МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2024 г.

Разработка программы для реализации обмена данными в промышленной сети Ethernet с гарантированным временем доставки пакетов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ–090301.2024.609 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ Д.В. Топольский
«___»_____2024 г.

Автор работы,
студент группы КЭ-405
_____ К.Н. Ширяев
«___»_____2024 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2024 г.

Челябинск-2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2024 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Ширяеву Климу Николаевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Разработка программы для реализации обмена данными в промышленной сети Ethernet с гарантированным временем доставки пакетов» утверждена приказом по университету от «22» апреля 2024 г. № 764-13/12

Срок сдачи студентом законченной работы: 01 июня 2024 г.

Исходные данные к работе:

- отладочная плата WaveShare XCore 407I;
- среда разработки Keil uVision;
- язык программирования C.

Перечень подлежащих разработке вопросов:

Выпускная квалификационная работа должна содержать разработку следующих вопросов:

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.
2. Разработка алгоритмов обмена данными с электронным блоком управления в промышленной сети Ethernet с гарантированным временем доставки пакетов.
3. Разработка программного кода для обмена данными с электронным блоком управления в промышленной сети Ethernet.
4. Тестирование и отладка программного кода (на микроконтроллере, на персональном компьютере).

Дата выдачи задания: 01 декабря 2023 г.

Руководитель работы _____/ *Д.В. Топольский /*

Студент _____/ *К.Н. Ширяев /*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2024	
Разработка алгоритмов обмена данными с электронным блоком управления в промышленной сети Ethernet с гарантированным временем доставки пакетов	01.04.2024	
Разработка программного кода для обмена данными с электронным блоком управления в промышленной сети Ethernet	01.05.2024	
Отладка и тестирование программного кода	15.05.2024	
Компоновка текста работы и сдача на нормоконтроль	24.05.2024	
Подготовка презентации и доклада	30.05.2024	

Руководитель работы _____/ *Д.В. Топольский* /

Студент _____/ *К.Н. Ширяев* /

АННОТАЦИЯ

Ширяев К.Н. Разработка программы для реализации обмена данными в промышленной сети Ethernet с гарантированным временем доставки пакетов – Челябинск: ЮУрГУ, ВШ ЭКН; 2024, 39 с., библиогр. список – 15 наим.

После анализа существующих аналогов протоколов промышленных сетей Ethernet, было принято решение выбрать протокол Modbus TCP для реализации обмена данными. Его преимущество заключается в распространенности и открытости.

На базе примера TCP Echo сервера из библиотеки LWIP разработана собственная программа для реализации обмена данными с помощью протокола Modbus TCP в режиме Slave на платформе микроконтроллера STM32. Реализованы функции чтения и записи регистров (0x03, 0x06).

Программа успешно прошла тестирование необходимых функций с помощью панели оператора Kinco GL070E. Получено гарантированное время доставки пакетов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Невозможно представить себе современную промышленность без контроллеров. Любая технологическая линия управляется специальными устройствами. В промышленности этими устройствами являются программируемыми логическими контроллерами (ПЛК).

ПЛК – это цифровая электронная система, предназначенная для применения в производственной среде, которая использует программируемую память для внутреннего хранения ориентированных на потребителя инструкций по реализации таких специальных функций, как логика, установление последовательности, согласование по времени, счет и арифметические действия для контроля посредством цифрового или аналогового ввода/вывода данных различных видов машин или процессов.

Существуют две концепции построения систем промышленной автоматизации: локальные и распределенные автоматизированные системы [1].

Локализованные автоматизированные системы управляются одним ПЛК, который выполняет функцию сбора данных и управления процессами в одном конкретном месте.

В случае с распределенными автоматизированными системами справедливо все то, что сказано про локализованные, за исключением того, что отдельные компоненты системы могут находиться на значительном расстоянии друг от друга.

Отдельные компоненты автоматизированных систем могут соединяться между собой с помощью промышленных сетей и полевых шин.

Первый промышленный протокол Modbus был создан компанией Modicon (Schneider Electric) в 1979 году [2]. Изначально этот стандарт использовал интерфейс RS-232, но позднее появилась возможность работать через RS-485.

С течением времени, чтобы обеспечить выполнение задач в заданный промежуток времени, в 1990-х годах была разработана технология Real-time Ethernet. Главным преимуществом Ethernet для сетей реального времени, безусловно, является скорость. Стандарты Fast (100 Мбит/с) и Gigabit Ethernet (1

Гбит/с) на порядок превосходят самую быструю полевую шину Profibus (12 Мбит/с) [3].

Однако, Ethernet стандарта IEEE 802.3 не может обеспечить функционирование сетей реального времени. Все дело в механизме доступа CSMA/CD (множественный доступ с контролем несущей частоты и обнаружением коллизий), функционирующем на канальном (втором) уровне модели OSI. Данный механизм позволяет начать конкретной станции передачу данных только когда она дождется окончания передачи данных остальными участниками. Причем интервал ожидания является величиной произвольной, что делает невозможным расчет длительности цикла передачи.

Для функционирования Ethernet в реальном времени, придумали ввести механизм, позволяющий избежать коллизии.

1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ ПО ТЕМАТИКЕ РАБОТЫ

1.1. Современные протоколы промышленных сетей

EtherCAT – один из быстрейших протоколов промышленной сети на данный момент, способный работать в режиме жесткого реального времени. Разработан компанией Beckhoff в 2003 году.

Данный протокол позволяет отправить одну команду для опроса и управления сразу несколькими подчиненными устройствами. Slave устройства тем временем на лету считывают нужные им данные или записывают данные для передачи, а затем пересылают фрейм следующему устройству. Такой механизм позволяет на высокой скорости управлять до 65 535 устройствами в одной сети, а также выбирать любую топологию сети. Также, EtherCAT совместим с любым стандартным Ethernet оборудованием, так как для передачи данных данный протокол использует стандартные фреймы Ethernet [4].

PROFINET – промышленная сеть, разработанная немецким концерном Siemens. Топология сети может быть любой.

Существуют два протокола для PROFINET [5]:

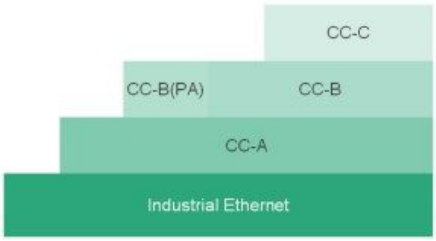
1. PROFINET IO.
2. PROFINET CBA.

Протокол PROFINET IO бывает трех классов [6]:

1. PROFINET NRT (без реального времени).
2. PROFINET RT (реальное время).
3. PROFINET IRT (изохронное реальное время).

Класс PROFINET IRT имеет два канала обмена: изохронный и асинхронный. Изохронный канал передает критичные ко времени данные. В асинхронном же канале передаются так называемые real-time данные, а также различная вспомогательная информация. Естественно, никакая информация не может прервать изохронный цикл.

Протокол PROFINET IO имеет несколько классов применения: CC-A, CC-B, CC-CC (рисунок 1). Эти классы дают возможность выбрать полевые устройства с необходимой функциональностью [7].



	A	B	C
Real-Time Data Exchange – cycle times down to 1ms	☑	☑	☑
Alarms and Diagnostics	☑	☑	☑
Network Topology Support	☑	☑	☑
SNMP Support		☑	☑
Rea-Time Data Exchange – cycle times down to 31.25µs			☑

Рисунок 1 – Классы соответствия PROFINET IO

Следующим протоколом промышленной сети, реализованный поверх стандарта Ethernet, является Powerlink, разработанный австрийской компанией B&R в 2001 году.

Данный протокол использует механизм смешанного опроса, который делит взаимодействие между устройствами на несколько фаз. В изохронной фазе, для которой настраивается требуемое время отклика, передаются критичные ко времени данные. В асинхронной фазе будут переданы остальные некритичные данные [8].

Кроме того, в протоколе применяется механизм планирования обмена. В сеть отправляется маркер, указывающий на то, какое устройство в текущий момент имеет право на передачу данных (только одно устройство может обмениваться данными в любой момент времени).

Этот протокол работает следующим образом: опрашивающий контроллер в изохронной фазе отправляет запросы последовательно каждому узлу, от которого требуется получение важных данных. Как было упомянуто ранее, временной цикл настраиваемый. Затем в асинхронной фазе обмена происходит опрос

некритических данных от всех узлов, которые отправляют ответ после получения разрешающего маркера доступа к передаче данных в сеть. Временные интервалы между синхронной и асинхронной фазами могут изменяться. Завершающим протоколом в обзоре является EtherNet/IP, разработанный компанией Rockwell Automation в 2000 году. Стоит отметить, что вторая часть в названии обозначает именно Industrial Protocol.

Топология сети EtherNet/IP может быть любой и состоять из кольца, дерева, шины, либо звезды [9].

EtherNet/IP способен передавать данные, требующие точного соблюдения временных рамок, между опрашивающим контроллером и устройствами. Некритичные данные передаются через TCP, а критичные — через UDP. Для синхронизации времени EtherNet/IP применяет протокол CIPsync [10].

EtherNet/IP имеет заранее подготовленные типовые конфигурационные файлы, что облегчает настройку этого протокола.

1.2. Операционные системы реального времени

Помимо промышленных сетей, существуют также операционные системы реального времени (ОСРВ). Как и промышленные сети, ОСРВ отличаются скоростью обработки внешних сигналов и реагировании в заданный промежуток времени.

Наиболее популярной на сегодняшний день является FreeRTOS. Из достоинств – мощный функционал, подробная документация, наличие различных библиотек [11]. К тому же, она бесплатная. Поддерживает значительное количество процессоров, к ним относятся: Atmel, STMicroelectronics, Intel и т.д.

Еще одна ОСРВ – KeilRTX. До недавнего времени была коммерческой, но сейчас бесплатная. Из недостатков – поддержка только архитектуры ARM, невысокая скорость в сравнении с конкурентами [12]. Из достоинств – бесплатная, наличие различных библиотек.

Из коммерческих ОСРВ можно выделить uc/os. Из преимуществ – значительное количество различных функций и библиотек, поддержка значительного количества устройств, с недавнего времени стала бесплатной. Из недостатков – данная операционная система сложна в использовании [13].

1.3. Системы реального времени на обычных операционных системах

Также, реализация систем жесткого реального времени возможна и на обычных операционных системах. Так, реализация на Windows возможна с помощью ПО RtaccWin, разработанного немецкой компанией Acontis.

Данная программа резервирует как минимум одно ядро у операционной системы, что позволяет работать приложениям реального времени с гарантированным временем цикла менее 1 миллисекунды. Кроме того, не используются стандартные драйверы Windows, вместо этого RtaccWin обеспечивает прямой доступ к оборудованию, такому как сетевые карты, что позволяет пользоваться Ethernet без сетевого стека Windows, а при параллельном использовании не возникает никаких побочных эффектов [14].

Конфигурация системы Windows для работы с приложениями реального времени осуществляется с помощью ПО RtaccWinConfig (см. рисунок 2).

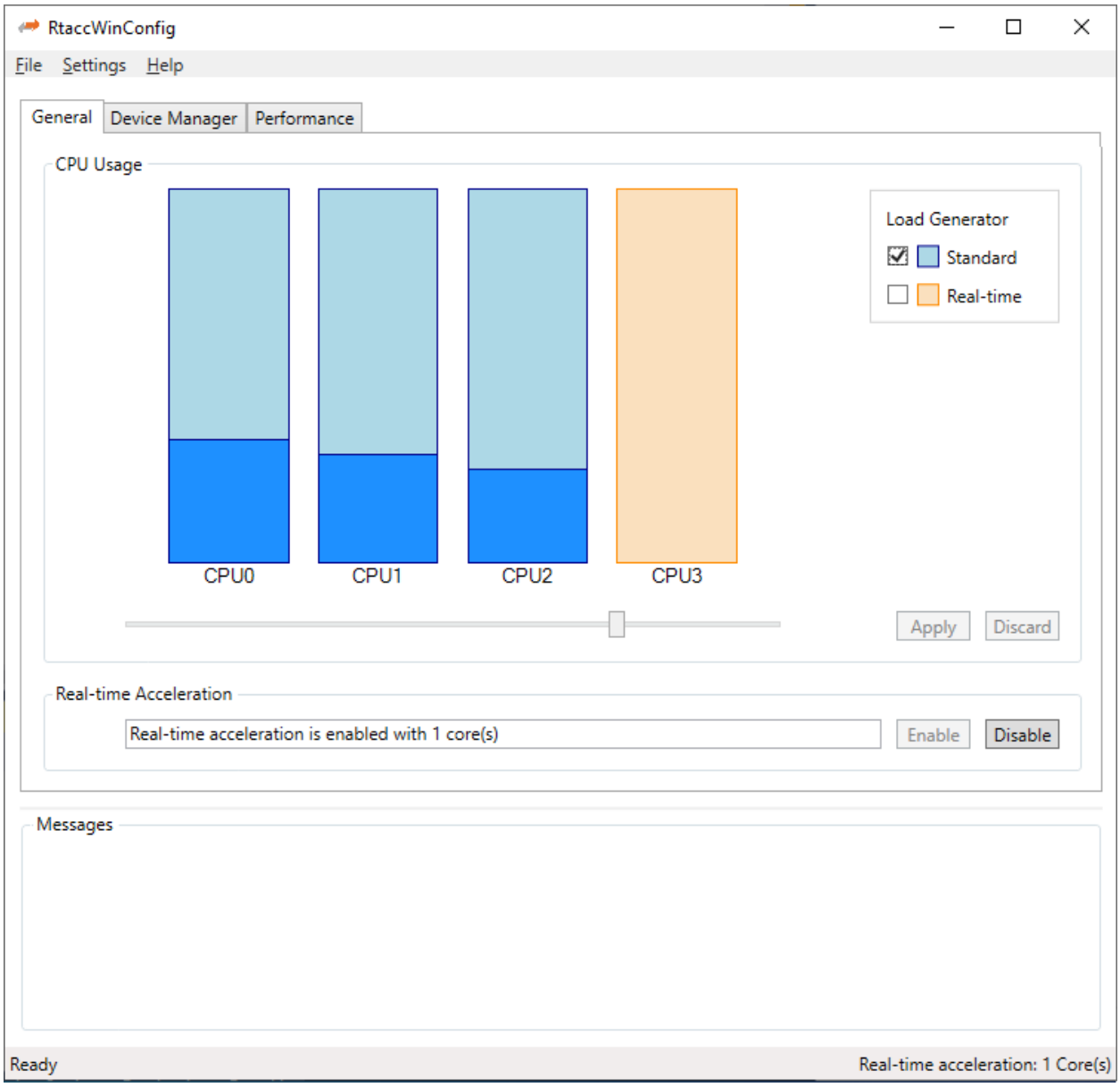


Рисунок 2 – Экранная форма конфигурации ядер процессора в ПО RtaccWinConfig

Помимо этого, как ранее было упомянуто, другой функцией программы конфигурации RtaccWinConfig является возможность получения быстрого и прямого доступа ко всем областям памяти выбранного оборудования компьютера (см. рисунок 3).

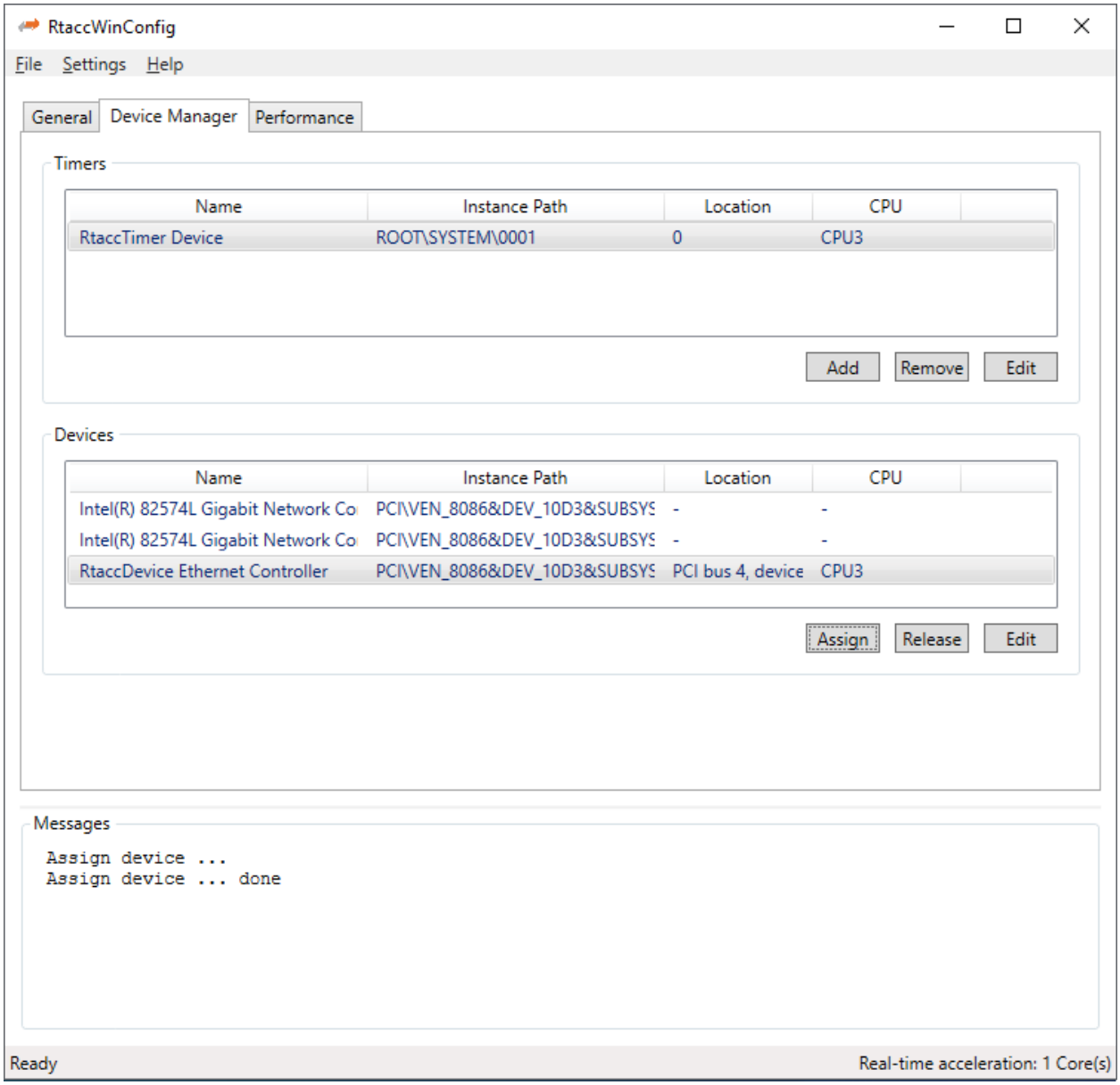


Рисунок 3 – Окно выбора необходимого оборудования компьютера для работы приложений реального времени с помощью программы RtaccWinConfig

Также, компания Beckhoff разработала программу TwinCAT, работающая по тому же принципу, что и RtaccWin. TwinCAT запускает на освободившихся ядрах сеть EtherCAT, что позволяет выступать персональному компьютеру в качестве master устройства [4].

Что касается операционной системы GNU/Linux, то существует патч реального времени PREEMPT_RT. Данный патч переводит обработчики прерываний в потоки ядра, которые подчиняются общим правилам планирования потоков системы [15].

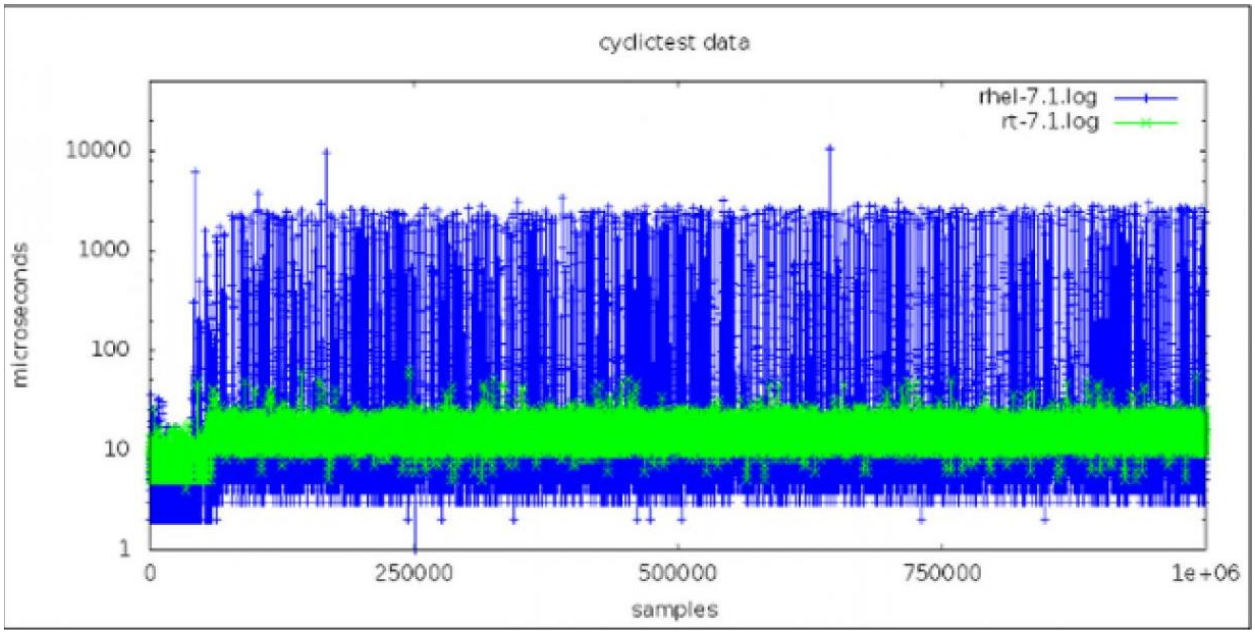


Рисунок 4 – Сравнение ядра с патчем реального времени со стандартным ядром

Представленный график показывает замер времени отклика из миллиона повторений для систем, использующих стандартное ядро линукс и ядро с патчем реального времени соответственно (рисунок 4). Синие точки – стандартное ядро, зеленые – ядро с патчем реального времени. Видно, что ядро с патчем обладает большей предсказуемостью, так как время отклика обладает меньшей дисперсностью.

1.4. Интерфейсы Modbus, их преимущества и недостатки

Были рассмотрены актуальные протоколы для реализации промышленных сетей. Среди них – PROFINET, EtherNet/IP, EtherCAT и другие.

В качестве протокола обмена данными в промышленной сети Ethernet для электронного блока управления был выбран Modbus TCP. Причиной тому является

его распространенность. Несомненно, Modbus является самым популярным протоколом в автоматизированных системах управления технологическими процессами. Связано это, как и было упомянуто в обзоре литературы, с тем, что Modbus является первым промышленным протоколом. К тому же, он является открытым, что делает его доступным для разработчиков собственных устройств.

Базируется Modbus на архитектуре ведущий – ведомый(master – slave). Имеет режимы ASCII и RTU. В режиме ASCII данные поступают закодированными из одноименной таблицы и передаются в HEX формате. Перед началом пакета ставится двоеточие, в конце же - символ возврата каретки "CR+LF". Пакет RTU же содержит в себе двоичные данные, а конец пакета определяется по интервалам времени, что требует от устройства дополнительные ресурсы в виде точного таймера.

Изначально стандарт Modbus работал только по интерфейсу RS-232. RS-232 – это проводной дуплексный интерфейс с максимальной скоростью 11,5 Кбайт в секунду. Соединяется разъемами типа D-sub. Обычно 9-ти контактный DB9, реже 25-ти контактный DB25 (рисунок 5).

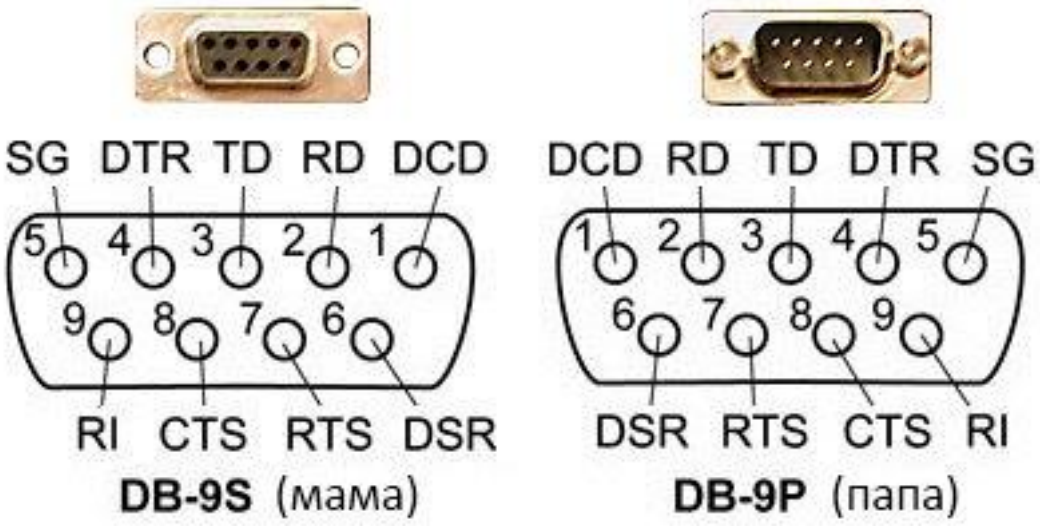


Рисунок 5 – Распиновка разъема DB9 для RS-232

Однако, стандарт RS-232 имеет существенный недостаток. Дело в том, что работает он на расстоянии до 15 метров.

RS-485 совместно с RS-422 лишены этого недостатка. Данные протоколы работают на расстоянии до 1200 метров. Скорость передачи зависит от длины линии связи. На линии 12 метров скорость может достигать 10 Мбит/с, а на линии максимальной длины 1200 метров скорость передачи не превышает 100 кбит/с. Разница в том, что в RS-422 может быть одно передающее устройство, а принимающих до 10. Эта особенность делает RS-422 намного менее распространенным, чем RS-485.

Помимо интерфейсов, перечисленных выше, существует стандарт реализации протокола Modbus на стеке TCP/IP. Технология Ethernet поддерживает большое количество устройств, обеспечивает протоколу Modbus высокую скорость 100 Мбит/с и выше, а также дает возможность преодолевать любые расстояния.

Таким образом, сравнивая все преимущества и недостатки, протокол Modbus на стеке TCP/IP является самым современным и подходящим решением.

1.5. Структура пакета Modbus

ADU (Application Data Unit) – пакет Modbus целиком, вместе с заголовками, PDU, адресом устройства, контрольной суммой и т.д.

PDU (Protocol Data Unit) – основная часть Modbus пакета, одинаковая для всех интерфейсов.

Адрес устройства (адрес slave-устройства) – адрес получателя. В одном сегменте Modbus-сети могут находится до 247 устройств. Master-устройство адреса не имеет. Адрес “0” используется для широковещательных запросов от master-устройства.

Контрольная сумма – алгоритм проверки целостности пакета. Она подсчитывается и добавляется в конец фрейма передающим устройством, и сравнивается принимающим устройством с контрольной суммой, подсчитанной им по принятым данным. В протоколе Modbus TCP контрольная сумма не используется, а проверки целостности вынесена на уровень самого протокола TCP. Структура пакета Modbus отражена на рисунке 6.

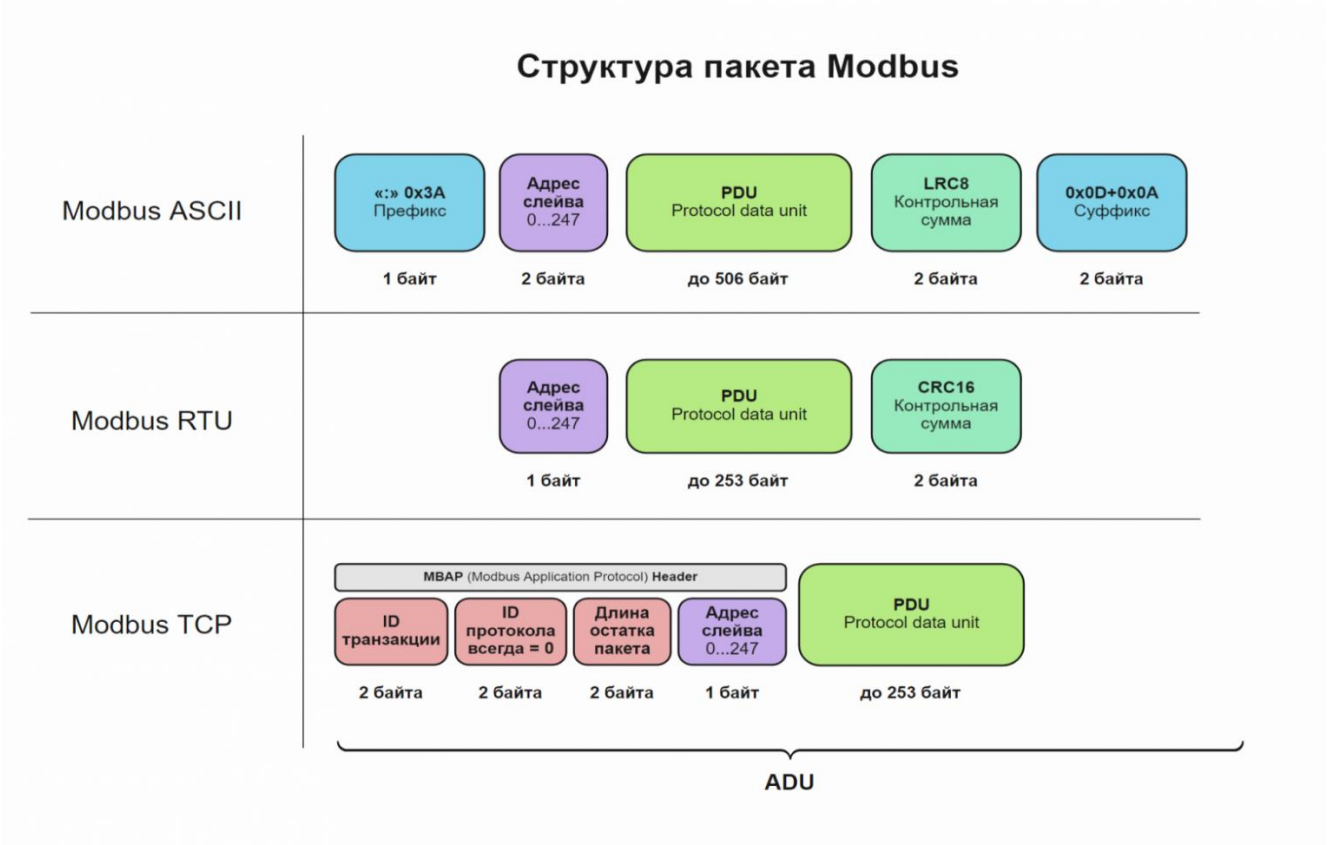


Рисунок 6 – Структура пакета Modbus

Команды Modbus показаны в таблице 1.

Таблица 1 – Типы команд Modbus

Код функции	**Что делает функция**	**Тип значения**	**Тип доступа**
01 (0x01)	Чтение DO (Read Coil Status)	Дискретное	Чтение
02 (0x02)	Чтение DI (Read Input Status)	Дискретное	Чтение
03 (0x03)	Чтение AO (Read Holding Registers)	16 битное	Чтение
04 (0x04)	Чтение AI (Read Input Registers)	16 битное	Чтение
05 (0x05)	Запись одного DO (Force Single Coil)	Дискретное	Запись
06 (0x06)	Запись одного AO (Preset Single Register)	16 битное	Запись
15 (0x0F)	Запись нескольких DO (Force Multiple Coils)	Дискретное	Запись
16 (0x10)	Запись нескольких AO (Preset Multiple Registers)	16 битное	Запись

Пример Modbus TCP пакета. Запрос от master к slave-устройству: 0001 0000 0006 11 03 006B 0003. Расшифровка пакета показана в таблице 2.

Таблица 2 – Запрос от Master к Slave-устройству

Байт	Назначение байта
0001	Идентификатор транзакции
0000	Идентификатор протокола
0006	Длина (Далее идут 6 байт)
11	Адрес устройства(17 = 11 hex)
03	Функциональный код (читаем Analog Output Holding Registers)
006B	Адрес первого регистра (40108-40001 = 107 = 6b (hex)
0003	Количество требуемых регистров (с 40108 по 40110)

Пример ответа slave-устройства: 0001 0000 0009 11 03 06 02 2B 00 64 00 7F. Расшифровка пакета показана в таблице 3.

Таблица 3 – Ответ Slave на запрос Master-устройства

Байт	Назначение байта
0001	Идентификатор транзакции
0000	Идентификатор протокола
0009	Длина (Далее идут 9 байт)
11	Адрес устройства(17 = 11 hex)
03	Функциональный код (читаем Analog Output Holding Registers)
06	Количество байт далее (6 байт идут следом)
02	Значение старшего разряда регистра (02 hex)
2B	Значение младшего разряда регистра (2B hex)
00	Значение старшего разряда регистра (00 hex)
64	Значение младшего разряда регистра (64 hex)
00	Значение старшего разряда регистра (00 hex)
7F	Значение младшего разряда регистра (7F hex)

Итого, имеем: AO0 = 02 2B (HEX) = 555 в десятичной системе счисления. AO1 = 00 64 (HEX) = 100. AO2 = 00 7F (HEX) = 127.

Также, для каждого функционального кода существуют отведенные диапазоны регистров. Диапазоны показаны в таблице 4.

Таблица 4 – Диапазоны регистров

Диапазон	**Функции**
10001-19999	• 02 – чтение группы регистров;
20001-29999	• 01 – чтение группы регистров; • 05 – запись 1 регистра; • 15 – запись группы регистров;
30001-39999	• 04 – чтение группы регистров;
40001-49999	• 03 – чтение группы регистров; • 06 – запись 1 регистра; • 16 – запись группы регистров.

Выводы по главе 1:

1. В настоящее время существует достаточное количество протоколов для коммуникации контроллеров в промышленных сетях. Сеть Ethernet дает возможность обмениваться данными со скоростью до 1 Гбит/с.

2. Ethernet стандарта IEEE 802.3 не может обеспечить функционирование сетей реального времени из-за механизма доступа CSMA/CD.

3. Помимо промышленных сетей, существуют операционные системы реального времени. Наиболее популярная из них – FreeRTOS, доступна на большом количестве микроконтроллеров, в том числе на STMicroelectronics.

4. Реализация систем реального времени возможна и на обычных операционных системах, привычным рядовому пользователю.

5. Modbus TCP является современным решением, удовлетворяющим всем потребностям данной выпускной работы.

2. РАЗРАБОТКА АЛГОРИТМОВ ОБМЕНА ДАННЫМИ С ЭЛЕКТРОННЫМ БЛОКОМ УПРАВЛЕНИЯ В ПРОМЫШЛЕННОЙ СЕТИ ETHERNET

Требуется разработать программу для реализации обмена данными в промышленной сети Ethernet. Для этого необходимо определить алгоритм приема и передачи данных, а также функциональные требования.

Функциональные требования к программе следующие:

1. Реализация протокола Modbus TCP в режиме Slave;
2. Реализация функций Modbus чтения и записи регистров (0x03, 0x06);
3. Определение гарантированного времени доставки пакетов.

Как было упомянуто в главе 1, Modbus работает по принципу ведущий – ведомый. Соответственно, Master-устройство подает запрос, а Slave-устройство отвечает. За Master-устройство была взята панель оператора Kinco GL070E. На примере данной панели составлен алгоритм обмена данными. Схема обмена данными на примере Master-устройства изображена на рисунке 7.

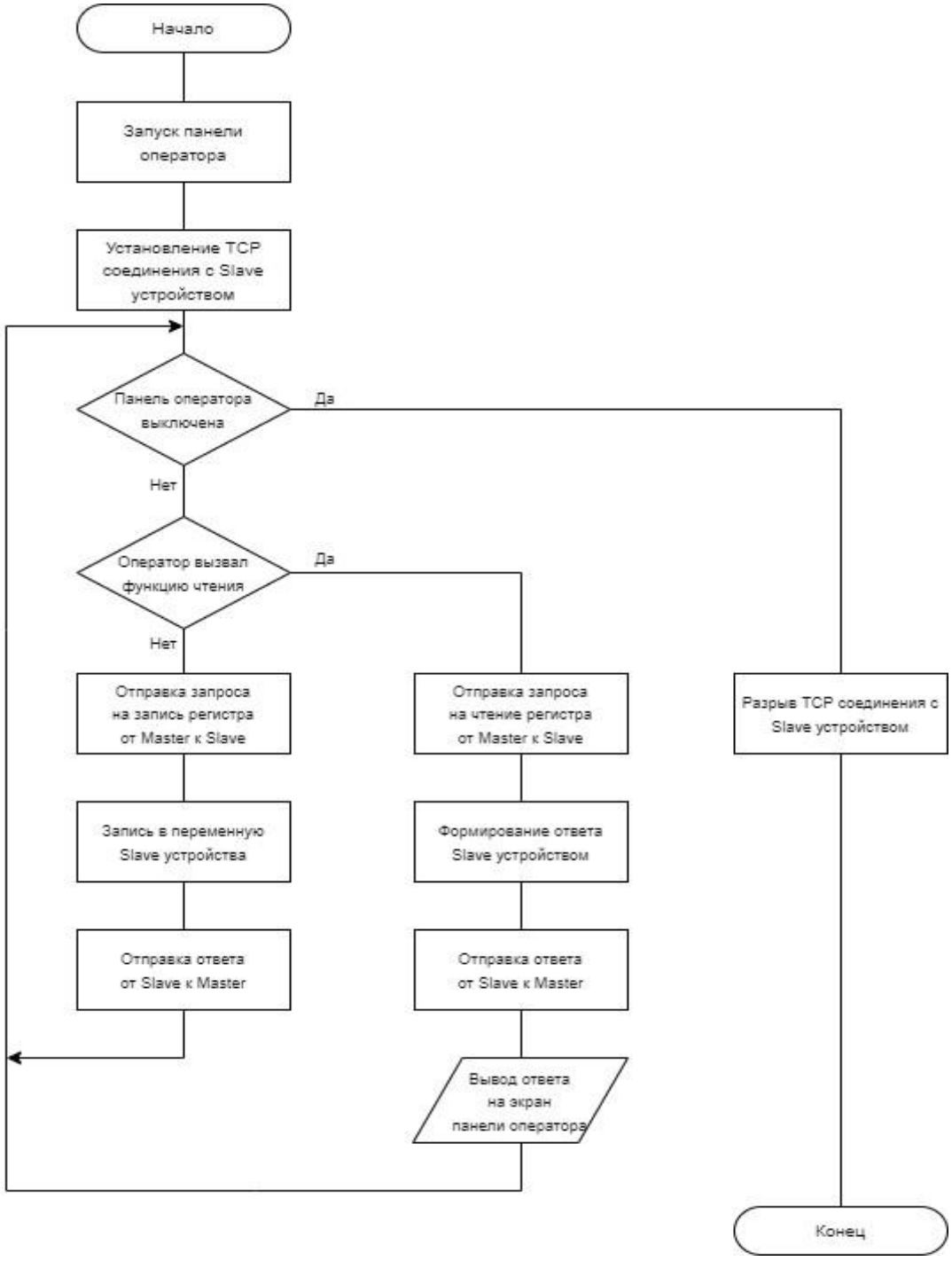


Рисунок 7 – Графическая схема алгоритма Master-устройства Modbus TCP

Выводы по главе 2:

1. Сформированы функциональные требования к программе.

2. Modbus строится по принципу ведущий – ведомый.

3. Определена схема обмена данными на примере Master-устройства Kinco GL070E.

3. РАЗРАБОТКА ПРОГРАММНОГО КОДА ДЛЯ ОБМЕНА ДАННЫМИ С ЭЛЕКТРОННЫМ БЛОКОМ УПРАВЛЕНИЯ В ПРОМЫШЛЕННОЙ СЕТИ ETHERNET

3.1. Подготовка отладочной платы XCore 407I

В основу прототипа Modbus Slave-устройства решено было взять отладочную плату XCore 407I от фирмы Waveshare (рисунок 8). Данная плата сделана на основе микроконтроллера STM32F407IGT6, который имеет на борту интерфейс RMII. С помощью RMII происходит связь с физическим уровнем. RMII предполагает подключение с помощью двух проводов на скорости 50 МГц, что дает нам скорость приема и передачи данных 100 Мбит в секунду.

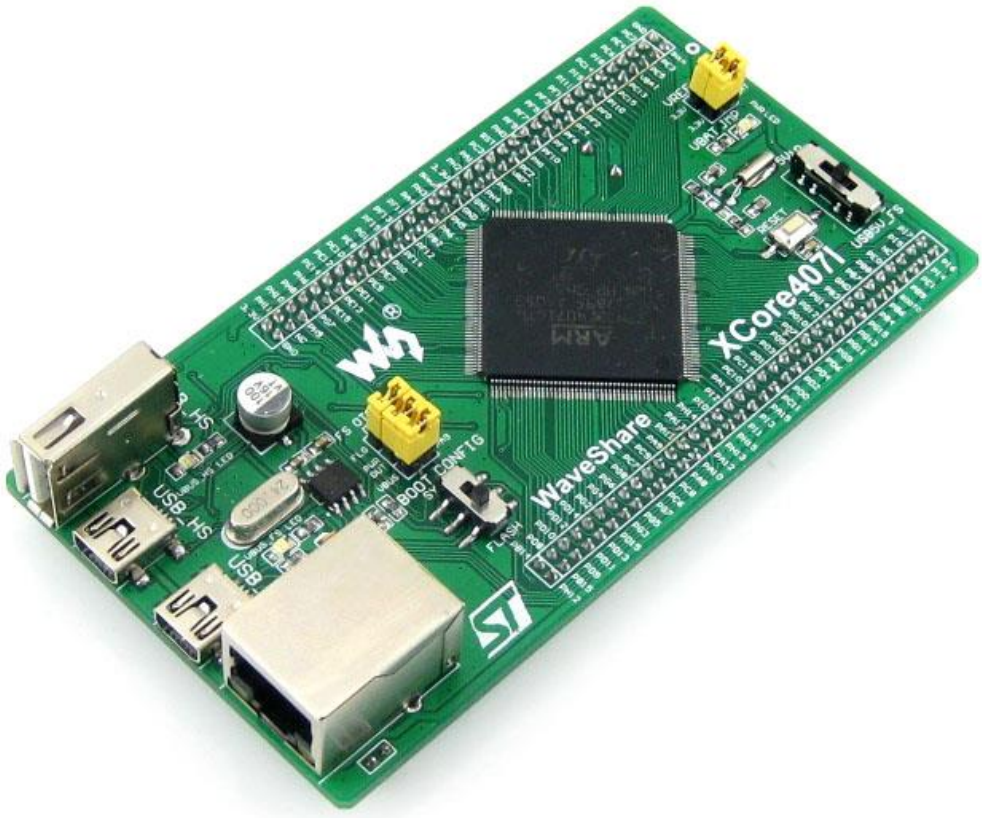


Рисунок 8 – Отладочная плата Waveshare XCore 407I

К отладочной плате также был подключен преобразователь USB – UART (PL2303HX), что позволяет выводить необходимую информацию на экран компьютера для понятной отладки программы (см. рисунок 9).

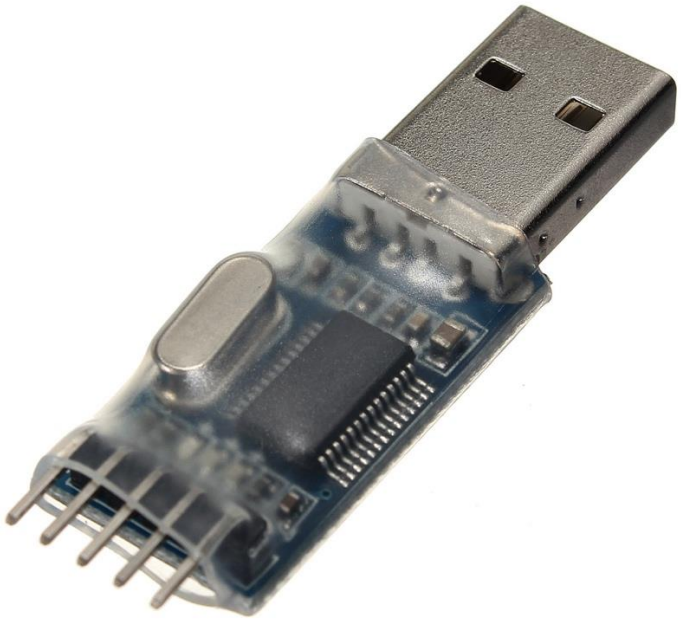


Рисунок 9 – Преобразователь USB – UART на базе PL2303HX

Для подключения данного преобразователя к микроконтроллеру следует определить пины приемопередатчика UART. В документации (datasheet) микроконтроллера указано, что пины у узла USART3 следующие: RX – PC11, TX – PC10. Схема подключения преобразователя к отладочной плате отражена в таблице 5.

Таблица 5 – Схема подключения преобразователя USB – UART к микроконтроллеру STM32F407IGT6

	STM32	USB to UART (PL2303HX)
Красный провод	3.3В	3.3В
Черный провод	GND	GND
Оранжевый провод	PC10	RX
Зеленый провод	PC11	TX

Следует обратить внимание, что пины RX, TX преобразователя интерфейсов USB – UART подключаются к пинам TX, RX микроконтроллера соответственно.

3.2. Разработка программы

Для разработки была выбрана среда программирования Keil uVision 5. Язык программирования: C.

В комплекте с отладочной платой поставляется архив с примерами. Так как наш прототип должен быть Modbus Slave-устройством, за основу был взят проект базового TCP Server с функцией echo.

Для отладки был подключен USB – UART преобразователь, его необходимо инициализировать. Из архива с примерами был взят и вставлен в проект файл конфигурации пинов и заголовочный файл USART. Заголовочный файл подключается в основном файле проекта (main.c). Далее вызываются функции конфигурации.

В файле "tcp_echoserver.c" был изменен порт на 502 (считается стандартным для Modbus TCP), был изменен обработчик пакетов. Изначально, как и предполагается функции echo, входящий пакет копировался и отсылался обратно точно такой же. Сейчас же пакет обрабатывается через созданную библиотеку Modbus (рисунок 10). Данная библиотека Modbus и является результатом выпускной квалификационной работы.

```
230     copyString(mb_request_buf, es->p->payload, es->p->len);
231     uint16_t buf_len = es->p->len;
232     buf_len = mb_process(mb_answer_buf, mb_request_buf, buf_len);
233     copyString(es->p->payload, mb_answer_buf, buf_len);
234     tcp_echoserver_send(tpcb, es);
```

Рисунок 10 – Листинг кода вызова функции обработки пакета Modbus

Входящий пакет поступает в обработчик Modbus. Там определяется функциональный код Modbus, по нему уже вызывается нужная функция(чтение(см. рисунок 12), запись(см. рисунок 13)) и высчитывается длина PDU (см. рисунок 11). В данный момент реализованы функции 0x03 (Read Holding Registers) и 0x06 (Write Holding Register).

```
19 uint16_t mb_process(char *mb_answer_buf, char *mb_request_buf, uint16_t request_buf_len) {
20   memset(mb_answer_buf, '\0', MB_ADU_MAXSIZE);
21   mb_mbap_copy(mb_answer_buf, mb_request_buf);
22   uint16_t pduLen = 0;
23
24   uint8_t func = mb_request_buf[MB_PDU_FUNC];
25
26   switch (func) {
27   case MB_FUNC_READ_COILS:
28     break;
29   case MB_FUNC_READ_DISCRETE:
30     break;
31   case MB_FUNC_READ_HOLDING:
32     pduLen = mb_read(mb_answer_buf, mb_request_buf, request_buf_len);
33     break;
34   case MB_FUNC_READ_INPUT:
35     pduLen = mb_read(mb_answer_buf, mb_request_buf, request_buf_len);
36     break;
37   case MB_FUNC_WRITE_S_COIL:
38   case MB_FUNC_WRITE_S_HOLDING:
39     pduLen = mb_write(mb_answer_buf, mb_request_buf, request_buf_len);
40     break;
41   case MB_FUNC_WRITE_M_COIL:
42   case MB_FUNC_WRITE_M_HOLDING:
43     break;
44   default:
45     pduLen = mb_error(mb_answer_buf, func, MB_EXCEPTION_FUNC_UNSUPPORTED);
46     break;
47   }
48
49   mb_answer_buf[MB_MBAP_LEN_H] = (pduLen + 1) >> 8;
50   mb_answer_buf[MB_MBAP_LEN_L] = (pduLen + 1) & 0xff;
51   return (pduLen + MB_MBAP_SIZE);
52 }
```

Рисунок 11 – Листинг функции mb_process

```
54 static uint16_t mb_read(char *mb_answer_buf, char *mb_request_buf, uint16_t request_buf_len) {
55   uint8_t func = mb_request_buf[MB_PDU_FUNC];
56   uint16_t start_address = mb_request_buf[MB_PDU_R_ST_ADDR_L] + (mb_request_buf[MB_PDU_R_ST_ADDR_H] << 8);
57   uint16_t quantity = mb_request_buf[MB_PDU_R_QUANTITY_L] + (mb_request_buf[MB_PDU_R_QUANTITY_H] << 8);
58
59   if (mb_start_address(func, start_address, quantity) != MB_EXCEPTION_OK) {
60     return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_ADDR);
61   } else {
62
63     mb_answer_buf[MB_PDU_FUNC] = func;
64     mb_answer_buf[MB_PDU_REPL_N] = mb_pdu_calculate_N(func, quantity);
65
66     switch (func) {
67     case MB_FUNC_READ_COILS:
68       break;
69     case MB_FUNC_READ_DISCRETE:
70       break;
71     case MB_FUNC_READ_HOLDING:
72       mb_answer_buf[MB_PDU_FUNC + 2] = datak1;
73       mb_answer_buf[MB_PDU_FUNC + 3] = datak0;
74       break;
75     case MB_FUNC_READ_INPUT:
76       break;
77     }
78
79   }
80   return mb_pdu_calculate_N(func, quantity) + 2;
81 }
82
```

Рисунок 12 – Листинг функции mb_read

```
83  static uint16_t mb_write(char *mb_answer_buf, char *mb_request_buf, uint16_t request_buf_len) {
84    uint8_t func = mb_request_buf[MB_PDU_FUNC];
85    uint16_t address = mb_request_buf[MB_PDU_W_REG_ADDR_L] + (mb_request_buf[MB_PDU_W_REG_ADDR_H] << 8);
86    uint16_t valToWrite = mb_request_buf[MB_PDU_W_REG_VAL_L] + (mb_request_buf[MB_PDU_W_REG_VAL_H] << 8);
87
88    if (mb_start_address(func, address, 1) != MB_EXCEPTION_OK) {
89      return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_ADDR);
90    } else if (mb_process_val(func, valToWrite) != MB_EXCEPTION_OK) {
91      return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_VAL);
92    } else {
93
94      mb_answer_buf[MB_PDU_FUNC] = func;
95      mb_answer_buf[MB_PDU_W_REG_ADDR_H] = mb_request_buf[MB_PDU_W_REG_ADDR_H];
96      mb_answer_buf[MB_PDU_W_REG_ADDR_L] = mb_request_buf[MB_PDU_W_REG_ADDR_L];
97      mb_answer_buf[MB_PDU_W_REG_VAL_H] = mb_request_buf[MB_PDU_W_REG_VAL_H];
98      mb_answer_buf[MB_PDU_W_REG_VAL_L] = mb_request_buf[MB_PDU_W_REG_VAL_L];
99
100     datak0 = mb_request_buf[MB_PDU_W_REG_VAL_L];
101     datak1 = mb_request_buf[MB_PDU_W_REG_VAL_H];
102     printf("L: %02X , H: %02X", datak0, datak1);
103
104   }
105   return 5;
106 }
```

Рисунок 13 – Листинг функции mb_write

Выводы по главе 3:

1. Используемая среда разработки в данной работе – Keil uVision 5. Язык программирования: C.

2. Для реализации Modbus TCP протокола за основу был взят TCP Echo сервер.

3. Изменен обработчик пакетов.

4. Написана библиотека Modbus. Функция mb_process данной библиотеки обрабатывает и выдает измененный PDU.

5. Полный исходный код представлен в листинге А.1, А.2 приложения А.

4. ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММНОГО КОДА

В качестве Master-устройства была выбрана панель оператора Kinco GL070E, работающая с протоколом Modbus. В данной панели есть интерфейс RS-485, а также Ethernet.

В среде разработки Kinco DTools был создан новый проект. Панель оператора была сконфигурирована с IP-адресом 192.168.1.100.

На главный экран добавлено текстовое поле (рисунок 14). Данное текстовое поле как выводит значение с переменной контроллера, так и может изменять эту переменную (см. рисунок 15). Адрес Modbus-регистра – 1.

Рисунок 14 – Главный экран Kinco GL070E

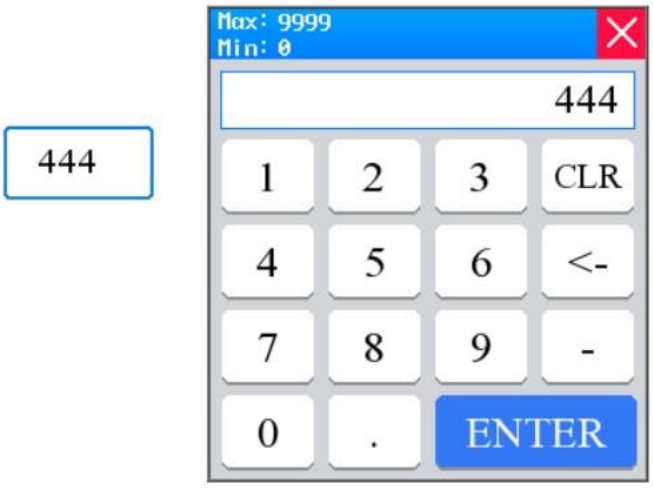


Рисунок 15 – Редактирование переменной в микроконтроллере STM32 через Kinco HMI

После редактирования переменной через панель оператора, отсылается пакет на slave-устройство. Микроконтроллер обрабатывает данный пакет, совершает запись двух байт из входящего пакета в переменную. Для проверки правильности передачи данных, микроконтроллер отсылает только что измененную переменную по UART. Результат можем видеть на рисунке 16.

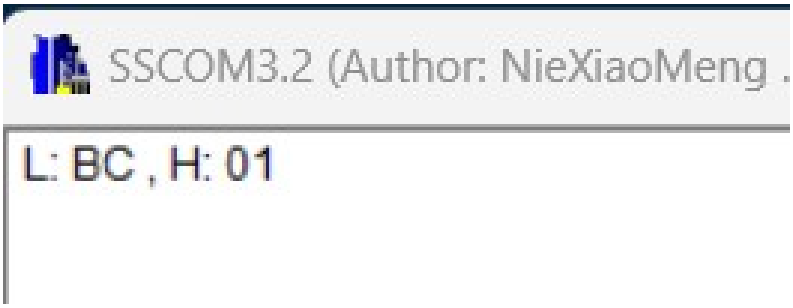


Рисунок 16 – Программа SSCOM для мониторинга порта UART

Получили результат: 01 BC (HEX), что означает 444 в десятичной системе счисления (см. рисунок 17). Данный результат совпадает с введенным числом в панели оператора.

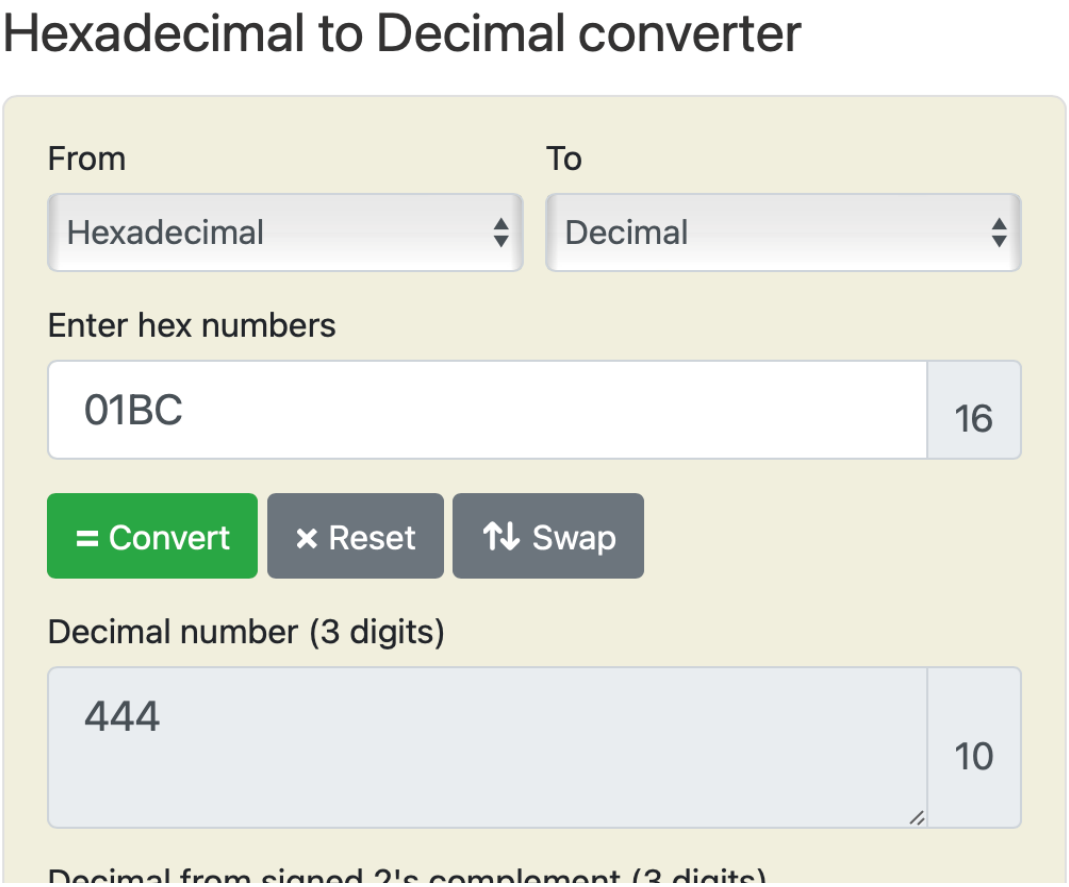


Рисунок 17 – Перевод полученного результата в десятичную систему счисления

Следующим этапом тестирования является определение гарантированного времени доставки пакетов. Средством измерения выбрана программа-анализатор трафика Wireshark. Были отправлены 5 запросов от Master-устройства, а также получены 5 ответов от Slave-устройства (в данном случае микроконтроллера STM32) (рисунок 18). Полученные результаты отражены в таблице 6.

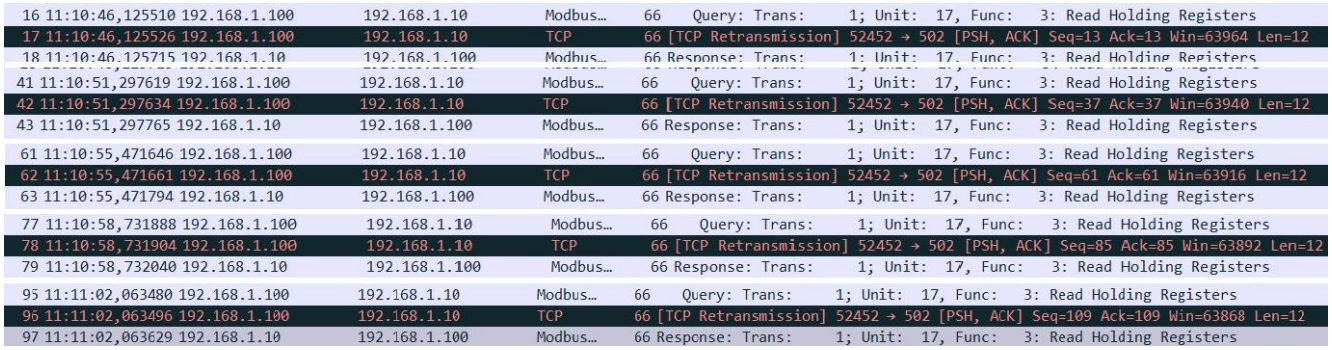


Рисунок 18 – Снимок пакетов из программы-анализатора Wireshark

Таблица 6 – Результаты определения гарантированного времени доставки пакетов

Время отправки пакета от Master к Slave-устройству	**Время получения ответа от Slave Master-устройством**	**Результат, миллисекунды**
11:10:46,125510	11:10:46,125715	0,205 мс
11:10:51,297619	11:10:51,297765	0,146 мс
11:10:55,471646	11:10:55,471794	0,148 мс
11:10:58,731888	11:10:58,732040	0,152 мс
11:11:02,063480	11:11:02,063629	0,149 мс

Максимальное время доставки пакетов равняется 0,205 мс. Можем гарантировать время доставки пакетов менее 1 миллисекунды.

Выводы по главе 4:

1. Программа успешно прошла тестирование. Проверены функции чтения и записи (0x03 и 0x06 соответственно).
2. Гарантированное время доставки пакетов – менее 1 миллисекунды.
3. Исходный код представлен в листингах А.1, А.2 приложения А.

ЗАКЛЮЧЕНИЕ

В результате работы была разработана программа для обмена данными в промышленной сети Ethernet с гарантированным временем доставки пакетов. Программа прошла тестирование, было получено гарантированное время доставки пакетов. Для решения поставленной задачи, был проведен следующий ряд работ:

1. Проведен аналитический обзор научно-технической, нормативной и методической литературы по тематике работы;
2. Разработаны алгоритмы обмена данными с электронным блоком управления в промышленной сети Ethernet с гарантированным временем доставки пакетов.
3. Разработан программный код для обмена данными с электронным блоком управления в промышленной сети Ethernet.
4. Произведена отладка и тестирование программного кода.

Таким образом, программа позволяет обмениваться данными в промышленной сети Ethernet с помощью протокола Modbus TCP. Гарантированное время доставки пакетов – менее 1 миллисекунды.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Современные принципы построения промышленных АСУ. [Электронный ресурс]. URL: http://wwwcdl.bmstu.ru/mt10/UTS/frames4.html (дата обращения: 16.05.2024).

2. What is Modbus and How does it work? [Электронный ресурс]. URL: https://www.se.com/us/en/faqs/FA168406/ (дата обращения: 16.05.2024).

3. Сети Real-Time Ethernet: от теории к практической реализации. [Электронный ресурс]. URL: https://www.cta.ru/cms/f/434753.pdf (дата обращения: 16.05.2024).

4. EtherCAT Tutorial An Introduction for Real-Time Hardware Communication on Windows. [Электронный ресурс]. URL: https://biblio.vub.ac.be/vubirfiles/36814626/ShortPaper_Tutorial.pdf (дата обращения: 16.05.2024).

5. PROFINET. [Электронный ресурс]. URL: https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10526256?tree=CatalogTree (дата обращения: 16.05.2024).

6. PROFINET – RT vs IRT. [Электронный ресурс]. URL: https://assets.new.siemens.com/siemens/assets/api/uuid:30e0ba94-b1af-4488-98dd-8fe716382cc0/siemens-profinet-rt-vs-irt-webinar-13oct2020.pdf (дата обращения: 16.05.2024).

7. Selection of PROFINET device classes and technology. [Электронный ресурс]. URL: https://assets.new.siemens.com/siemens/assets/api/uuid:bcc1c23d-bbd4-4582-b5f9-dc199b9742d4/difa-b10254-00-7600profinetdeviceclassesen-144.pdf (дата обращения: 16.05.2024).

8. POWERLINK – Технология. [Электронный ресурс]. URL: https://www.br-automation.com/ru/produkcija/seti-i-moduli-polevykh-shin/additional-information/powerlink-tekhnologija/ (дата обращения: 16.05.2024).

9. EtherNet/IP. Take advantage of native plant connectivity. [Электронный ресурс]. URL:

https://portal.endress.com/dla/5000961/5120/000/01/CP01099S_04_EN_04.21.pdf (дата обращения: 16.05.2024).

10. EtherNet/IP Network Devices. [Электронный ресурс]. URL: https://literature.rockwellautomation.com/idc/groups/literature/documents/um/enet-um006_-en-p.pdf (дата обращения 16.05.2024).

11. About FreeRTOS Kernel. [Электронный ресурс]. URL: https://www.freertos.org/RTOS.html (дата обращения 16.05.2024).

12. RTX Real-Time Operating System. [Электронный ресурс]. URL: https://www.keil.com/arm/rl-arm/kernel.asp (дата обращения: 16.05.2024).

13. Операционная система uC/OS-II. [Электронный ресурс]. URL: https://e-learning.bmstu.ru/iu6/pluginfile.php/597/mod_folder/content/0/uCOSII-intro.pdf (дата обращения: 16.05.2024).

14. Windows for real-time applications through Real-Time Accelerator technology. [Электронный ресурс]. URL: https://www.acontis.com/en/windows-realtime.html (дата обращения: 16.05.2024).

15. Technical details of the real-time preemption. [Электронный ресурс]. URL: https://wiki.linuxfoundation.org/realtime/documentation/technical_details/start (дата обращения: 16.05.2024).

ПРИЛОЖЕНИЕ А. Фрагменты исходного кода программы обмена данными в промышленной сети Ethernet

Листинг А.1 – Исходный код modbus.c

```
#include "modbus.h"
#include <string.h>

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

static uint16_t mb_read(char *mb_answer_buf, char *mb_request_buf, uint16_t
request_buf_len); //функция чтения
static uint16_t mb_write(char *mb_answer_buf, char *mb_request_buf, uint16_t
request_buf_len); //функция записи
static uint16_t mb_error(char *mb_answer_buf, uint8_t func, uint16_t
exceptionCode); //вызывается при некорректном пакете
static uint8_t mb_start_address(uint16_t func, uint16_t start_address,
uint16_t quantity); //проверка адреса получателя
static uint8_t mb_process_val(uint16_t func, uint16_t val); //проверка
передаваемой переменной
static void mb_mbap_copy(char *mb_answer_buf, char *mb_request_buf);
//функция копирования пакета
static uint16_t mb_pdu_calculate_N(uint16_t func, uint16_t quantity);
//подсчет длины сообщения

uint8_t datak0 = 0x2B;
uint8_t datak1 = 0x02;

uint16_t mb_process(char *mb_answer_buf, char *mb_request_buf, uint16_t
request_buf_len) { //основная ф-ция библиотеки
  memset(mb_answer_buf, '\0', MB_ADU_MAXSIZE);
  mb_mbap_copy(mb_answer_buf, mb_request_buf);
  uint16_t pduLen = 0;

  uint8_t func = mb_request_buf[MB_PDU_FUNC];

  switch (func) { //исходя из функц. кода, вызываем нужную функцию
чтения/записи, а нужная ф-ция в свою очередь возвращает длину pdu
        case MB_FUNC_READ_COILS:
             break;
        case MB_FUNC_READ_DISCRETE:
             break;
        case MB_FUNC_READ_HOLDING:
             pduLen      =      mb_read(mb_answer_buf,      mb_request_buf,
request_buf_len);
             break;
        case MB_FUNC_READ_INPUT:
             pduLen      =      mb_read(mb_answer_buf,      mb_request_buf,
request_buf_len);
             break;
        case MB_FUNC_WRITE_S_COIL:
        case MB_FUNC_WRITE_S_HOLDING:
             pduLen      =      mb_write(mb_answer_buf,      mb_request_buf,
request_buf_len);
             break;
        case MB_FUNC_WRITE_M_COIL:
        case MB_FUNC_WRITE_M_HOLDING:
```

Продолжение листинга А.1

```
            break;
      default:
            pduLen = mb_error(mb_answer_buf, func,
      MB_EXCEPTION_FUNC_UNSUPPORTED);  //вызываем ф-цию ошибки, если
      вызываемая ф-ция не найдена
            break;
   }

   mb_answer_buf[MB_MBAP_LEN_H] = (pduLen + 1) >> 8;
   mb_answer_buf[MB_MBAP_LEN_L] = (pduLen + 1) & 0xff;
   return (pduLen + MB_MBAP_SIZE);
}

static uint16_t mb_read(char *mb_answer_buf, char *mb_request_buf, uint16_t
request_buf_len) { //ф-ция чтения, принимает массив исходящего пакета,
входящий пакет и его длину
   uint8_t func = mb_request_buf[MB_PDU_FUNC];
   uint16_t start_address = mb_request_buf[MB_PDU_R_ST_ADDR_L] +
(mb_request_buf[MB_PDU_R_ST_ADDR_H] << 8);
   uint16_t quantity = mb_request_buf[MB_PDU_R_QUANTITY_L] +
(mb_request_buf[MB_PDU_R_QUANTITY_H] << 8);

   if (mb_start_address(func, start_address, quantity) != MB_EXCEPTION_OK) {
         return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_ADDR);
   } else {
         mb_answer_buf[MB_PDU_FUNC] = func;
         mb_answer_buf[MB_PDU_REPL_N] = mb_pdu_calculate_N(func, quantity);

         switch (func) {
               case MB_FUNC_READ_COILS:
                     break;
               case MB_FUNC_READ_DISCRETE:
                     break;
               case MB_FUNC_READ_HOLDING:
                     mb_answer_buf[MB_PDU_FUNC + 2] = datak1;
                     mb_answer_buf[MB_PDU_FUNC + 3] = datak0;
                     break;
               case MB_FUNC_READ_INPUT:
                     break;
         }

   }
   return mb_pdu_calculate_N(func, quantity) + 2;
}

static uint16_t mb_write(char *mb_answer_buf, char *mb_request_buf, uint16_t
request_buf_len) { //ф-ция записи, принимает массив исходящего пакета,
входящий пакет и его длину
   uint8_t func = mb_request_buf[MB_PDU_FUNC];
   uint16_t address = mb_request_buf[MB_PDU_W_REG_ADDR_L] +
(mb_request_buf[MB_PDU_W_REG_ADDR_H] << 8);
   uint16_t valToWrite = mb_request_buf[MB_PDU_W_REG_VAL_L] +
(mb_request_buf[MB_PDU_W_REG_VAL_H] << 8);

   if (mb_start_address(func, address, 1) != MB_EXCEPTION_OK) {
         return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_ADDR);
   } else if (mb_process_val(func, valToWrite) != MB_EXCEPTION_OK) {
```

Продолжение листинга А.1

```
        return mb_error(mb_answer_buf, func, MB_EXCEPTION_DATA_VAL);
   }
   else {

        mb_answer_buf[MB_PDU_FUNC] = func;
        mb_answer_buf[MB_PDU_W_REG_ADDR_H]                                    =
mb_request_buf[MB_PDU_W_REG_ADDR_H];
        mb_answer_buf[MB_PDU_W_REG_ADDR_L]                                    =
mb_request_buf[MB_PDU_W_REG_ADDR_L];
        mb_answer_buf[MB_PDU_W_REG_VAL_H]                                     =
mb_request_buf[MB_PDU_W_REG_VAL_H];
        mb_answer_buf[MB_PDU_W_REG_VAL_L]                                     =
mb_request_buf[MB_PDU_W_REG_VAL_L];

        datak0 = mb_request_buf[MB_PDU_W_REG_VAL_L];
        datak1 = mb_request_buf[MB_PDU_W_REG_VAL_H];
        printf("L: %02X , H: %02X", datak0, datak1);

   }
   return 5;
}

static void mb_mbap_copy(char *mb_answer_buf, char *mb_request_buf) {
   mb_answer_buf[MB_MBAP_TRANSACTION_ID_H]                                    =
mb_request_buf[MB_MBAP_TRANSACTION_ID_H];
   mb_answer_buf[MB_MBAP_TRANSACTION_ID_L]                                    =
mb_request_buf[MB_MBAP_TRANSACTION_ID_L];
   mb_answer_buf[MB_MBAP_PROTOCOL_ID_H]                                       =
mb_request_buf[MB_MBAP_PROTOCOL_ID_H];
   mb_answer_buf[MB_MBAP_PROTOCOL_ID_L]                                       =
mb_request_buf[MB_MBAP_PROTOCOL_ID_L];
   mb_answer_buf[MB_MBAP_CLIENT_ID] = mb_request_buf[MB_MBAP_CLIENT_ID];
}

static  uint8_t  mb_start_address(uint16_t  func,  uint16_t  start_address,
uint16_t quantity) {
   uint8_t exception_code = MB_EXCEPTION_OK;
   switch (func) {
        case MB_FUNC_READ_COILS:
             if ((start_address + quantity) > MB_COILS_Q)
                  exception_code = 2;
             break;
        case MB_FUNC_READ_DISCRETE:
             if ((start_address + quantity) > MB_DISCRETE_Q)
                  exception_code = 2;
             break;
        case MB_FUNC_READ_HOLDING:
             if ((start_address + quantity) > MB_HOLDING_Q)
                  exception_code = 2;
             break;
        case MB_FUNC_READ_INPUT:
             if ((start_address + quantity) > MB_INPUT_Q)
                  exception_code = 2;
             break;

        case MB_FUNC_WRITE_S_COIL:
             if ((start_address + quantity) > MB_COILS_Q)
```

Окончание листинга А.1

```
                    exception_code = 2;
              break;
        case MB_FUNC_WRITE_S_HOLDING:
              if ((start_address + quantity) > MB_HOLDING_Q)
                    exception_code = 2;
              break;
   }
   return exception_code;
}

static uint8_t mb_process_val(uint16_t func, uint16_t val) {
   uint8_t exception_code = MB_EXCEPTION_OK;
   switch (func) {
        case MB_FUNC_WRITE_S_COIL:
              if ((val != 0xFF00) && (val != 0x0000))
                    exception_code = 3;
              break;
   }
   return exception_code;
}



static uint16_t mb_error(char *mb_answer_buf, uint8_t func, uint16_t
exceptionCode) {
   mb_answer_buf[MB_PDU_FUNC] = func | 0x80;
   mb_answer_buf[MB_PDU_EXCEPTION_CODE] = exceptionCode;
   return 2;
}

static uint16_t mb_pdu_calculate_N(uint16_t func, uint16_t quantity) {
   switch (func) {
        case MB_FUNC_READ_COILS:
        case MB_FUNC_READ_DISCRETE:
              if (quantity % 8 != 0)
                    return ((quantity / 8) + 1);
              else return (quantity / 8);
              break;
        case MB_FUNC_READ_HOLDING:
        case MB_FUNC_READ_INPUT:
              return quantity * 2;
              break;
        default:
              return 0;
   }
}
```

Листинг А.2 – исходный код modbus.h

```
#ifndef INC_MODBUS_H_
#define INC_MODBUS_H_
#include "inttypes.h"

// ---------КОМАНДЫ-------------------------------//
#define MB_FUNC_READ_COILS          (0x01)
#define MB_FUNC_READ_DISCRETE       (0x02)
```

Окончание листинга А.2

```
#define MB_FUNC_READ_HOLDING         (0x03)
#define MB_FUNC_READ_INPUT           (0x04)
#define MB_FUNC_WRITE_S_COIL         (0x05)
#define MB_FUNC_WRITE_S_HOLDING      (0x06)
#define MB_FUNC_WRITE_M_COIL         (0x0F)
#define MB_FUNC_WRITE_M_HOLDING      (0x10)
//---------КОДЫ ОШИБОК-----------------------//
#define MB_EXCEPTION_OK                                (0x00)
#define MB_EXCEPTION_FUNC_UNSUPPORTED                  (0x01)
#define MB_EXCEPTION_DATA_ADDR                         (0x02)
#define MB_EXCEPTION_DATA_VAL                          (0x03)
//---------MB TCP----------------------//
#define MB_MBAP_TRANSACTION_ID_H     (0)
#define MB_MBAP_TRANSACTION_ID_L     (1)
#define MB_MBAP_PROTOCOL_ID_H        (2)
#define MB_MBAP_PROTOCOL_ID_L        (3)
#define MB_MBAP_LEN_H                (4)
#define MB_MBAP_LEN_L                (5)
#define MB_MBAP_CLIENT_ID            (6)
#define MB_MBAP_SIZE                 (7)
#define MB_TCP_PORT                  (502)
#define MB_MBAP_END                  (MB_MBAP_CLIENT_ID)

#define MB_PDU_MAXSIZE               (253)
// ----------------------------------------------------------//
#define MB_ADU_MAXSIZE         (MB_PDU_MAXSIZE + MB_MBAP_SIZE)
#define MB_PDU_START           (MB_MBAP_END+1)

#define MB_COILS_ST            (0)
#define MB_COILS_Q             (0xFF)
#define MB_DISCRETE_ST         (MB_COILS_ST + MB_COILS_Q)
#define MB_DISCRETE_Q          (0)
#define MB_HOLDING_ST          (MB_DISCRETE_ST + MB_DISCRETE_Q)
#define MB_HOLDING_Q           (4)
#define MB_INPUT_ST            (MB_HOLDING_ST + MB_HOLDING_Q)
#define MB_INPUT_Q             (0)
#define MB_MEMORY_SIZE         (MB_COILS_Q + MB_DISCRETE_Q + MB_HOLDING_Q +
MB_INPUT_Q)
// ----------------------------------------------------------//
//---------АДРЕСА В PDU-----------------------//
#define MB_PDU_FUNC                              (MB_PDU_START)
#define MB_PDU_R_ST_ADDR_H                       (MB_PDU_START + 1)
#define MB_PDU_R_ST_ADDR_L                       (MB_PDU_START + 2)
#define MB_PDU_R_QUANTITY_H                      (MB_PDU_START + 3)
#define MB_PDU_R_QUANTITY_L                      (MB_PDU_START + 4)

#define MB_PDU_W_REG_ADDR_H                      (MB_PDU_START + 1)
#define MB_PDU_W_REG_ADDR_L                      (MB_PDU_START + 2)
#define MB_PDU_W_REG_VAL_H                       (MB_PDU_START + 3)
#define MB_PDU_W_REG_VAL_L                       (MB_PDU_START + 4)

#define MB_PDU_EXCEPTION_CODE                    (MB_PDU_START + 1)
#define MB_PDU_REPL_N                            (MB_PDU_START + 1)

uint16_t  mb_process(char  *mb_answer_buf,  char  *mb_request_buf,  uint16_t
request_buf_len);
#endif /* INC_MODBUS_H_ */
```