ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

ЭЛЕКТРОННАЯ БИБЛИОТЕЧНАЯ СИСТЕМА

САМАРА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА «ИНЖЕНЕРИЯ ЗНАНИЙ»

ПРОГРАММИРОВАНИЕ КОММУНИКАЦИОННОГО ИНТЕРФЕЙСА В СИСТЕМАХ СБОРА ИНФОРМАЦИИ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА MODBUS

Методические указания к лабораторной работе по дисциплине «Программное обеспечение микропроцессорных систем» для студентов по специальности 230105 «Программное обеспечение вычислительной техники и автоматизированных систем»

Составитель
к.т.н. **Тулупова В.В.**

Самара
2012

ПРОГРАММИРОВАНИЕ КОММУНИКАЦИОННОГО ИНТЕРФЕЙСА В СИСТЕМАХ СБОРА ИНФОРМАЦИИ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА MODBUS

Цель работы: изучение методов и приемов программирования коммуникационных интерфейсов для систем сбора информации на примере протокола Modbus.

1 Распределенные системы сбора информации и управления. Общие сведения о промышленных сетях.

С ростом количества датчиков, увеличением площади территории, на которой расположена автоматизированная система и усложнением алгоритмов управления становится более эффективным применение распределенных систем [1]. *Распределенные системы* состоят из множества контроллеров и модулей ввода-вывода, взаимодействующих между собой для выполнения общих задач. Каждый контроллер работает со своей группой устройств ввода-вывода и обслуживает определенную часть объекта управления. Следует отметить, что понятие «распределенные системы» охватывает довольно большую область приложений систем сбора информации и управления. С одной стороны, в предельном случае, элементы системы могут находиться на разных континентах земного шара, а связь между ними может выполняться через Интернет. С другой стороны, контроллеры и устройства ввода-вывода могут быть расположены в одном крейте, взаимодействуя между собой по общей шине обмена данных (объединительной магистрали).

Преимущества таких программно-аппаратных решений хорошо известны:

- большое быстродействие благодаря распределению задач между параллельно работающими процессорами;
- повышенная надежность (отказ одного из контроллеров не влияет на работоспособность других);
- более простое наращивание или реконфигурирование системы, приводящее к уменьшению трудозатрат инсталляции системы;
- улучшенная помехоустойчивость и точность благодаря уменьшению длины линий передачи аналоговых сигналов от датчиков к устройствам ввода;
- меньший объем кабельной продукции, пониженные требования к кабелю и более низкая его стоимость;
- меньшие расходы на монтаж и обслуживание кабельных соединений;

Обмен информацией между устройствами распределенной системы происходит в общем случае через *промышленную сеть* (*Fieldbus*, «полевую шину»). *Fieldbus* является цифровой, двунаправленной, многоточечной, последовательной коммуникационной сетью, используемой для связи

изолированных друг от друга устройств, таких как контроллеры, датчики, силовые приводы и т. п.

Промышленные сети должны отвечать специфическому набору требований:

- жесткая детерминированность поведения, предполагающая, что все возможные события в сети могут быть заранее определены;
- работа на длинных линиях с использованием недорогих физических сред (например, витая пара);
- повышенная надежность физического и канального уровней передачи данных для работы в промышленной среде (при высоком уровне электромагнитных помех);
- наличие специальных высоконадежных механических соединительных компонентов.

В настоящее время насчитывается более 50 типов промышленных сетей (Modbus, Profibus, DeviceNet, CANopen, LonWorks, ControlNet, SDS, Seriplex, ArcNet, BACnet, FDDI, FIP, FF, ASI, Ethernet, WorldFIP, Foundation Fieldbus, Interbus, BitBus и др.). Однако широко распространенными является только часть из них. В России подавляющее большинство АСУ ТП используют сети Modbus и Profibus. В последние годы возрос интерес к сетям на основе CANopen и DeviceNet.

Соединение промышленной сети с ее компонентами (устройствами, узлами сети) выполняется с помощью *интерфейсов*. *Сетевым (коммуникационным) интерфейсом* называют логическую и (или) физическую границу между устройством и средой передачи информации. Обычно этой границей является набор аппаратных компонентов и связанного с ними программного обеспечения. При различных модификациях внутренней структуры устройства или программного обеспечения интерфейс остается без изменений, что является одним из признаков, позволяющих выделить интерфейс в составе оборудования.

Для обмена информацией взаимодействующие устройства должны иметь одинаковый *протокол обмена*. В простейшей форме протокол – это набор правил, которые управляют обменом информацией. Он определяет синтаксис и семантику сообщений, операции управления, синхронизацию и состояния при коммуникации. Протокол может быть реализован аппаратно, программно или программно-аппаратно. Название сети обычно совпадает с названием протокола, что объясняется его определяющей ролью при создании сети.

Обычно сеть использует несколько протоколов, образующих *стек протоколов* – набор связанных коммуникационных протоколов, которые функционируют совместно и используют некоторые или все семь уровней модели взаимодействия открытых систем OSI (Open System Interconnection) [2]. Для большинства сетей стек протоколов реализован с помощью специализированных сетевых микросхем или встроен в универсальный микроконтроллер.

В любой модели взаимодействия можно выделить устройство, которое управляет другим (подчиненным) устройством. Устройство, инициирующее обмен, называют *ведущим*, *активным* или *главным* (*Master*, eng.). Устройство, которое отвечает на запросы Master, называют *ведомым*, *пассивным* или *подчиненным* (*Slave*, eng.). В сети может быть одно или несколько ведущих устройств. Такие сети называются *одномастерными* или *многомастерными* соответственно. В многомастерной сети возникает проблема разрешения конфликтов между устройствами, пытающимися одновременно получить доступ к среде передачи информации. Конфликты могут быть разрешены методом передачи маркера (как, например, в сети Profibus), методом побитного сравнения идентификатора (используется в CAN), методом прослушивания сети (используется в Ethernet) и методом предотвращения коллизий (используется в беспроводных сетях) [1].

Во всех сетях применяется *широковещательная рассылка* без определенного адреса, т.е. всем участникам сети. Такой режим используется обычно для синхронизации процессов в сети, например для одновременного запуска процесса ввода данных всеми устройствами ввода или для синхронизации часов.

Передача информации в сети выполняется через канал между передающим и приемным устройством. Канал является понятием теории информации и включает в себя линию связи и приемопередающие устройства. В общем случае вместо термина «*канал связи*» используют термин «*среда передачи*», в качестве которой может выступать, например, оптоволокно, радиоэфир или витая пара проводов.

Сети могут иметь топологию «*звезда*», «*кольцо*», «*шина*» или *смешанную* [2]. «Звезда» в промышленной автоматизации используется редко. «Кольцо» используется в основном для передачи маркера в многомастерных сетях. Шинная топология является самой общепринятой.

2 Modbus – протокол Fieldbus

Протокол Modbus и сеть Modbus [3-5] являются самыми распространенными в мире. Несмотря на свой возраст (стандартом де-факто Modbus стал ещё в 1979 г.) Modbus не только не устарел, но, наоборот, существенно возросло количество новых разработок и объем организационной поддержки этого протокола.

Преимуществами Modbus являются отсутствие необходимости в специализированных интерфейсных микросхемах, простота программной реализации и элегантность принципов функционирования. Высокая степень открытости протокола обеспечивается полностью бесплатными текстами стандартов, которые можно скачать с сайта www.modbus-IDA.org. Кроме того, Modbus имеет высокую достоверность передачи данных, связанную с применением надежного метода контроля ошибок. Modbus позволяет

унифицировать команды обмена благодаря стандартизации номеров (адресов) регистров и функций их чтения-записи.

Основным недостатком Modbus является сетевой обмен по типу «ведущий/ведомый», что не позволяет ведомым устройствам передавать данные по мере их появления и поэтому требует интенсивного опроса ведомых устройств ведущим.

Разновидностями Modbus являются протоколы Modbus Plus [5] – многомастерный протокол с кольцевой передачей маркера и Modbus TCP [6], рассчитанный на использование в сетях Ethernet и Интернет.

Протокол Modbus имеет два режима передачи: RTU и ASCII. В ASCII-режиме сообщение всегда начинается с символа “**:**” и заканчивается последовательностью символов “возврат каретки-перевод строки” (“\r\n”), а каждый байт сообщения содержит два символа в кодировке ASCII. Интервалы между символами сообщения могут быть до 1 секунды.

При использовании RTU-режима каждый байт сообщения является двухразрядным (двухсимвольным) шестнадцатеричным числом. Сообщение начинается с интервала тишины равного времени передачи 3.5 шестнадцатеричных символов (14 бит) при выбранной скорости передачи в сети. Сообщение передается непрерывным потоком. Если при передаче обнаруживается пауза продолжительностью более 1.5 шестнадцатеричных символов (6 бит), то считается, что сообщение содержит ошибку и должно быть отклонено принимающим модулем. Вслед за последним передаваемым байтом также следует интервал тишины продолжительностью не менее 3.5 символов. Новое сообщение должно начинаться после этого интервала. Эти величины пауз должны строго соблюдаться при скоростях ниже 19200 бит/с. При более высоких скоростях рекомендуется использовать фиксированные значения пауз, 1.75 мс и 750 мкс соответственно.

Стандарт предусматривает, что режим RTU в протоколе Modbus должен присутствовать обязательно, а режим ASCII является опционным. Пользователь может выбирать любой из них, но все модули, включенные в сеть Modbus, должны иметь один и тот же режим передачи. Далее рассматривается только протокол Modbus RTU.

Модель OSI протокола Modbus содержит три уровня: физический, канальный и прикладной.

При реализации физического интерфейса стандарт рекомендует использовать интерфейс RS-485 с двухпроводной линией передачи, но допускается применение четырехпроводной линии и интерфейса RS-232.

Modbus-устройство обязательно должно поддерживать скорости обмена 9600 и 19200 бит/с, из них 19200 бит/с устанавливается «по умолчанию». Допускаются также скорости 1200, 2400, 4800, …, 38400 бит/с, 65 кбит/с, 115 кбит/с. Скорость передачи должна выдерживаться в

передатчике с погрешностью не хуже 1 %, а приемник должен принимать данные при отклонении скорости передачи до 2 %.

На канальном уровне стандарт предполагает, что только одно ведущее устройство и до 247 ведомых могут быть объединены в промышленную сеть. Обмен данными всегда инициируется ведущим устройством. Ведомые устройства не могут обмениваться данными друг с другом. Поэтому в любой момент времени в сети Modbus может произойти только один акт обмена. Ведущее устройство может посылать запросы всем устройствам одновременно (широковещательный режим).

Первый байт сообщения (кадра) всегда является адресом ведомого устройства и принимает значения от 1 до 247 (адрес 0 - широковещательное сообщение), второй байт представляет собой функциональный код (FC). Некоторые FC неявно определяют длину сообщения, в то время как остальные позволяют её менять. Следующие байты в сообщении представляют собой поле данных произвольной длины *n*. Сообщение завершается 16-битным полем контроля циклическим избыточным кодом (CRC16). Формат сообщения в RTU режиме представлен на рисунке 1.

старт (интервал тишины)	*Адрес*	*FC*	*Данные*	*CRC16*	*конец (интервал тишины)*
T1-T2-T3-T4	8 бит	8 бит	n x 8 бит	16 бит	T1-T2-T3-T4

Рисунок 1 – Формат сообщения Modbus RTU

Поле «Адрес» всегда содержит только адрес ведомого устройства, даже в ответах на команду, посланную ведущим. Благодаря этому ведущее устройство знает, от какого модуля пришел ответ. Допустимый адрес передачи находится в диапазоне от 0 до 247. Каждому подчиненному устройству присваивается адрес в пределах от 1 до 247. Адрес 0 используется для широковещательной передачи, его распознает каждое устройство.

Поле «FC» говорит ведомому устройству о том, какое действие нужно выполнить. Диапазон значений поля - от 1 до 255.

Поле «Данные» может содержать произвольное количество байт. В нем может содержаться информация о параметрах, используемых в запросах ведущего устройства или ответах ведомого. Оно может содержать адреса регистров или выходов, их количество, счетчик передаваемых байтов данных. Поле данных может не существовать (иметь нулевую длину) для определенных сообщений (значений FC).

Поле «CRC16» содержит два байта контрольной суммы. Эта контрольная сумма является результатом вычисления циклического избыточного кода (Cyclical Redundancy Check – CRC), сделанного над содержанием всего сообщения. Стартовые, стоповые биты и бит паритета в вычислении CRC не участвуют. Алгоритм вычисления CRC16 приводится далее.

Передача символов сообщения осуществляется, начиная с младшего бита (рисунок 2).

С контролем четности

Старт	1	2	3	4	5	6	7	8	*Паритет*	*Стоп*

Без контроля четности

Старт	1	2	3	4	5	6	7	8	*Стоп*	*Стоп*

Рисунок 2 – Формат передачи символа в режиме RTU

Бит «Паритет» устанавливают равным 1, если количество двоичных единиц в байте нечетное, и равным 0, если оно четное. Такой паритет называют четным (*even parity*), а метод контроля называют *контролем четности*. Если при четном количестве двоичных единиц в байте бит паритета равен 1, то говорят, что паритет является нечетным (*odd parity*). Контроль четности может отсутствовать. В этом случае вместо бита паритета используют второй стоповый бит.

Стандартная сеть Modbus использует ещё один метод контроля ошибок - контроль сообщения в целом с помощью контрольной суммы. Контрольная сумма вычисляется передающим устройством и добавляется в конец сообщения. Принимающее устройство вычисляет контрольную сумму в процессе приема и сравнивает ее с полем CRC принятого сообщения. Если оба значения совпадают, считается, что сообщение не содержит ошибок. Самый распространенный способ определения контрольной суммы – разбить подлежащие передаче двоичные данные на блоки известной длины (от нескольких байт до нескольких тысяч байт). Далее содержимое каждого блока, рассматриваемое как двоичное число, делится на двоичное число длиной один или два байта. Контрольной суммой является остаток от деления, а двоичное число, используемое при делении, называется порождающим многочленом и обычно записывается в виде $x^n + x^{n-1} + \ldots + x^2 + x + 1$. Эта полиномиальная форма представляет собой способ записи. Порождающий многочлен на один бит длиннее результирующей контрольной суммы, и он начинается и заканчивается единицей. В нём указываются только степени, отличные от нуля (степень полинома равна длине контрольной суммы в битах). В сети Modbus используют стандартный порождающий многочлен в соответствии с рекомендацией МККТТ V.41 $x^{16} + x^{12} + x^5 + 1$, что соответствует двоичному шестнадцатиразрядному числу 1000100000010001 [7]. Пример подпрограммы на языке Си, вычисляющей CRC для этого порождающего многочлена приведен на рисунке 3. Вычисление осуществляется для массива байт *buf* с длиной *ln*.

```
unsigned crc_calc(unsigned char buf[], unsigned char ln)
unsigned buf_idx,crc;
unsigned char idx,flag;
```

```
{
  crc=65535;
  buf_idx=0;
  while(ln>0)
  {  crc^=buf[buf_idx];
    for(idx=0; idx<8; ++idx)
    {  flag=crc & 1;
      crc/=2;
      if (flag) crc^=0xA001;
    }
    ln--;
    buf_idx++;
  }
  return(crc);
}
```

Рисунок 3 – Листинг подпрограммы вычисления CRC16

Операции контроля паритета и CRC выполняются как главным устройством, так и подчиненным, а решение о возникновении ошибочной ситуации принимает только главное устройство.

Дополнительно пользователь может устанавливать продолжительность интервала таймаута, в течение которого главное устройство будет ожидать ответа от подчиненного. Если подчиненное устройство обнаружило ошибку передачи при проверке паритета или контрольной суммы, то оно не формирует ответ главному.

При запросе главного устройства к подчиненному возможна одна из четырех ситуаций:

- Если подчиненное устройство приняло запрос без коммуникационных ошибок и может при этом нормально распознать запрос, то оно возвращает нормальный ответ.
- Если подчиненное устройство не приняло запрос, ответ не возвращается. Главное устройство ожидает ответа на запрос в течение определенного таймаута.
- Если подчиненное устройство приняло запрос, но обнаружило коммуникационную ошибку (ошибку паритета, ошибку таймаута или несовпадение контрольных сумм), то ответ не возвращается. Главное устройство ожидает ответа на запрос в течение определенного таймаута.
- Если подчиненное устройство приняло запрос без коммуникационной ошибки, но не может выполнить затребованную функцию (например, чтение несуществующих регистров), то возвращается сообщение об ошибке и ее причинах.

Сообщение об ошибке имеет два поля, которые отличаются от полей нормального ответа. Первое из них – поле FC. В нормальном ответе

подчиненное устройство повторяет значение FC из запроса, при этом старший значащий бит FC установлен в 0. При возврате сообщения об ошибке подчиненное устройство устанавливает этот бит в 1. То есть по установленному старшему биту в поле FC главное устройство распознает сообщение об ошибке и может проанализировать поле данных сообщения.

Второе поле – поле данных: В нормальном ответе подчиненное устройство может возвращать данные или статистику в поле данных, то есть ту информацию, которая затребована в запросе. В случае ошибки подчиненное устройство возвращает код ошибки в поле данных.

На рисунке 4 показан пример запроса главного устройства и сообщения об ошибке подчиненного устройства. В данном примере главное устройство обращается к подчиненному устройству с адресом 10 (или в шестнадцатеричном виде - 0A). Выполняемая подчиненным устройством функция – «Запись выходного регистра», значение FC равно 06. В рассматриваемом запросе требуется записать в регистр с номером 002A данные 04FE.

ЗАПРОС

Имя поля	**Пример *(Hex)***
Адрес подчиненного	0A
Функция	06
Ст. байт номера регистра	00
Мл. байт номера регистра	2A
Ст. байт данных	04
Мл. байт данных	FE
Ст. байт CRC16	39
Мл. байт CRC16	2A

ОТВЕТНОЕ СООБЩЕНИЕ ОБ ОШИБКЕ

Адрес подчиненного	0A
Функция	86
Код ошибки	02
Ст. байт CRC16	63
Мл. байт CRC16	B2

Рисунок 4 – Пример запроса и сообщения об ошибке

Если указанный регистр не существует, подчиненное устройство возвращает сообщение об ошибке с кодом (02), соответствующее ошибке «Несуществующий адрес данных». Например, если в качестве подчиненного устройства используется программируемый логический контроллер (ПЛК) с 32 выходными регистрами, то этот код ошибки будет возвращаться при обращении к несуществующим регистрам. Список кодов возможных ошибок представлен в Приложении А.

Прикладной уровень Modbus RTU версии 1.1b основан на использовании запросов, содержащих коды функций (FC) [4]. Прикладной уровень независим от физического и канального уровней, в частности, он может использовать протоколы Ethernet TCP/IP (Modbus TCP/IP), Modbus Plus, интерфейсы RS-232, RS-422, RS-485, оптоволоконные каналы, радиоканалы и другие физические среды для передачи сигналов.

При использовании протокола прикладного уровня с различными протоколами транспортного и канального уровней основной блок Modbus-сообщения, включающий код функции и данные, сохраняется неизменным. Этот блок называется *Protocol Data Unit (PDU)*, он является элементом данных протокола. К блоку PDU при использовании его в различных промышленных сетях могут добавляться дополнительные поля, и тогда он называется *Application Data Unit (ADU)* – элемент данных приложения.

3 Пример коммуникации с использованием протокола Modbus

Возьмем в качестве примера типовую задачу автоматизированного управления процессом – управление прессом для пластика [7]. Технологический процесс показан на рисунке 5. Контейнер содержит расплавленный пластический материал, температура которого должна поддерживаться в пределах заданного диапазона. ПЛК постоянно считывает текущую температуру и рассчитывает тепло, необходимое для её поддержания на требуемом уровне. Тепло поступает от нагревательного элемента, управляемого ПЛК. Время его работы согласовано с количеством тепла, которое необходимо подвести.

Нижняя часть пресса состоит из цилиндра и поршня, выталкивающее определенное количество расплавленного пластика через насадку. Когда поршень находится в крайнем правом положении, цилиндр заполняется пластиком. Затем поршень быстро перемещается влево, выдавливая заданное по техпроцессу количество пластика. Положение поршня контролируется датчиком импульсов, который генерирует определенное число импульсов на каждый миллиметр перемещения. Объем выдавливаемого пластического материала определяется числом импульсов за время перемещения. Движение поршня прекращается при достижении заданного числа импульсов.

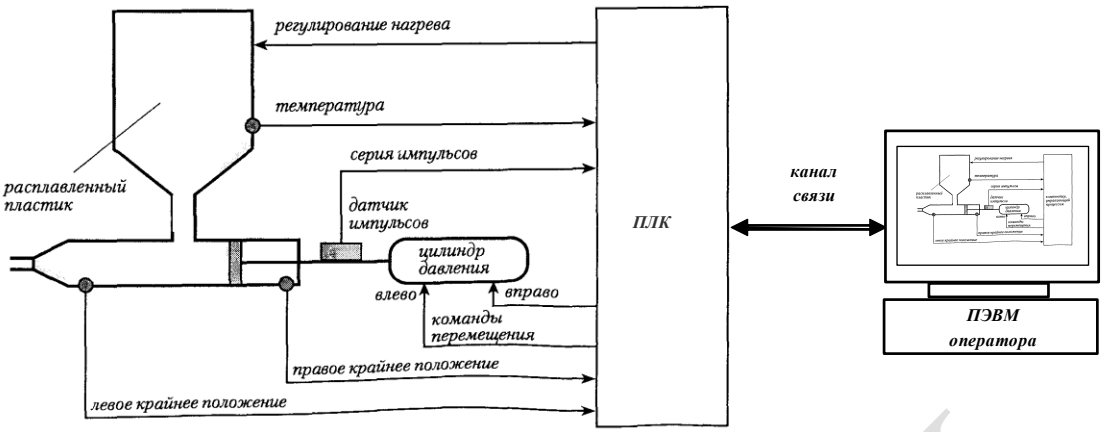

Рисунок 5 – Пример технологического процесса. Пресс для пластика

Чтобы обеспечить приемлемую производительность, температура пластика должна иметь заданное значение к тому моменту времени, когда поршень при движении вправо пройдет выходное отверстие контейнера. ПЛК регулирует температуру и движение поршня одновременно. Значение температуры поступает непрерывно в виде аналогового сигнала от датчика. Положение поршня рассчитывается по числу импульсов, поступающих на дискретный вход ПЛК. Ещё два датчика генерируют дискретные сигналы при достижении поршнем крайнего положения.

Рабочее место оператора технологического процесса оснащено ПЭВМ, на которой установлена специализированная программа, предназначенная для визуализации параметров техпроцесса на специальной мнемосхеме, сигнализации об аварийных событиях, регистрации в базе данных контрольных информационных параметров.

Обмен технологическими и измерительными данными между ПЭВМ и ПЛК осуществляется по последовательному интерфейсу RS-232 с использованием протокола Modbus. Ведущим устройством является ПЭВМ. Данные в памяти ПЛК, доступные для записи и чтения через интерфейс, показаны в таблице 1. Структура команд обмена данными по протоколу Modbus приведена в приложении Б.

Таблица 1 – Параметры ПЛК

№ регистра	Обозначение	Тип данных	Статус	Описание
Дискретные параметры				
0	$S_{лп}$	*bit*	*R*	Сигнал крайнего левого положения, бит 0
0	$S_{пп}$	*bit*	*R*	Сигнал крайнего правого положения, бит 1
0	$C_{лп}$	*bit*	*RW*	Команда «Движение влево», бит 2
0	$C_{пп}$	*bit*	*RW*	Команда «Движение вправо», бит 3
Аналоговые параметры				
1	***T***	*unsigned*	*R*	Температура пластика
2	***S***	*unsigned*	*R*	Перемещение поршня

3	t_θ	*unsigned*	*RW*	Время нагрева
Технологические параметры				
4	$T_{ТП}$	*unsigned*	*W*	Заданная температура пластика
5	$S_{ТП}$	*unsigned*	*W*	Заданное перемещение поршня
Конфигурационные параметры				
6	A_T**[3]**	*unsigned long [3]*	*RW*	Коэффициенты градуировочной характеристики датчика температуры
12	A_S	*unsigned long*	*RW*	Тарировочный коэффициент импульсного датчика перемещения

На ПЭВМ от ПЛК передаются измеренные значения дискретных и аналоговых параметров для визуализации и архивирования, а в ПЛК записываются уставки (технологические параметры) и градуировочные характеристики датчиков температуры и перемещения. Схема алгоритма обмена данными для ПЭВМ показана на рисунке 6, реализующая его программа – на рисунке 7.

Входными данными подпрограммы *transfer* (рисунок 6, блок 2) являются адрес ведомого устройства в сети Modbus (*addr*), функциональный код, указывающий ведомому устройству, какое действие ему нужно выполнить (*FC*), номер используемого регистра параметров, или номер первого в группе используемых регистров (*num*), массив с данными для регистров (*data*) и его длина (*ln*). Параметры *addr* и *FC* – однобайтовые беззнаковые целые, остальные параметры – беззнаковые целые длиной два байта. Формирование сообщения начинается с записи этих данных в байтовый массив *buf* (блок 3), причем для выделения старшего байта параметра *num* используется операция деления на 2^8=256, а для выделения младшего байта - операция логического умножения на 255 (называемой также «маскирование шестнадцатеричной константой 0xff»). Далее в цикле (блоки 4-6) аналогичным образом записываются элементы массива шестнадцатиразрядных данных *data*, число шагов цикла – *ln*. Для сформированного таким образом сообщения с помощью подпрограммы *crc_calc*, определяемой на рисунке 3, вычисляется контрольная сумма *crc* (блок 7), которая также фиксируется в массиве *buf* младшим байтом вперёд (блок 8). С помощью низкоуровневой подпрограммы *send_message* осуществляется передача сформированного сообщения *buf* в качестве запроса ведомому устройству (блок 9). Далее ведущее устройство дожидается ответного сообщения от ведомого, для этого используется низкоуровневая подпрограмма *rec_message* (блок 10). Определения подпрограмм *send_message* и *rec_message* зависят от используемых аппаратных средств и раскрываются далее.

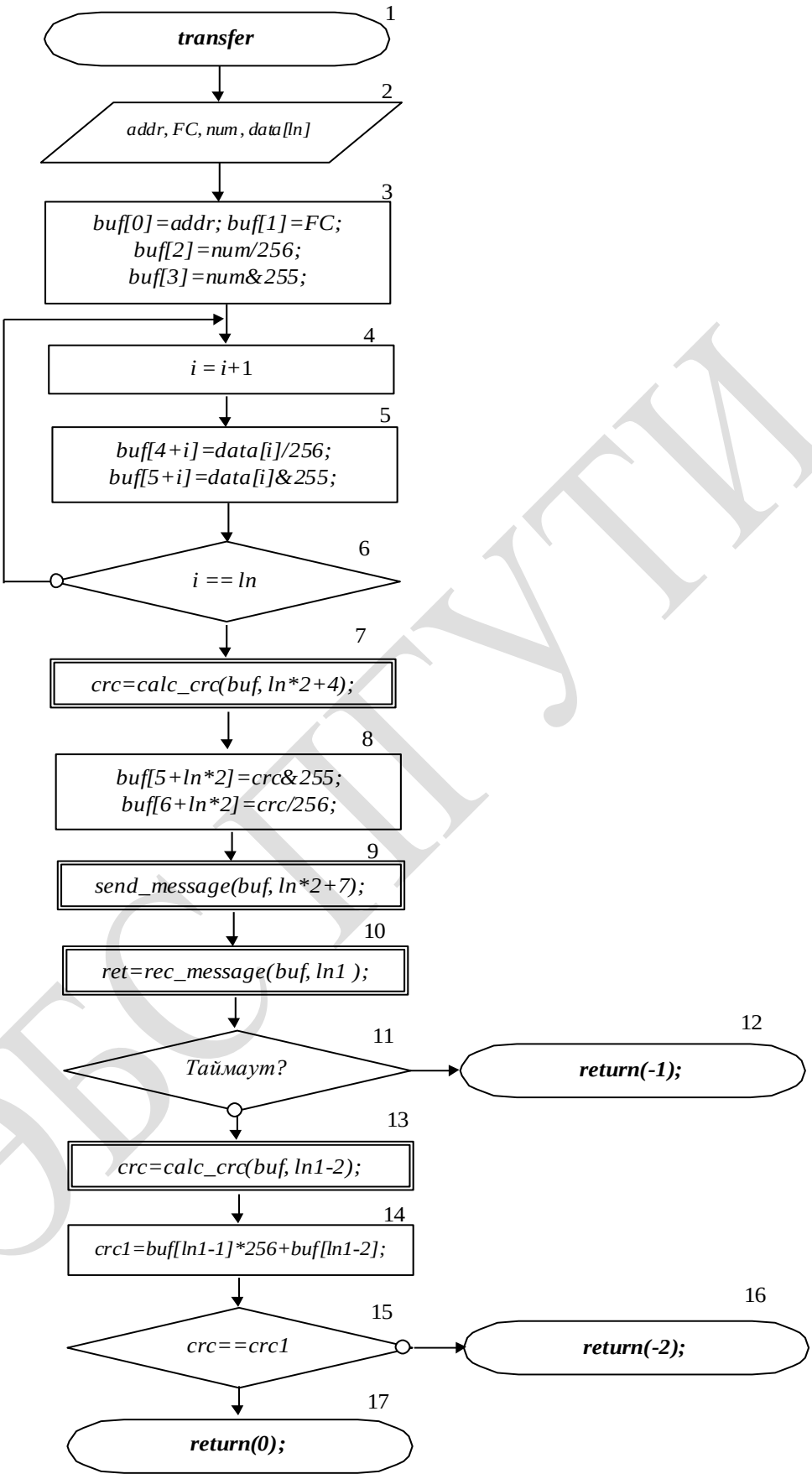

Рисунок 6 – Схема алгоритма обмена данными по протоколу Modbus для ведущего устройства

```
/************* Используемые функции
*************************************************/
void send_message                    // передача массива байт
(unsigned char message[],                  // массив
 unsigned n);                              // его длина

int rec_message          // прием массива байт с контролем ошибки таймаута
(unsigned char message[],                  // массив
 unsigned n);                              // его длина

volatile unsigned char buf[256];           // массив данных для запроса и ответа

/************* Подпрограмма запроса и получения ответного сообщения ********/
int transfer(unsigned char FC,       // значение FC
             unsigned char addr,     // адрес Slave-устройства
             unsigned num,                 // № используемого регистра
             unsigned ln,            // длина массива data[ ]
             unsigned data[ ])       // массив информационных данных для FC
{unsigned crc, crc1;
int ret=0;                           // код завершения функции
unsigned ln1=NFC;                          // длина сообщения-ответа от Slave
register int i;                            // переменная цикла

/******************* Формирование сообщения-запроса к Slave ***************************/
buf[0]=addr; buf[1]=FC;
buf[2]=num/256;              // старший байт num
buf[3]=num&0xff;             // младший байт num
for(i=0; i<ln; ++i           // заполняем специфичные поля для FC
{   buf[4+i]=data[i]/256;
    buf[5+i]=data[i]&255;    }
crc=crc_calc(buf, ln*2+4);           // вычисление CRC16 запроса
buf[ln*2+5]=crc&0xff;        // младший байт CRC16
buf[ln*2+6]=crc/256;         // старший байт CRC16
/*****************************************************************************
*******************************/

send_message(buf,ln*2+7);            // посылка сообщения Slave-устройству
ret=rec_message(buf, ln1);           // прием ответного сообщения от Slave
```

```
if(ret<0) return ret;                    // выход из подпрограммы по ошибке приема
crc=crc_calc(buf, ln1-2);                // вычисление CRC16 ответа
crc1=buf[ln1-2]+buf[ln1-1]*256;          // формирование CRC16 из ответа Slave
ret=(crc1!=crc)?(-2):(0);                // контроль ошибки CRC16
return ret;
}
```

Рисунок 7 – Подпрограмма обмена данными по протоколу Modbus для ведущего устройства

Если ответное сообщение принято без ошибки таймаута (*ret*=0), то вычисляется его контрольная сумма и производится сравнение с контрольной суммой, которая была записана в массив сообщения ведомым устройством (блок 13-15). При несовпадении контрольных сумм подпрограммой *transfe*r возвращается код ошибки -2 (блок 16), при нормальном исходе *transfer* возвращает нуль (блок 17). Формирование шестнадцатиразрядной переменной *crc1* из байтов массива *buf* осуществляется в блоке 14.

Подпрограмма обмена данными по протоколу Modbus *transfer* является примером проблемно-ориентированной процедуры, не зависящей от используемой аппаратной реализации. Она функционирует на процессоре ведущего устройства. Очевидно, что для реализации двустороннего обмена данными подпрограмма, аналогичная *transfer*, должна существовать (и функционировать при обмене) и для ведомого устройства. Следует также отметить, что в подпрограмме обмена данными для ведомого устройства работа с массивами сообщений происходит в обратном порядке: в начале осуществляется прием сообщения, потом – проверка контрольной суммы, разбор данных сообщения, формирование ответного сообщения в соответствии с *FC*, вычисление и запись контрольной суммы ответного сообщения, передача ведущему устройству. Низкоуровневые подпрограммы *send_message* и *rec_message* зависят от используемого оборудования, как на стороне ведущего устройства, так и на стороне ведомого и поэтому, в общем случае, могут быть различны в своих реализациях.

В лабораторной работе используется ПЛК с архитектурой ПЭВМ без операционной системы, на ПЭВМ оператора техпроцесса эмулируется MS DOS. В качестве физического интерфейса используется RS-232[1]. Это предоставляет нам возможность изучить программную модель последовательного интерфейса ПЭВМ и средства низкоуровневого программирования при отсутствии операционной системы. В качестве языка программирования используется Borland C.

[1] . Используется нуль-модемное соединение (разводка разъема DB25 для такого типа соединения показана в Приложении В).

4 Программная модель последовательного интерфейса ПЭВМ

Базовые адреса последовательных портов ПЭВМ располагаются в области данных BIOS, начиная с адреса *0040:0000*. Первый порт ***com1*** имеет базовый адрес *0x3F8* и занимает диапазон адресов от *0x3F8* до *0x3FF*, ***com2*** – *0x2F8* и диапазон *0x2F8...0x2FF*, ***com3*** – *0x3E8* и диапазон *0x3E8...0x3EF*, ***com4*** – *0x2E8* и диапазон *0x2E8...0x2EF* [8]. Порты ***com1*** и ***com3*** в стандартной конфигурации используют вектор прерывания с номером 12 и линию ***Irq4*** контроллера прерываний, ***com2*** и ***com4*** – вектор 11 и линию ***Irq3***. Для краткости далее излагается программирование порта ***com1***, но все это применимо и для других портов.

Программная модель контроллера последовательного интерфейса ПЭВМ содержит 10 программируемых однобайтовых регистров. Большинство из них используется для инициализации порта. Доступ к этим 10 регистрам осуществляется через 7 адресов портов. Для расширения адресации используется старший бит регистра контроля линии. В таблице 2 показана регистровая модель порта ***com1***.

Таблица 2 – Регистровая модель последовательного порта

Адрес регистра	Чтение/запись (*R/W*)	Назначение регистра	Примечание
0x3F8	*W*	Регистр хранения передатчика	бит 7=0 в *0x3FB*
0x3F8	*R*	Регистр данных приемника	бит 7=0 в *0x3FB*
0x3F8	*W*	Делитель скорости обмена (мл.)	бит 7=1 в *0x3FB*
0x3F9	*W*	Делитель скорости обмена (ст.)	бит 7=1 в *0x3FB*
0x3F9	*W*	Регистр разрешения прерывания	бит 7=0 в *0x3FB*
0x3FA	*R*	Регистр идентификации прерывания	
0x3FB	*R/W*	Регистр управления линией	
0x3FC	*R/W*	Регистр управления модемом	
0x3FD	*W*	Регистр статуса линии	
0x3FE	*W*	Регистр статуса модема	

При отсутствии модема из 10 регистров необходимы только 6. Регистр хранения передатчика содержит байт данных, который будет послан, а регистр данных приемника – последний полученный байт данных. Регистры управления и статуса линии инициализируют и управляют линией связи, используя скорость обмена, содержащуюся в двух регистрах делителя скорости обмена. Из оставшихся 4 регистров регистры управления и статуса модема необходимы только для связи через модем, а два регистра, связанные с прерыванием – только в процедурах управления прерыванием.

В начале рассматривается программирование последовательного порта в режиме опроса готовности (без использования прерываний).

4.1 Инициализация последовательного порта

При инициализации последовательного порта устанавливаются следующие параметры передачи: длина слова, число стоповых бит, установка четности и скорость обмена. Инициализация осуществляется с использованием 4 регистров (регистры делителя скорости обмена, регистр контроля линии и регистр разрешения прерывания).

На первом шаге инициализации устанавливается скорость обмена с использованием регистров делителя. Делитель скорости обмена – это число, на которое надо разделить частоту системного генератора (115200 Гц), чтобы получить желаемую скорость обмена. Например, для скорости обмена 110 бод делитель равен 1040 (*0xC00*), а для 9600 бод – 12 (*0xC*). Старший байт делителя задается в регистре по адресу *0x3F9*, а младший – в *0x3F8*. При этом бит 7 регистра управления линией (*0x3FB*) должен быть установлен в 1, иначе значения будут записаны в другие регистры. Удобнее устанавливать регистры скорости обмена первыми, так как они единственные, которые требуют установки бита 7 в регистре управления линией. На втором шаге устанавливаются остальные параметры передачи заданием соответствующих бит в регистре управления линией (*0x3FB*). Пример задания скорости обмена 110 бод с использованием функций *outportb()* представлен на рисунке 8.

```
#define Com1Base 0x3F8              // базовый адрес com1

void outportb(unsigned addr,        // адрес регистра
              unsigned char data);  // данные для записи
...
outportb(Com1Base+3, 0x80);         // установка бита 7
outportb(Com1Base, 0);              // младший байт делителя
outportb(Com1Base+1, 0xC);          // старший байт
```

Рисунок 8 – Фрагмент программы, иллюстрирующий задание скорости обмена

Формат регистра управления линией показан на рисунке 9. Биты 0÷4 используются непосредственно для задания параметров передачи, а биты 5÷7 используются в специальных случаях. При установке бита 5 («фиксация четности») бит паритета в передаваемом байте всегда принимает значение 0 при установленном контроле четности (биты 3-4 равны 11) или 1 при контроле нечетности (биты 3-4 равны 01). При установленном бите 6 выводится строка нулей в качестве сигнала перерыва определенной станции (BREAK). Назначение бита 7 было рассмотрено ранее.

0 бит	1 бит	2 бит	3 бит	4 бит	5 бит	6 бит	7 бит
длина символа (бит) 00 – 5 бит 01 – 6 бит 10 – 7 бит 11 – 8 бит		***кол-во стоповых бит*** 0 – 1 бит 1 – 1.5 бита, если длина символа равна 5, и 2 бита, иначе	***паритет*** x0 – нет контроля 01 – проверка на нечетность 11 – проверка на четность		***фиксация четности***	***уста-новка пере-рыва***	***смена адреса***

Рисунок 9 – Формат регистра управления линией

Пусть используются следующие параметры передачи информации: длина слова – 5 бит, 1 стоповый бит, проверка на четность. В двоичном виде это соответствует управляющему слову 00011000, или в шестнадцатеричном виде – *0x18*. Задание параметров передачи фактически сводится к вызову функции *outportb(Com1Base+3, 0x18)*.

На последнем шаге инициализации необходимо запретить или разрешить прерывания в регистре разрешения прерываний (*0x3F9*). Если прерывания не используются, можно просто поместить в этот регистр нуль.

4.2 Передача данных

Передача байта данных осуществляется записью его в регистр хранения передатчика (*0x3F8*), предварительно убедившись в том, что передача предыдущего символа завершена (регистр свободен). Признаком того, что регистр передатчика свободен, является наличие 1 в бите 5 регистра статуса линии (*0x3FD*).

Каждый бит этого регистра сигнализирует об определенном событии (ошибке) обмена[2]. В таблице 3 указаны эти события.

Таблица 3 – Соответствие битов регистра статуса линии событиям

Бит	Назначение
0	Данные получены и готовы для чтения, после завершения операции чтения бит сбрасывается.
1	Ошибка переполнения, возникающая в ситуации, когда был принят новый байт данных, а предыдущий не считан программой (в результате предыдущий байт потерян). Сбрасывается при чтении регистра статуса.
2	Ошибка паритета. Сбрасывается при чтении регистра статуса.
3	Ошибка синхронизации (например, отсутствие стопового бита). Сбрасывается при чтении регистра статуса.
4	Обнаружен запрос BREAK. Сбрасывается при чтении регистра

[2] Возникновению события соответствует 1.

	статуса.
5	Регистр хранения передатчика пуст и в него можно записать новый байт для передачи. Сбрасывается после завершения операции.
6	Регистр сдвига передатчика пуст. Этот регистр получает данные из регистра хранения и преобразует их в последовательный код для передачи (если этот бит равен 1, то микросхема UART может принять очередной символ от процессора)
7	Тайм-аут (устройство не связано с компьютером)

Пример передачи байта данных *byte* с использованием функций *outportb()* и *inportb()* приведен на рисунке 10. В цикле опроса готовности с помощью операции маскирования шестнадцатеричной константой 0x20 осуществляется проверка пятого бита регистра статуса линии (*Com1Base+5*). Как только в этом бите появляется 1, осуществляется вывод байта *byte* в регистр хранения передатчика *Com1Base* (адреса регистров в соответствии с регистровой моделью таблицы 2).

```
unsigned char inportb(unsigned addr);        // адрес регистра
...
while((inportb(Com1Base+5)&0x20)==0);        // цикл опроса готовности
outportb(Com1Base, byte);                    // передача байта
```

Рисунок 10 – Фрагмент программы, иллюстрирующий передачу байта данных

4.3 Прием данных

Аналогично передаче данных перед вводом байта из регистра данных приемника (*0x3F8*) необходимо убедиться в том, что бит 0 регистра статуса линии (*0x3FD*) установлен в 1. Это означает, что байт принят и находится в буферном регистре приемника. Пример приема байта данных *byte* иллюстрирует фрагмент программы, приведенный на рисунке 11.

```
unsigned long ct=0x80000000L;          // счетчик циклов= уставка
// цикл опроса готовности с проверкой таймаута
while( ((inportb(Com1Base+5)&1)==0) && (ct) ) --ct;
if(ct==0L) return(ErrTimeout);         // выход с кодом ошибки
byte=intportb(Com1Base);                    // прием байта
return(byte);                          // выход без ошибки
```

Рисунок 11 – Фрагмент программы для иллюстрации приема байта данных

Для исключения зацикливания программы при приеме байта используют программный контроль таймаута – подсчет времени выполнения функции и прерывание ее при превышении допустимого значения. В примере, приведенном на рисунке 11, иллюстрируется проверка

таймаута с использованием счетчика циклов (такая проверка обязательна, если отсутствует аппаратная поддержка).

4.4 Прием и передача данных с использованием прерываний

Так как процесс последовательной передачи данных протекает достаточно медленно, имеет смысл выполнять его в фоновом режиме, используя прерывания по окончании передачи или приема байта данных.

Для управления этими прерываниями используется регистр разрешения прерываний (*0x3F9*), формат которого приведен в таблице 4.

Таблица 4 – Биты разрешения прерываний

Бит	Назначение
0	Разрешение прерывания при готовности принимаемых данных.
1	Разрешение прерывания после передачи байта.
2	Разрешение прерывания по обнаружению состояния BREAK или по ошибке
3	Разрешение прерывания по изменению состояния входных линий на разъеме RS-232-C (CTS, DSR, RI, DCD).
4-7	Не используются.

Для разрешения прерываний необходимо установить в 1 биты этого регистра, соответствующие тем прерываниям, которые будут обрабатываться.

Таблица 5 – Соответствие битов 0 и 1 источнику прерываний

Значение битов 1 и 2	Источник (причина) прерываний
0 0	Изменение состояния модема. Устанавливается при изменении состояния входных линий CTS, RI, DCD, DSR. Сбрасывается после чтения регистра статуса модема.
0 1	Буфер передатчика пуст. Сбрасывается при записи новых данных в регистр хранения передатчика.
1 0	Данные приняты и доступны для чтения. Сбрасывается после чтения данных из регистра данных приемника.
1 1	Ошибка передачи (переполнение приемника, ошибка паритета, ошибка формата данных) или обнаружение состояния BREAK. Сбрасывается после чтения регистра статуса линии.

Когда произошло прерывание, подпрограмма обработки прерывания должна определить источник прерывания, прочитав содержимое регистра идентификации прерывания (*0x3FA*). Источник прерывания кодируется

битами 1, 2 регистра (таблица 5). Если в младшем бите регистра установлена 1, то нет прерываний, ожидающих обслуживания[3].

Пример подпрограммы обработки прерывания по приему байта данных[4], формирующий массив данных *buffer*, приведен на рисунке 12.

```
unsigned char buffer[8];        // буфер для приема 8 байт
int ctb=8;                      // задание количества байт
int ib;                         // индекс в массиве buffer[ ]
...
void interrupt com_irq_handler(void)
{    if(ctb)                    // проверка необходимости приема
   { ctb--;                     // цикл приема очередного байта
     buffer[ib++]=inportb(Com1Base);  } // с размещением в буфере
   outportb(Com1Base+1,1);      // разрешение следующего прерывания
   outportb(0x20,0x20);         // «автоматический конец прерываний»
}
```

Рисунок 12 –Подпрограмма обработки прерывания по приему байта данных

Подпрограмма com_irq_handler фактически определяет тело цикла приема байта сообщения, запускаемое при каждом прерывании. Цикл выполняется заданное число раз (в этом примере – 8 раз). Модификатор *interrupt* сообщает компилятору Borland C о необходимости сохранения всех регистров в стеке и возврате из подпрограммы по ассемблерной команде iret (с выталкиванием из стека счетчика команд и слова состояния).

5 Порядок выполнения работы

1. Изучите программирование низкоуровневых примитивов последовательной связи (раздел 4). Напишите подпрограммы инициализации (параметры обмена по индивидуальному заданию), приема и передачи байта, приема и передачи массива байт.

2. Используя эти подпрограммы, реализуйте обмен данных между двумя компьютерами. Для этого необходимо написать две программы – программу передающей стороны и программу принимающей. Трансляция программ осуществляется компилятором Borland C помощью командного файла 1.bat. Проведите отладку обоих программ на различных тестах.

3. Изучите протокол Modbus (раздел 2 и Приложения А, Б). Используя ранее разработанные примитивы, напишите две программы – главного

[3] Может случиться так, что одновременно произойдет несколько прерываний. В этом случае бит 0 регистра идентификации прерывания тоже будет установлен в 1. Если такая ситуация имеет место, то перед завершением обработки прерывания необходимо снова прочитать регистр идентификации прерывания и обработать следующее прерывание. Так следует поступать до тех пор, пока бит 0 регистра идентификации прерывания не станет равным нулю.

[4] Предполагается, что разрешен единственный источник прерываний по приему байта данных.

компьютера и подчиненного компьютера, осуществляющие обмен «запрос-ответ» для функции FC8. Проведите отладку транслированных программ.

4. Доработайте эти программы в соответствии с индивидуальным заданием, то есть реализуйте обмен «запрос-ответ» для указанной в задании функции. Проведите отладку транслированных программ на различных тестах.

Лабораторная работа засчитывается, если все разработанные программы отлажены и функционируют в соответствии с индивидуальным заданием.

6 Контрольные вопросы[5]

1. Перечислите отличительные признаки работы с периферийным устройством в режиме прерывания.

2. Какие алгоритмические действия выполняются при инициализации последовательного интерфейса ПЭВМ IBM PC AT?

3. Сформулируйте алгоритм передачи массива данных по последовательному интерфейсу с использованием прерываний и в режиме опроса готовности.

4. В чем состоит алгоритм приема массива данных по последовательному интерфейсу с использованием прерываний и в режиме опроса готовности?

5. Какие возможны типы последовательных интерфейсов в зависимости от используемого механизма синхронизации данных при передаче. К какому типу относится последовательный порт IBM PC AT?

6. К каким уровням эталонной модели взаимодействия открытых систем относятся интерфейс RS-232, протокол Modbus?

7. Назовите методы контроля ошибок, возникающих при обмене под действием помех в канале связи. Какие методы контроля использовались при выполнении данной работы?

8*. Какими способами можно контролировать ошибку таймаута при приеме массива байт? Оцените область применения каждого из способов.

9*. Как вычисляется интервал тишины (3.5 байта) для заданной скорости обмена?

7 Индивидуальные задания

1. Параметры обмена: скорость передачи 19200 бод, длина слова 8 бит, 1 стоповый бит, контроль нечетности. Реализовать программы, соответствующие ведущему и ведомому узлу, для функций FC8 «Тест», FC4 «Чтение группы регистров».

2. Параметры обмена: скорость передачи 4800 бод, длина слова 8 бит, 2 стоповых бита, без контроля паритета. Реализовать программы, соответствующие ведущему и ведомому узлу, для функций FC8 «Тест», FC16 «Запись группы регистров».

[5] Контрольные вопросы, отмеченные ***,** имеют повышенную сложность и используются в индивидуальном порядке.

3. Параметры обмена: скорость передачи 9600 бод, длина слова 8 бит, 1 стоповый бит, контроль четности. Реализовать программы, соответствующие ведущему и ведомому узлу, для функций FC8 «Тест», FC6 «Запись выходного регистра».

8 Литература

1. Денисенко В.В. Компьютерное управление технологическим процессом, экспериментом, оборудованием. – М.: Горячая линия-Телеком, 2009. – 608 с.
2. Танненбаум Э. Компьютерные сети. – СПб.: Питер, 2003. – 992 с.
3. Modbus over serial line specification and implementation guide. V1.02. – www.Modbus-IDA.org. Dec. 20. 2006. – 44 p.
4. Modbus application protocol specification. V1.1b. – www.Modbus-IDA.org. Dec. 28. 2006. – 51 p.
5. Modicon Modbus Protocol Reference Guide. PI-MBUS-300 Rev.J.- MODICON. Inc., Industrial Automation Systems. June 1996. – 121 p.
6. Modbus messaging on TCP/IP implementation guide. V1.0b. - www.Modbus-IDA.org. Oct. 24. 2006. – 46 p.
7. Оллсон Г., Пиани Дж. Цифровые системы автоматизации и управления.- С.Пб: Невский диалект, 2001. – 556 с.
8. Фролов А.В., Фролов Г.В. Программирование модемов. – М.: Диалог-МИФИ, 1993 г. – 236 с.

Приложение А. Список кодов ошибок

Код	Название	Описание
1	ILLEGAL FUNCTION	Принятый код функции не может быть обработан на подчиненном устройстве.
2	ILLEGAL DATA ADDRESS	Адрес данных, указанный в запросе, не доступен данному подчиненному устройству.
3	ILLEGAL DATA VALUE	Значение, содержащееся в поле данных запроса, недопустимо для подчиненного устройства.
4	SLAVE DEVICE FAILURE	Во время выполнения затребованного действия подчиненным устройством произошла невосстанавливаемая ошибка.
5	ACKNOWLEDGE	Обработка запроса требует много времени. Этот ответ предохраняет главное устройство от генерации ошибки таймаута. Главное устройство может выдать команду Poll Program Complete для обнаружения завершения обработки запроса.
6	SLAVE DEVICE BUSY	Подчиненное устройство занято обработкой команды. Главное устройство должно повторить сообщение позже, когда подчиненное устройство освободится.
7	NEGATIVE ACKNOWLEDGE	Подчиненное устройство не может выполнить функцию, принятую в запросе. Этот код возвращается для неудачного программного запроса с кодами функций 13 или 14. При возникновении такой ошибки главное устройство должно запросить диагностическую информацию у подчиненного.
8	MEMORY PARITY ERROR	При чтении подчиненным устройством расширенной памяти обнаружена ошибка паритета. Главное устройство может повторить запрос, но обычно в таких случаях требуется ремонт.

Приложение Б. Примеры форматов сообщений ModBus в RTU-режиме

FC 04 - Чтение входных регистров

Формат запроса

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (4)
3	Ст. байт номера 1-го регистра
4	Мл. байт номера 1-го регистра
5	Ст. байт числа регистров (0)
6	Мл. байт числа регистров (**n**)
7	Ст. байт CRC16
8	Мл. байт CRC16

Формат ответа

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (4)
3	Счетчик байт (2**n**)
4	Ст. байт данных 1-го регистра
5	Мл. байт данных 1-го регистра
	…
2**n**+2	Ст. байт данных **n**-го регистра
2**n**+3	Мл. байт данных **n**-го регистра
2**n**+4	Ст. байт CRC16
2**n**+5	Мл. байт CRC16

FC 08 - Тест соединения

Формат запроса

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (8)
3	Ст. байт диагностического кода (0)
4	Мл. байт диагностического кода (0)
5	Ст. байт данных
6	Мл. байт данных
7	Ст. байт CRC16
8	Мл. байт CRC16

Формат ответа

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (8)
3	Ст. байт диагностического кода (0)
4	Мл. байт диагностического кода (0)
5	Ст. байт данных
6	Мл. байт данных
7	Ст. байт CRC16
8	Мл. байт CRC16

FC 6 - Запись выходного регистра

Формат запроса

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (6)
3	Ст. байт номера регистра
4	Мл. байт номера регистра
5	Ст. байт данных регистра
6	Мл. байт данных регистра
7	Ст. байт CRC16
8	Мл. байт CRC16

Формат ответа

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (6)
3	Ст. байт номера регистра
4	Мл. байт номера регистра
5	Ст. байт данных регистра
6	Мл. байт данных регистра
7	Ст. байт CRC16
8	Мл. байт CRC16

FC 16 - Запись выходных регистров

Формат запроса

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (0x10)
3	Ст. байт номера 1-го регистра
4	Мл. байт номера 1-го регистра
5	Ст. байт числа регистров (0)
6	Мл. байт числа регистров (**n**)
7	Счетчик байт (2**n**=2..0xFA)
8	Ст. байт данных 1-го регистра
9	Мл. байт данных 1-го регистра
	…
2**n**+6	Ст. байт данных **n**-го регистра
2**n**+7	Мл. байт данных **n**-го регистра
2**n**+8	Ст. байт CRC16
2**n**+9	Мл. байт CRC16

Формат ответа

№ байта	Содержимое
1	Адрес устройства Slave (1..0xF7)
2	Функциональный код (0x10)
3	Ст. байт номера 1-го регистра
4	Мл. байт номера 1-го регистра
5	Ст. байт числа регистров (0)
6	Мл. байт числа регистров (**n**)
7	Ст. байт CRC16
8	Мл. байт CRC16

Приложение В. Разводка разъема последовательной передачи данных DB25 при нуль-модемном соединении

№ контакта	Назначение	In/Out	Признак использования
1	Защитное заземление (Frame Ground, FG)		
2	Передаваемые данные (Transmitted Data, TD)	Out	+
3	Принимаемые данные (Received Data, RD)	In	+
4	Запрос для передачи (Request to send, RTS)	Out	
5	Сброс для передачи (Clear to Send, CTS)	In	
6	Готовность данных (Data Set Ready, DSR)	In	
7	Сигнальное заземление (Signal Ground, SG)		+
8	Детектор принимаемого с линии сигнала (Data Carrier Detect, DCD)	In	
9	*Не используются*		
10	*Не используются*		
11	*Не используются*		
12	*Не используются*		
13	*Не используются*		
14	*Не используются*		
15	*Не используются*		
16	*Не используются*		
17	*Не используются*		
18	*Не используются*		
19	*Не используются*		
20	Готовность выходных данных (Data Terminal Ready, DTR)	Out	
21	*Не используются*		
22	Индикатор вызова (Ring Indicator, RI)	In	
23	*Не используются*		
24	*Не используются*		
25	*Не используются*		