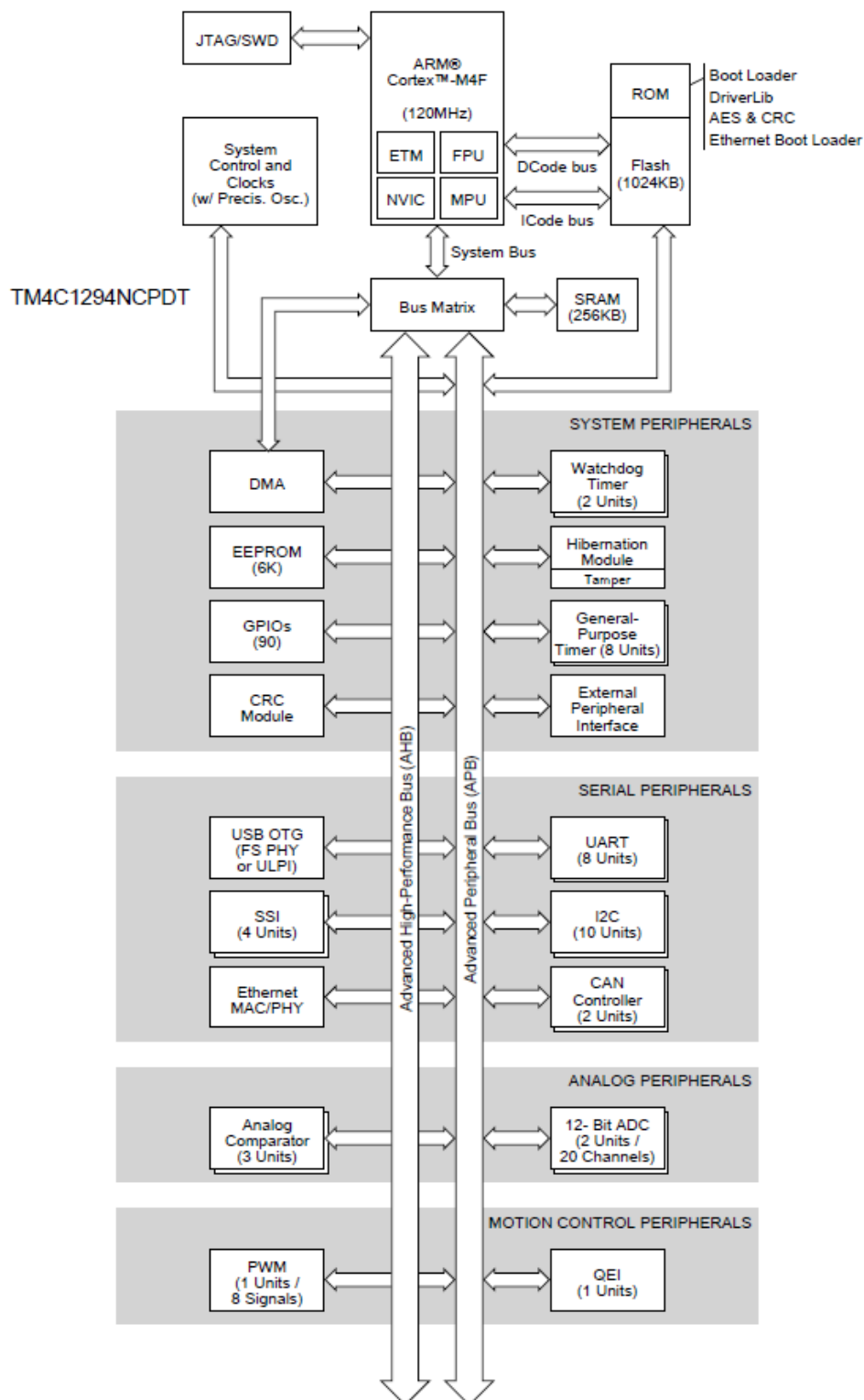


Рисунок А.1

Приложение Б

(обязательное)

Блок-схема алгоритма функционирования модуля инициализации
микроконтроллера

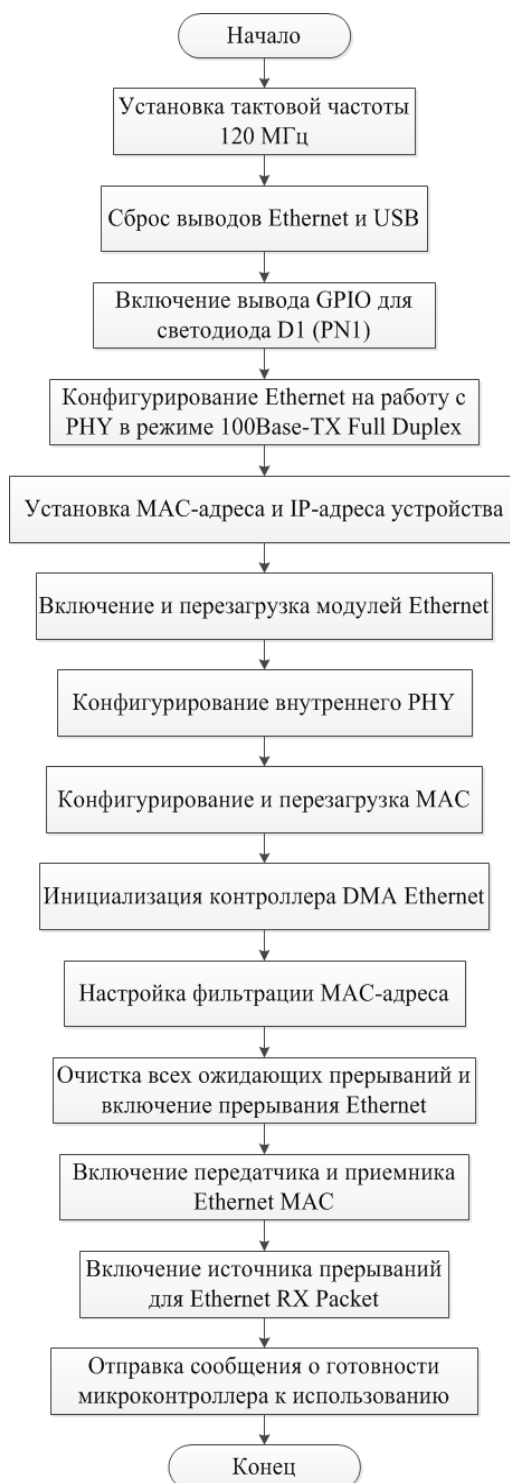


Рисунок Б.1

Приложение В

(обязательное)

Блок-схема алгоритма функционирования модуля приема данных

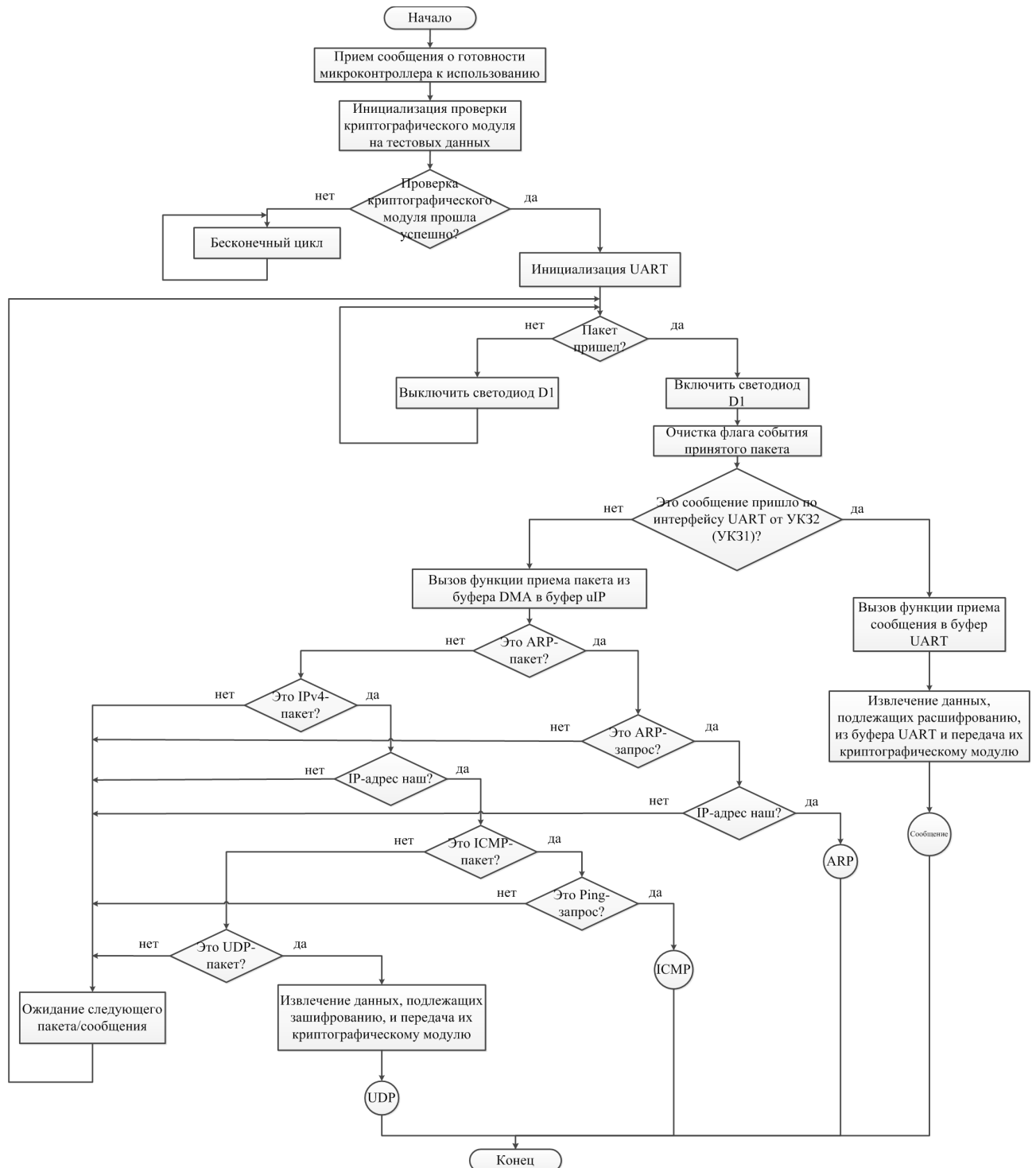


Рисунок В.1

Приложение Г

(обязательное)

Блок-схема алгоритма функционирования модуля передачи данных

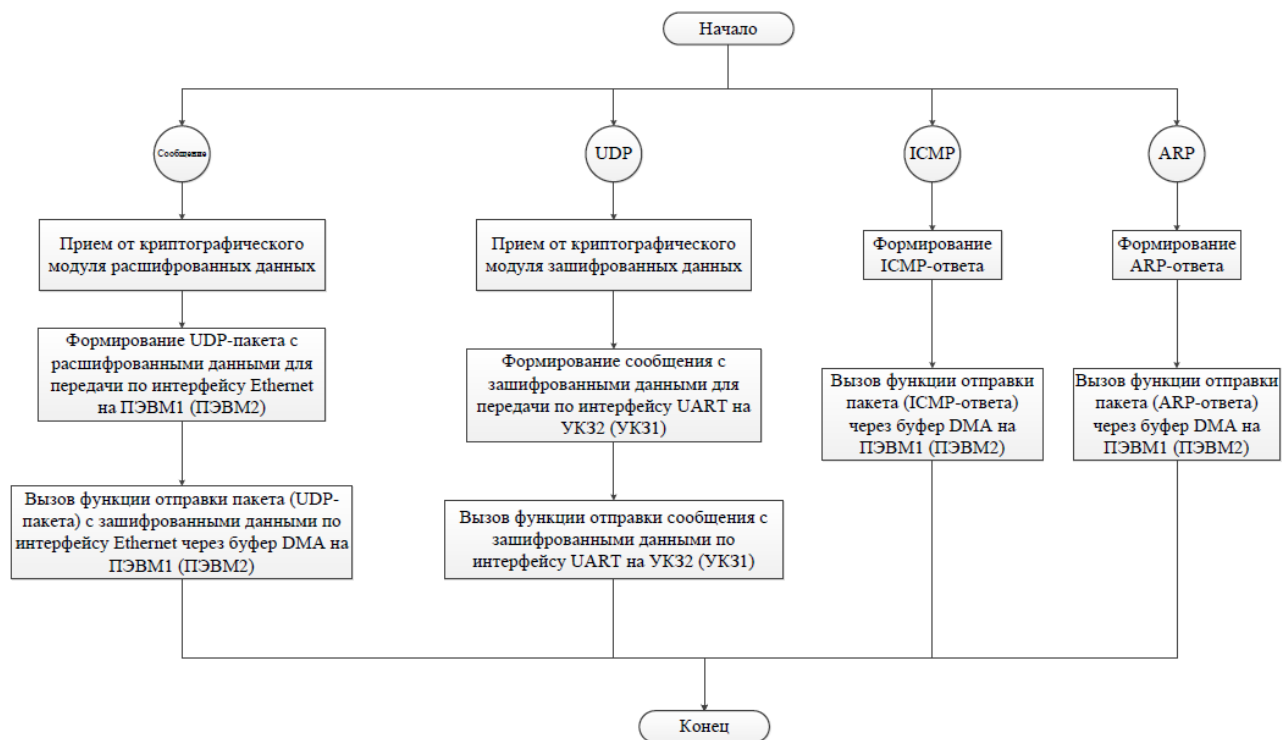


Рисунок Г.1

Приложение Д

(обязательное)

Листинг разработанного программного обеспечения

```
//cipher_gost.c

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

//таблица подстановок
static unsigned char const k1[16] = {12, 4, 6, 2, 10, 5, 11, 9,
14, 8, 13, 7, 0, 3, 15, 1};
static unsigned char const k2[16] = {6, 8, 2, 3, 9, 10, 5, 12, 1,
14, 4, 7, 11, 13, 0, 15};
static unsigned char const k3[16] = {11, 3, 5, 8, 2, 15, 10, 13,
14, 1, 7, 4, 12, 9, 6, 0};
static unsigned char const k4[16] = {12, 8, 2, 1, 13, 4, 15, 6, 7,
0, 10, 5, 3, 14, 9, 11};
static unsigned char const k5[16] = {7, 15, 5, 10, 8, 1, 6, 13, 0,
9, 3, 14, 11, 4, 2, 12};
static unsigned char const k6[16] = {5, 13, 15, 6, 9, 2, 12, 10,
11, 7, 8, 1, 4, 3, 14, 0};
static unsigned char const k7[16] = {8, 14, 2, 5, 6, 9, 1, 12, 15,
4, 11, 0, 13, 10, 3, 7};
static unsigned char const k8[16] = {1, 7, 14, 13, 0, 5, 8, 3, 4,
15, 10, 6, 9, 12, 11, 2};

static unsigned char k87[256];
static unsigned char k65[256];
static unsigned char k43[256];
static unsigned char k21[256];

//линейное преобразование
void kboxinit(void)
{
    int i;
    for (i = 0; i < 256; i++) {
        k87[i] = k8[i >> 4] << 4 | k7[i & 15];
        k65[i] = k6[i >> 4] << 4 | k5[i & 15];
        k43[i] = k4[i >> 4] << 4 | k3[i & 15];
        k21[i] = k2[i >> 4] << 4 | k1[i & 15];
    }
}

//преобразование блока подстановок
static uint32_t f(uint32_t x)
{

```

```

//блок подстановок
x = k87[x>>24 & 255] << 24 | k65[x>>16 & 255] << 16 |
    k43[x>> 8 & 255] << 8 | k21[x & 255];

//циклический сдвиг на 11 в сторону старших разрядов
return x<<11 | x>>(32-11);
}

//сложение с раундовым ключом
void gostcrypt(uint32_t const in[2], uint32_t out[2], uint32_t
const key[8])
{
    uint8_t i;
    register uint32_t n1, n2;
    n1 = in[0];
    n2 = in[1];
    //каждое вычисление n1 и n2 - один цикл
    for (i=0; i<23; i+=2)
    {
        n2 ^= f(n1+key[i % 8]);
        n1 ^= f(n2+key[(i+1)%8]);
    }
    for (i=24; i<31; i+=2)
    {
        n2 ^= f(n1+key[31-i]);
        n1 ^= f(n2+key[31-(i+1)]);
    }
    out[0] = n2;
    out[1] = n1;
}

void gostdecrypt(uint32_t const in[2], uint32_t out[2], uint32_t
const key[8])
{
    uint8_t i;
    register uint32_t n1, n2;
    n1 = in[0];
    n2 = in[1];

    for (i=0; i<7; i+=2)
    {
        n2 ^= f(n1+key[i]);
        n1 ^= f(n2+key[i+1]);
    }
    for (i=8; i<31; i+=2)
    {
        n2 ^= f(n1+key[(31-i)%8]);
        n1 ^= f(n2+key[(31-(i+1))%8]);
    }
    out[0] = n2;
    out[1] = n1;
}

```

```

}

//функция наложения гаммы
void gostcnt(uint8_t const *in, uint8_t *out, int len, uint32_t
sp[2], uint32_t const key[8])
{
    uint32_t gamma[2];

    while (len--)
    {
        gostcrypt(sp, gamma, key);

        *out++ = *in++ ^ (gamma[0]&0xff);
        *out++ = *in++ ^ ((gamma[0]>>8)&0xff);
        *out++ = *in++ ^ ((gamma[0]>>16)&0xff);
        *out++ = *in++ ^ ((gamma[0]>>24)&0xff);

        *out++ = *in++ ^ (gamma[1]&0xff);
        *out++ = *in++ ^ ((gamma[1]>>8)&0xff);
        *out++ = *in++ ^ ((gamma[1]>>16)&0xff);
        *out++ = *in++ ^ ((gamma[1]>>24)&0xff);

        //преобразование синхропосылки
        sp[0]++;
        if (sp[0] == 0)
        {
            sp[1]++;
        }
    }
}

//Ethernet.c

//
// Объявляем библиотеки
//
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>
#include "driverlib/debug.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib/emacs.h"

```

```

#include "inc/hw_emac.h"
#include "inc/hw_ints.h"
#include <string.h>

//
// Объявляем IP-адреса устройства
//
#define DEFAULT_IP0          10

#define DEFAULT_IP1          0

#define DEFAULT_IP2          5

#define DEFAULT_IP3          10

//
// Объявляем маски для IP-адреса
//
#define DEFAULT_NETMASK0      255

#define DEFAULT_NETMASK1      255

#define DEFAULT_NETMASK2      255

#define DEFAULT_NETMASK3      0

//
// Объявляем MAC-адреса устройства
//
#define DEFAULT_MACADDR0      0x00

#define DEFAULT_MACADDR1      0x1a

#define DEFAULT_MACADDR2      0xb6

#define DEFAULT_MACADDR3      0x02

#define DEFAULT_MACADDR4      0xa2

#define DEFAULT_MACADDR5      0x07

// Объявляем системную тактовую частоту в Гц
uint32_t g_ui32SysClock;

//*****
//*****
//
// Ethernet DMA дескрипторы.
//
// Хотя uIP использует один буфер, для работы аппаратного
обеспечения MAC

```

```

// требуется минимум 3 дескриптора
//
//*****
*****

#define NUM_TX_DESCRIPTOR 3
#define NUM_RX_DESCRIPTOR 3
tEMACDMADescriptor g_psTxDescriptor[NUM_TX_DESCRIPTOR];
tEMACDMADescriptor g_psRxDescriptor[NUM_RX_DESCRIPTOR];

uint32_t g_ui32RxDescIndex;
uint32_t g_ui32TxDescIndex;

//*****
*****
//
// Буферы передачи и приема
//
//*****
*****

#define RX_BUFFER_SIZE 1536
#define TX_BUFFER_SIZE 1536
uint8_t g_pui8RxBuffer[3][RX_BUFFER_SIZE];
uint8_t g_pui8TxBuffer[TX_BUFFER_SIZE];

//*****
*****
//
// Набор флагов. Биты флагов определяются следующим образом:
//
//      0 -> Индикатор прерывания
//      1 -> Получен пакет RX
//      2 -> Передача пакета TX DMA ожидается
//      3 -> Передача пакета RX DMA ожидается
//
//*****
*****

#define FLAG_SYSTICK          0
#define FLAG_RXPKT           1
#define FLAG_TXPKT           2
#define FLAG_RXPKTPEND       3
static volatile uint32_t g_ui32Flags;

//
//инициализация данных криптографического модуля
//

//синхропосылка
uint32_t synh[2]={0x00000000, 0x12345678};

```

```

//таблица замен
extern void    kboxinit(void);

//функция шифрования
extern void gostcnt(uint8_t const *in, uint8_t *out, int len,
uint32_t iv[2], uint32_t const key[8]);

//тестовые открытые данные
uint8_t etalon_test_data[32]= {0x59, 0x0a, 0x13, 0x3c, 0x6b, 0xf0,
0xde, 0x92,
                                0x20, 0x9d, 0x18, 0xf8, 0x04, 0xc7,
0x54, 0xdb,
                                0x4c, 0x02, 0xa8, 0x67, 0x2e, 0xfb,
0x98, 0x4a,
                                0x41, 0x7e, 0xb5, 0x17, 0x9b, 0x40,
0x12, 0x89};

//эталонные зашифрованные данные

uint8_t etalon_shifr_data[32]= {0x3c, 0xb9, 0xb7, 0x97, 0x0c,
0x11, 0x98, 0x4e,
                                0x69, 0x5d, 0xe8, 0xd6, 0x93,
0x0d, 0x25, 0x3e,
                                0xef, 0xdb, 0xb2, 0x07, 0x88,
0x86, 0x6d, 0x13,
                                0x2d, 0xa1, 0x52, 0xab, 0x80,
0xb6, 0x8e, 0x56};

//тестовые зашифрованные данные
uint8_t shifr_data[32], test_data[32];

//тестовый ключ

uint32_t test_key[8] =
{
    0xffeeddcc,
    0xbbaa9988,
    0x77665544,
    0x33221100,
    0xf0f1f2f3,
    0xf4f5f6f7,
    0xf8f9fafb,
    0xfcfdfeff
};

//*****
//
// Объявление переменных

```

```

//
//*****

uint16_t error = 0;

uint32_t rcv_udp_cnt = 0;

uint8_t link = 0x0;

uint16_t FrameLen[5] = {0,0,0,0,0};

uint8_t FrameMas[5][RX_BUFFER_SIZE];

uint8_t RcvPointer = 0;

uint8_t ReadPointer = 0;

uint16_t len = 0;

uint8_t RcvCnt = 0;

uint16_t uip_len;

uint8_t g_pui8UIPBuffer[RX_BUFFER_SIZE + 2];
uint8_t *uip_buf = g_pui8UIPBuffer;

//*****
*****
//
// Процедура ошибки, вызываемая, если в библиотеке драйверов
обнаружена ошибка
//
//*****
*****

#ifdef DEBUG
void __error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*****
*****
//
// Конфигурирование UART и его выводов
//
//*****
*****

void ConfigureUART(void)
{

```

```

//
// Включение периферийного устройства GPIO, используемое UART
//
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

//
// Включение UART0
//
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

//
// Настройка GPIO выводов для режима UART
//
ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

//
// Инициализация UART для консоли I/O.
//
UARTStdioConfig(0, 15000000, g_ui32SysClock); //115200
}

//*****
//*****
//
// The interrupt handler for the Ethernet interrupt.
// Обработчик прерываний для прерываний Ethernet???
//
//*****
//*****

void EthernetIntHandler(void)
{
    uint32_t ui32Temp;

    //
    // Чтение и очистка прерывания
    //
    ui32Temp = ROM_EMACIntStatus(EMAC0_BASE, true);
    ROM_EMACIntClear(EMAC0_BASE, ui32Temp);

    //
    // Проверка не произошло ли прерывание RX
    //
    if(ui32Temp & EMAC_INT_RECEIVE)
    {
        //
        // Установка флага в состояние "пакет получен"
        //
        HWREGBITW(&g_ui32Flags, FLAG_RXPKT) = 1;
    }
}

```

```

    }

    //
    // DMA закончил передачу пакета в передатчик?
    //
    if(ui32Temp & EMAC_INT_TRANSMIT)
    {
        //
        // Установка флага в состояние "пакет отправлен"
        //
        HWREGBITW(&g_ui32Flags, FLAG_TXPKT) = 0;
    }
}

//*****
//
// Передача пакета из предоставленного буфера
//
//*****

static int32_t PacketTransmit(uint32_t ui32Base, uint8_t *pui8Buf,
int32_t i32BufLen)
{
    int_fast32_t i32Loop;

    //
    // Установка флага в состояние "пакет отправляется"
    //
    HWREGBITW(&g_ui32Flags, FLAG_TXPKT) = 1;

    //
    // Ожидание, пока предыдущий пакет будет передан
    //
    while(g_psTxDescriptor[g_ui32TxDescIndex].ui32CtrlStatus &
        DES0_TX_CTRL_OWN)
    {
        //
        // Время ожидания
        //
    }

    //
    // Проверка того, что мы не будем переполнять буфер
    // передачи. Это
    // необязательно, поскольку буфер uIP меньше, чем наш буфер
    // передачи DMA,
    // но на всякий случай...
    //
    if(i32BufLen > TX_BUFFER_SIZE)

```

```

{
    i32BufLen = TX_BUFFER_SIZE;
}

//
// Копирование пакетных данных в буфер передачи
//
for(i32Loop = 0; i32Loop < i32BufLen; i32Loop++)
{
    g_pui8TxBuffer[i32Loop] = pui8Buf[i32Loop];
}

//
// Переход к следующему дескриптору
//
g_ui32TxDescIndex++;
if(g_ui32TxDescIndex == NUM_TX_DESCRIPTOR)
{
    g_ui32TxDescIndex = 0;
}

//
// Заполнение поля "размер пакета" и сообщение передатчику о
// начале работы
//
    g_psTxDescriptor[g_ui32TxDescIndex].ui32Count =
(uint32_t)i32BufLen;
    g_psTxDescriptor[g_ui32TxDescIndex].ui32CtrlStatus =
        (DES0_TX_CTRL_LAST_SEG | DES0_TX_CTRL_FIRST_SEG |
        DES0_TX_CTRL_INTERRUPT | DES0_TX_CTRL_IP_ALL_CKHSUMS |
        DES0_TX_CTRL_CHAINED | DES0_TX_CTRL_OWN);

//
// Tell the DMA to reacquire the descriptor now that
// we've filled it in.
// Говорим DMA ???
//
ROM_EMATxDMAPollDemand(EMAC0_BASE);

//
// Возвращаем количество отправленных байт
//
return(i32BufLen);
}

//*****
//
// Чтение пакета из буфера приема DMA в пакет буфера uIP ???
//
//*****

```

```
static int32_t PacketReceive(uint32_t ui32Base, uint8_t *pui8Buf,
int32_t i32BufLen)
{
    int_fast32_t i32FrameLen, i32Loop;

    //
    // Проверка аргументов
    //
    ASSERT(ui32Base == EMAC0_BASE);
    ASSERT(pui8Buf != 0);
    ASSERT(i32BufLen > 0);

    //
    // По умолчанию мы предполагаем, что у нас плохой кадр
    //
    i32FrameLen = 0;

    //
    // Убеждаемся, что у нас есть дескриптор приема
    //
    if(!(g_psRxDescriptor[g_ui32RxDescIndex].ui32CtrlStatus &
DES0_RX_CTRL_OWN))
    {
        //
        // У нас есть дескриптор приема, поэтому
        // проверяем содержит ли он
        // действительный кадр. Находим ошибку
        // дескриптора, указав, что входящий
        // пакет был усечен или, это последний кадр в
        // пакете -> бит ошибки приема
        //
        if(!(g_psRxDescriptor[g_ui32RxDescIndex].ui32CtrlStatus &
DES0_RX_STAT_ERR))
        {
            //
            // У нас есть действительный кадр для копирования
            // содержимого в
            // предоставленный буфер. Сначала
            // проверяем установлен ли флаг
            // "последний дескриптор". Мы
            // оцениваем буфер приема так, чтобы
            // он всегда мог иметь действительный
            // кадр, поэтому этот флаг никогда
            // не должен быть ПУСТЫМ на этом
            // этапе, но...
            //
            // We have a valid frame so copy the
            // content to the supplied
            // buffer. First check that the "last descriptor" flag
```

```

is set. We
// sized the receive buffer such that it can always
hold a valid
// frame so this flag should never be CLEAR at this
point but...
//
if(g_psRxDescriptor[g_ui32RxDescIndex].ui32CtrlStatus
&
    DES0_RX_STAT_LAST_DESC)
{
    i32FrameLen =
        ((g_psRxDescriptor[g_ui32RxDescIndex].ui32Ctrl
Status &
        DES0_RX_STAT_FRAME_LENGTH_M) >>
        DES0_RX_STAT_FRAME_LENGTH_S);

    //
    // Необязательная проверка, так как мы оцениваем
буфер uIP таким
    // образом, чтобы он был
того же размера, что и буфер приема DMA,
    // но на всякий случай...
    //
    if(i32FrameLen > i32BufLen)
    {
        i32FrameLen = i32BufLen;
    }

    FrameLen[RcvPointer] =
i32FrameLen; //сохраним длину принятого пакета

    for(i32Loop = 0; i32Loop <
i32FrameLen; i32Loop++) //копируем в буфер
    {
        FrameMas[RcvPointer]
[i32Loop] = g_pui8RxBuffer[g_ui32RxDescIndex][i32Loop];
    }
    RcvCnt++;
    RcvPointer++; //указатель
на запись

    if (RcvPointer>4)
        RcvPointer = 0;

}

//
// Переход к следующему дескриптору в цепочке
//

```

```

        g_ui32RxDescIndex++;
        if(g_ui32RxDescIndex == NUM_RX_DESCRIPTOR)
        {
            g_ui32RxDescIndex = 0;
        }

        //
        // Отмечаем слудующий дескриптор в цепочке доступным для
записи получателем
        //
        g_psRxDescriptor[g_ui32RxDescIndex].ui32CtrlStatus =
DES0_RX_CTRL_OWN;
    }

    //
    // Возвращаем длину кадра
    //
    return(i32FrameLen);
}

//*****
//*****
//
// Инициализация дескрипторов передачи и приема DMA. Требуется как
минимум 3
// дескриптора в каждой цепочке. Это излишне, так как uIP
использует единый
// буфер передачи и приема, поэтому мы помещаем каждый дескриптор
в тот же
// буфер и следим за тем, чтобы передавался только один дескриптор
DMA
// за один раз
//
//*****
//*****

void InitDescriptors(uint32_t ui32Base)
{
    uint32_t ui32Loop;

    //
    // Инициализируем каждый из дескрипторов передачи. Бит OWN
остается пустым,
    // потому что пока не были настроены никакие передачи
    //
    for(ui32Loop = 0; ui32Loop < NUM_TX_DESCRIPTOR; ui32Loop++)
    {
        g_psTxDescriptor[ui32Loop].ui32Count =
            (DES1_TX_CTRL_SADDR_INSERT |
            (TX_BUFFER_SIZE << DES1_TX_CTRL_BUFF1_SIZE_S));
    }
}

```

```

g_psTxDescriptor[ui32Loop].pvBuffer1 = g_pui8TxBuffer;
g_psTxDescriptor[ui32Loop].DES3.pLink =
    (ui32Loop == (NUM_TX_DESCRIPTOR - 1)) ?
    g_psTxDescriptor : &g_psTxDescriptor[ui32Loop + 1];
g_psTxDescriptor[ui32Loop].ui32CtrlStatus =
    (DES0_TX_CTRL_LAST_SEG | DES0_TX_CTRL_FIRST_SEG |
    DES0_TX_CTRL_INTERRUPT | DES0_TX_CTRL_CHAINED |
    DES0_TX_CTRL_IP_ALL_CKHSUMS);
}

//
// Инициализируем каждый из дескрипторов приема. Бит OWN
очищаем здесь,
// чтобы убедиться, что приемник не начинает ничего
немедленно записывать
//
for(ui32Loop = 0; ui32Loop < NUM_RX_DESCRIPTOR; ui32Loop++)
{
    g_psRxDescriptor[ui32Loop].ui32CtrlStatus = 0;
    g_psRxDescriptor[ui32Loop].ui32Count =
        (DES1_RX_CTRL_CHAINED |
        (RX_BUFFER_SIZE << DES1_RX_CTRL_BUFF1_SIZE_S));
    g_psRxDescriptor[ui32Loop].pvBuffer1 =
g_pui8RxBuffer[ui32Loop];
    g_psRxDescriptor[ui32Loop].DES3.pLink =
        (ui32Loop == (NUM_RX_DESCRIPTOR - 1)) ?
        g_psRxDescriptor : &g_psRxDescriptor[ui32Loop + 1];
}

//
// Устанавливаем указатели дескриптора на аппаратное
обеспечение ???
//
ROM_EMACRxDMADescriptorListSet(ui32Base, g_psRxDescriptor);
ROM_EMACTxDMADescriptorListSet(ui32Base, g_psTxDescriptor);

//
// Начнем с начала обеих цепочек дескрипторов. Фактически мы
устанавливаем
// индекс дескриптора передачи в последний дескриптор в
цепочке, так как он
// будет увеличен до использования, и это означает, что
первая передача будет
// выполнена с использованием правильного дескриптора
//
g_ui32RxDescIndex = 0;
g_ui32TxDescIndex = NUM_TX_DESCRIPTOR - 1;
}

//контрольная сумма crc16 ???

```

```

/*
uint16_t checksum(uint16_t sum, uint8_t *data, uint16_t len)
{
    uint16_t t;
    const uint8_t *dataptr;
    const uint8_t *last_byte;

    dataptr = data;
    last_byte = data + len - 1;

    while(dataptr < last_byte)
    {
        // At least two more bytes.
        t = (dataptr[0] << 8) + dataptr[1];
        sum += t;
        if(sum < t)
        {
            sum++;
        }
        dataptr += 2;
    }

    if(dataptr == last_byte)
    {
        t = (dataptr[0] << 8) + 0;
        sum += t;
        if(sum < t)
        {
            sum++;
        }
    }

    // Return sum in host byte order.
    return sum;
}
*/

```

```

int main(void)
{
    struct eth_addr {uint8_t addr[6];};
    static struct eth_addr sTempAddr;

    struct ip_addr {uint8_t addr[4];};
    struct ip_addr sTempIPAddr;

    uint32_t ui32PHYConfig, ui32SysClock;
    uint8_t link = 0x0;
    uint16_t /*tmp*/ i;

    //

```

```

    // Установка тактовой частоты PLL в 120 МГц.
    //
    g_ui32SysClock = MAP_SysCtlClockFreqSet((      SYSCTL_XTAL_25MHZ
| //опорная частота

                                                                    SYSCTL_OSC_MAIN   |
//внутренний генератор

                                                                    SYSCTL_USE_PLL
| //умножитель частоты

    SYSCTL_CFG_VCO_480), 120000000);

//
// Сброс (установка в 0) выводов Eth и USB
//
PinoutSet(false, false);

//
// Включение вывода GPIO для светодиода D1 (PN1)
//
ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);

//
// Конфигурируем Ethernet на работу с внутренним модулем
PHY в режиме 100base-TX Full Duplex
//
    ui32PHYConfig      =      (EMAC_PHY_TYPE_INTERNAL      |
EMAC_PHY_INT_MDIX_EN |
    EMAC_PHY_AN_100B_T_FULL_DUPLEX);

//
// Установка MAC-адреса
//
sTempAddr.addr[0] = DEFAULT_MACADDR0;
sTempAddr.addr[1] = DEFAULT_MACADDR1;
sTempAddr.addr[2] = DEFAULT_MACADDR2;
sTempAddr.addr[3] = DEFAULT_MACADDR3;
sTempAddr.addr[4] = DEFAULT_MACADDR4;
sTempAddr.addr[5] = DEFAULT_MACADDR5;

//
// Установка IP-адреса
//
sTempIPAddr.addr[0] = DEFAULT_IP0;
sTempIPAddr.addr[1] = DEFAULT_IP1;
sTempIPAddr.addr[2] = DEFAULT_IP2;
sTempIPAddr.addr[3] = DEFAULT_IP3;

//

```

```

// Включение и перезагрузка модулей Ethernet
//
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_EMAC0);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_EPHY0);
ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_EMAC0);
ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_EPHY0);

//
// Ожидаем пока MAC будет готов
//
while(!ROM_SysCtlPeripheralReady(SYSCTL_PERIPH_EMAC0))
{
    //
    //Ожидание
    //
}

//
// Конфигурируем внутренний PHY для его использования
//
MAP_EMACPHYConfigSet(EMAC0_BASE, ui32PHYConfig);

//
// Перезагрузка MAC
//
ROM_EMACReset(EMAC0_BASE);

//
// Инициализируем MAC и устанавливаем режим DMA
//
ROM_EMACInit(EMAC0_BASE, ui32SysClock,
EMAC_BCONFIG_PRIORITY_FIXED, 4, 4,
0);
EMAC_BCONFIG_MIXED_BURST |

//
// Устанавливаем параметры конфигурации MAC
//
ROM_EMACConfigSet(EMAC0_BASE,
EMAC_CONFIG_CHECKSUM_OFFLOAD |
EMAC_CONFIG_FULL_DUPLEX |
EMAC_CONFIG_7BYTE_PREAMBLE |
EMAC_CONFIG_IF_GAP_96BITS |
EMAC_CONFIG_USE_MACADDR0 |
EMAC_CONFIG_SA_FROM_DESCRIPTOR |
EMAC_CONFIG_BO_LIMIT_1024),
(EMAC_MODE_RX_STORE_FORWARD |
EMAC_MODE_TX_STORE_FORWARD |
EMAC_MODE_TX_THRESHOLD_64_BYTES |
EMAC_MODE_RX_THRESHOLD_64_BYTES), 0);

```

```

//
// Инициализируем дескрипторы DMA Ethernet
//
InitDescriptors(EMAC0_BASE);

//
// Программируем аппаратное обеспечение с его MAC-адресом (для
// фильтрации) ???
//
ROM_EMACAddrSet(EMAC0_BASE, 0, (uint8_t *)&sTempAddr);

//
// Устанавливаем параметры фильтрации MAC. Мы получаем все
// широковещательные
// и многоадресные пакеты вместе с их адресами, специально
// адресованными нам
//
ROM_EMACFrameFilterSet(EMAC0_BASE, (EMAC_FRMFILTER_SADDR |
AST |
EMAC_FRMFILTER_PASS_MULTIC
L));

//
// Очищаем все ожидающие прерывания
//
ROM_EMACIntClear(EMAC0_BASE, EMACIntStatus(EMAC0_BASE,
false));

//
// Включаем передатчик и приемник Ethernet MAC
//
ROM_EMACTxEnable(EMAC0_BASE);
ROM_EMACRxEnable(EMAC0_BASE);

//
// Включаем прерывание Ethernet
//
ROM_IntEnable(INT_EMAC0);

//
// Включаем источник прерываний для Ethernet RX Packet
//
ROM_EMACIntEnable(EMAC0_BASE, EMAC_INT_RECEIVE);

//
// Отмечаем первый дескриптор приема как доступный для DMA,
// чтобы начать
// процесс приема
//
g_psRxDescriptor[g_ui32RxDescIndex].ui32CtrlStatus |=

```

```

DES0_RX_CTRL_OWN;

//
// Инициализация шифратора и проверка на тестовых данных
//
kboxinit();
gostcnt(etalon_test_data, shifr_data, 4, synh, test_key);

for (int i=0;i<31;i++)
    if (etalon_shifr_data[i] != shifr_data[i])
    {
        //если выработанная и эталонная гаммы не
совпали
        while (1);
    }
synh[0] = 0;
gostcnt(shifr_data, test_data, 4, synh, test_key);
//
// Сравниваем зашифрованные данные на тестовых данных с
эталонными
// зашифрованными данными. Если массивы совпадают, то
идем дальше,
// иначе завершение программы
//

for (int i=0;i<31;i++)
    if (etalon_test_data[i] != test_data[i])
    {
        //если выработанная и эталонная гаммы не
совпали
        while (1);
    }

//
// Инициализируем UART
//
ConfigureUART();

//
// Основной цикл программы
//
while(1)
{
    //
    if ((ROM_EMACPHYRead(EMAC0_BASE, 0, EPHY_BMSR)
& EPHY_BMSR_LINKSTAT) == 0)
    {
        //
        link = 0x0;
    }
    else

```

```

        {
            //
            link = 0x55;
        };

        if (link == 0)
        {
            // Выключить D1
            LEDWrite(CLP_D1, 0);
        }
        else
        {
            // Включить D1
            LEDWrite(CLP_D1, 1);
        }

        //
        // Ожидаем появления события. Это может быть событие
System Tick или
        // событие RX Packet
        //
        if(g_ui32Flags)
        {

            //
            // Проверка пакета RX и его чтение
            //
            if(HWREGBITW(&g_ui32Flags, FLAG_RXPKT))
            {
                //
                // Очистка флага события принятого пакета RX
                //
                HWREGBITW(&g_ui32Flags, FLAG_RXPKT) = 0;

                //
                // Получаем пакет и устанавливаем uip_len для
использования стека uIP
                //
                uip_len = (unsigned short)PacketReceive(EMAC0_BASE,
uip_buf, sizeof(g_pui8UIPBuffer));

                if (uip_len)
                {

                    if (FrameMas[ReadPointer][12] ==
0x08 && FrameMas[ReadPointer][13] == 0x06) //ARP?
                    {
                        //ARP
                        if (FrameMas[ReadPointer]
[20] == 0x00 && FrameMas[ReadPointer][21] == 0x01) //запрос?

```

```

(opcode=1)

{
    //получили Запрос

    memcmp(&FrameMas[ReadPointer][38], &sTempIPAddr, 4); //зачем
    здесь???

    if
    (memcmp(&FrameMas[ReadPointer][38], &sTempIPAddr, 4) == 0) //наш
    IP?
    {
        //наш IP
        //формируем ARP-
        ответ

        memcpy(&uip_buf[0], &FrameMas[ReadPointer][6], 6); //MAC
        отправителя -> MAC получателя

        memcpy(&uip_buf[6], &sTempAddr.addr, 6); //свой MAC -> MAC
        отправителя

        memcpy(&uip_buf[32], &FrameMas[ReadPointer][6], 6); //MAC
        отправителя -> MAC получателя

        uip_buf[12] =
        0x8; //ARP
        uip_buf[13] =
        0x6;

        uip_buf[14] =
        0; //HType = 1 (eth)
        uip_buf[15] = 1;

        uip_buf[16] =
        8; //PType = 8 (ip)
        uip_buf[17] = 0;

        uip_buf[18] =

        6; //HSize

        uip_buf[19] =

        4; //PSize

        uip_buf[20] =
        0; //opcode = 2 (ответ)
        uip_buf[21] = 2;

        memcpy(&uip_buf[22], &sTempAddr.addr, 6);

```

```

        memcpy(&uip_buf[28], &sTempIPAddr.addr, 4); //свой IP

        memcpy(&uip_buf[38], &FrameMas[ReadPointer][28], 4);
//получателя IP

        PacketTransmit(EMAC0_BASE, uip_buf, uip_len);
                                uip_len = 0;
                                }

                                }
                                else
                                {
                                        //получили ответ
                                }
                                }
                                else
                                {
                                        //не ARP
                                        if (FrameMas[ReadPointer]
[12] == 0x08 && FrameMas[ReadPointer][13] == 0x00) //IP?
                                        {
                                                //IP
                                                if
(memcmp(&FrameMas[ReadPointer][0x1E], &sTempIPAddr, 4) == 0 ) //наш
IP?
                                                {

                                                        if
(FrameMas[ReadPointer][23]==0x01) //ICMP?
                                                        {

                                                                //ICMP
                                                                if
(FrameMas[ReadPointer][34]==0x08) //ping?
                                                                {
                                                                        //
получили PING запрос
                                                                        //
мас

                                memcpy(&uip_buf[0], &FrameMas[ReadPointer][6], 6); //MAC
отправителя -> MAC получателя

                                memcpy(&uip_buf[6], &sTempAddr.addr, 6); //свой MAC -> MAC
отправителя

                                                                //
ip

```

```

for (i=12;i<26;i++)
{

uip_buf[i] = FrameMas[ReadPointer][i];
}

memcpy(&uip_buf[26],&sTempIPAddr,4); //my IP
memcpy(&uip_buf[30],&FrameMas[ReadPointer][26],4); //dest IP

uip_buf[24] = 0; //crc ip
uip_buf[25] = 0;
//
tmp = ~chksum(0, &uip_buf[14], 20);
//
uip_buf[25] = tmp&0xff;
//
uip_buf[24] = (tmp>>8)&0xff;
//
icmp

uip_buf[34] = 0; //ответ
uip_buf[35] = 0;

for (i=38;i<74;i++)
{

uip_buf[i] = FrameMas[ReadPointer][i];
}

uip_buf[36] = 0; //crc
uip_buf[37] = 0;
//
tmp = ~chksum(0, &uip_buf[34], 40); //crc icmp
//
uip_buf[37] = tmp&0xff;
//
uip_buf[36] = (tmp>>8)&0xff;

```

```

PacketTransmit(EMAC0_BASE, uip_buf, uip_len-4);

uip_len = 0;

}
else
{
    //не
ICMP
    if
    (FrameMas[ReadPointer][23]==0x11) //UDP?
    {
        //
UDP
    }

    //
вызов формирователя гаммы по размеру поля данных пакета

    len = (uip_len-46)/8;

    gostcnt(&FrameMas[ReadPointer][42],      &FrameMas[ReadPointer]
[42], len, synh, test_key);

    //
формирование адресов MAC и IP

    memcpy(&uip_buf[0], &FrameMas[ReadPointer][0], uip_len);
    //копируем весь буфер

    memcpy(&uip_buf[0], &FrameMas[ReadPointer][6], 6); //меняем
местами MAC

    memcpy(&uip_buf[6], &FrameMas[ReadPointer][0], 6);

    memcpy(&uip_buf[26], &FrameMas[ReadPointer][30], 4); //меняем
местами IP

    memcpy(&uip_buf[30], &FrameMas[ReadPointer][26], 4);

    PacketTransmit(EMAC0_BASE, uip_buf, uip_len-4); //передача

    uip_len = 0;

}

```

