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 «Информационно-измерительная техника»

РАБОТА ПРОВЕРЕНА

Рецензент, преподаватель

ГБПОУ «ЮГК»

_____/ Е.В. Фостаковская /

«_____» _____2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.т.н., проф.

_____/ А.Л. Шестаков /

«_____» _____2020г.

Разработка графического интерфейса и его функционального наполнения для датчика температуры

(наименование темы работы (проекта)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ – 12.03.01.2020.308-602.ВКР

(код направления/специальности, год, номер студенческого)

Руководитель, проф. кафедры ИнИТ

_____/ В.А. Ларионов /

«_____» _____2020 г.

Автор

студент группы КЭ – 225

_____/ П. С. Афанасьев /

«_____» _____2020 г.

Нормоконтролер, доцент кафедры ИнИТ

_____/ А.С. Волосников /

«_____» _____2020 г.

Челябинск 2020

АННОТАЦИЯ

Афанасьев П. С. Разработка графического интерфейса и его функционального наполнения для датчика температуры. – Челябинск: ЮУрГУ, КЭ-225, 2020, 134 с, 32 ил., библ. список – 34 наим., 1 прил.

Измерения температуры в технических системах являются составной частью процессов их разработки, испытаний, производства, эксплуатации и утилизации. Любое средство измерений имеет тенденцию к деградации со временем его метрологических характеристик. В настоящее время большинство предприятий по производству средств измерений стараются увеличить межповерочный интервал производимых ими устройств. Одним из способов решения данной проблемы является внедрение в датчик функции метрологического самоконтроля.

В результате разработки нового датчика температуры с метрологической функцией самоконтроля, появилась необходимость создания графического интерфейса и его функционального наполнения, который позволит корректно взаимодействовать с датчиками.

Разработка компьютерного программного обеспечения, выполняемая в данной работе, будет включать в себя разработку программы и алгоритма работы графического интерфейса для датчика температуры с функцией метрологического самоконтроля.

В ходе работы было выполнено тестирование программы, получены результаты и сделаны соответствующие выводы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Измерения температуры в технических системах являются составной частью процессов их разработки, испытаний, производства, эксплуатации и утилизации. Любое средство измерений имеет тенденцию к деградации со временем его метрологических характеристик. Эффективность применения современной техники во многих случаях непосредственно связана с качеством используемой термометрической информации. Для подтверждения соответствия качества измеряемых данных, проводят процедуру поверки, которая может подтвердить метрологические неисправности средства измерения. Метрологическая неисправность – это состояние средства измерения, при котором все его нормирующие метрологические характеристики соответствуют установленным требованиям [1]. Средство измерения должно работать в течение всего межповерочного интервала. Межповерочный (межкалибровочный) интервал – это промежуток времени или наработка между двумя последовательными поверками (калибровками) [1].

В настоящее время большинство предприятий по производству средств измерений стараются увеличить межповерочный интервал производимых ими устройств. Одним из способов решения данной проблемы является внедрение в датчик функции метрологического самоконтроля.

Датчик – это конструктивно обособленное устройство, содержащее один или несколько первичных измерительных преобразователей [2]. Метрологическим самоконтролем датчика называют автоматическую проверку метрологических исправностей датчика в процессе его эксплуатации, которая осуществляется с использованием принятого опорного значения, формируемого с помощью встроенного в датчик средства или выделенного дополнительного параметра выходного сигнала [2]. Интеллектуальный датчик – адаптивный датчик с функцией метрологического самоконтроля. Поэтому датчик с функцией метрологического самоконтроля можно назвать интеллектуальным датчиком измерения.

Одной из важных задач при создании датчика с функцией метрологического самоконтроля является разработка компьютерного программного обеспечения, для взаимодействия датчика с персональным компьютером (ПК). Компьютерное программное обеспечение (ПО) - это совокупность программ и соответствующей документации, позволяющая использовать вычислительную технику для обработки данных и решения различных задач [3]. Особенности поверочной деятельности определяют специфические задачи ПО для метрологии: учет характеристик эталонов, обеспечение прослеживаемости к эталонам, учет неопределенности проводимых при поверке измерений (ПО должно иметь встроенный математический аппарат для вычисления неопределенностей при измерениях), обеспечение различных форм представления информации (на базе одного массива информации о приборах формировать различные отчеты (протоколы, свидетельства, графики поверки и пр.)), архивное хранение данных о поверке [4].

Разработка компьютерного программного обеспечения, выполняемая в данной работе, будет включать в себя разработку программы и алгоритма работы графического интерфейса для датчика температуры с функцией метрологического самоконтроля.

Направление разработки новых датчиков температуры является актуальным по-нескольким причинами: большое количество температурных датчиков, подлежащих поверке, вызывают огромные траты денег на их проведение; выведение из строя датчика температуры может повлечь за собой катастрофические последствия, поэтому необходимо расширять возможности датчиков и повышать их надежность.

Научной новизной представленной выпускной работы является создание нового алгоритма функционирования датчика температуры с персональным компьютером, на основе графического интерфейса пользователя и его функционального наполнения, с использованием программирования.

Практической ценностью выпускной работы является разработка графического интерфейса, который даёт возможность пользователю взаимодействовать с датчиком температуры.

Целью выпускной квалификационной работы является обеспечение корректного взаимодействия датчиком температуры с функцией метрологического самоконтроля через персональный компьютер путём создания программного обеспечения (графического интерфейса и его функционального наполнения).

Задачи выпускной квалификационной работы:

1) обзор и выбор программ компьютера для создания архитектуры графического интерфейса связи с датчиком температуры с функцией метрологического самоконтроля;
2) разработка программы и алгоритма работы графического интерфейса для датчика температуры с метрологической функцией самоконтроля;
3) отладка программы с использованием рабочих образцов датчика;
4) проведение корректировки программы по результатам испытаний.

1 ОБЗОР И ВЫБОР ПРОГРАММ КОМПЬЮТЕРА ДЛЯ СОЗДАНИЯ АРХИТЕКТУРЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ДЛЯ СВЯЗИ С ДАТЧИКОМ ТЕМПЕРАТУРЫ С ФУНКЦИЕЙ САМОДИАГНОСТИКИ

1.1 Обзор программ для разработки графических интерфейсов

Развитие информационных технологий и рост числа пользователей компьютеров предъявляют новые требования к пользовательским интерфейсам. Перед разработчиками программного обеспечения стоит непростая задача обеспечить низкий порог вхождения для широкого круга не обладающих достаточной квалификацией и не прошедших специальную подготовку в использовании ЭВМ лиц за счет упрощения средств диалога с машиной, использования интуитивно понятных графических метафор, унификации интерфейсов. Таким образом, овладение средствами разработки современных графических интерфейсов пользователя является необходимым элементом подготовки разработчика прикладного программного обеспечения.

Графический пользовательский интерфейс - это форма пользовательского интерфейса, которая позволяет пользователям взаимодействовать с электронными устройствами с помощью графических значков и звукового индикатора, такого как первичная запись, вместо текстовых пользовательских интерфейсов, вводимых меток команд или текстовой навигации.

Разработка визуальной композиции и временного поведения GUI является важной частью программирования прикладных программ в области взаимодействия человека с компьютером. Его цель - повысить эффективность и простоту использования для базового логического дизайна хранимой программы, дисциплины дизайна, называемой юзабилити. Методы ориентированного на пользователя дизайна используются для обеспечения того, чтобы визуальный язык, представленный в проекте, был хорошо адаптирован к задачам.

Принципы построения графического пользовательского интерфейса соответствуют шаблону программного обеспечения «модель - представление - контроллер», который отделяет внутренние представления информации от способа представления информации пользователю, что приводит к платформе, на которой пользователям показывают, какие функции возможны, а не требуют ввод командных кодов. Пользователи взаимодействуют с информацией, манипулируя визуальными виджетами, которые предназначены для реагирования в соответствии с типом данных, которые они содержат, и поддерживают действия, необходимые для выполнения задачи пользователя.

Внешний вид или «оболочка» операционной системы или прикладного программного обеспечения может быть изменена по желанию, поскольку графические пользовательские интерфейсы не зависят от функций приложения. Приложения обычно реализуют свои собственные уникальные элементы отображения графического интерфейса пользователя в дополнение к элементам графического интерфейса пользователя, уже присутствующим в существующей операционной системе.

Далее будет представлен обзор на современные программы для разработки графических интерфейсов.

1.1.1 Программный комплекс Delphi

Delphi - это язык программирования и интегрированная среда разработки (IDE) для быстрой разработки приложений для настольных компьютеров, мобильных устройств, веб-приложений и консольных программ, которые в настоящее время разрабатываются и поддерживаются Embarcadero Technologies, использующей парадигму, основанную на событиях. Язык также называется Object Pascal.

Компиляторы Delphi генерируют собственный код для Microsoft Windows, macOS, iOS, Android и Linux (только для x64). С 2016 года новые выпуски Delphi

выпускаются каждые шесть месяцев, и новые платформы добавляются примерно каждую секунду.

Delphi включает в себя редактор кода, визуальный дизайнер, встроенный отладчик, компонент управления исходным кодом и поддержку сторонних плагинов. Редактор кода имеет функции Code Insight (завершение кода), Error Insight (проверка ошибок в режиме реального времени) и рефакторинг. Дизайнер визуальных форм традиционно использовал библиотеку визуальных компонентов (VCL) для собственной разработки Windows, но позже была добавлена платформа FireMonkey (FMX) для кроссплатформенной разработки. Поддержка баз данных в Delphi очень сильна [5].

Delphi изначально разрабатывался Borland как инструмент быстрой разработки приложений для Windows как преемник Turbo Pascal. Delphi добавила полное объектно-ориентированное программирование к существующему языку, и с тех пор язык расширился для поддержки обобщенных и анонимных методов, а также поддержки родной компонентной объектной модели (COM). В 2006 году раздел инструментов разработчика Borland был переведен из Borland в 100%-ную дочернюю компанию, известную как CodeGear, которая была продана Embarcadero Technologies в 2008 году. В 2015 году Embarcadero была приобретена Idera Software, но знак Embarcadero был сохранен для подразделения инструментов разработчика.

Delphi и его аналог C ++, C ++ Builder, совместимы. Они имеют много основных компонентов, в частности IDE, VCL и большую часть библиотеки времени выполнения. Кроме того, они могут использоваться совместно в проекте. Например, C ++ Builder 6 и более поздние версии могут использовать исходный код Delphi и C ++ в одном проекте, в то время как пакеты, скомпилированные с помощью C ++ Builder, можно использовать из Delphi. В 2007 году продукты были выпущены совместно как RAD Studio, общий хост для Delphi и C ++ Builder, который можно приобрести с одним или обоими [5].

Delphi использует строго типизированный язык программирования высокого уровня, предназначенный для простоты использования и изначально основанный на более раннем языке Object Pascal. Изначально Pascal разрабатывался как язык общего назначения, «подходящий для краткого и логического выражения основных конструкций, известных в то время», и «его реализация должна была быть эффективной и конкурентоспособной с существующими компиляторами FORTRAN», но без Средства программирования высокого уровня или доступ к оборудованию. Turbo Pascal и его потомки, в том числе Delphi, поддерживают доступ к аппаратному обеспечению и низкоуровневому программированию, с возможностью включения кода, написанного на ассемблере и других языках. В объектной ориентации Delphi есть только полиморфизм на основе классов и интерфейсов.

Метаклассы это объекты первого класса. Объекты на самом деле являются ссылками на объекты (как в Java), на которые Delphi неявно отменяет ссылки, поэтому обычно нет необходимости вручную выделять память для указателей на объекты или использовать аналогичные методы, которые нужны некоторым другим языкам. Существуют выделенные типы строк со счетчиком ссылок, а также строки с нулевым символом в конце.

Delphi включает в себя интегрированную среду разработки. Все продукты Delphi поставляются с библиотекой визуальных компонентов (VCL), включая большую часть ее исходного кода. Сторонние компоненты (иногда с полным исходным кодом) и инструменты для улучшения IDE или для других задач разработки, связанных с Delphi, доступны, некоторые бесплатно. IDE включает графический интерфейс для локализации и трансляции созданных программ, которые могут быть развернуты в переводчике; Есть также сторонние инструменты с большим количеством функций для этой цели. Платформа VCL поддерживает высокий уровень совместимости исходного кода между версиями, что упрощает обновление существующего исходного кода до более новой версии Delphi.

1.1.2 Программный комплекс Python

Создание Python было начато в 1991 году Гвидо ван Россумом, когда он работал над распределенной операционной системой amoeba. Ему был нужен расширяемый язык, который будет поддерживать системные вызовы. База была взята из ABC и Модула-3. Он выбрал имя Питона в честь комедийного сериала от BBC «Летающий цирк Монти Пайтона», а вовсе не как имя змеи. С тех пор Python развивался при поддержке организаций, в которых работал Гвидо. Особенно активный язык в настоящее время совершенствуется, когда над ним работают не только команда разработчиков, но и целое сообщество программистов со всего мира. И все же последнее слово о направлении развития языка остается за Гвидо ван Россумом [6].

«Python – это интерпретируемый объектно-ориентированный язык программирования высокого уровня с динамической семантикой. Интегрированные структуры данных высокого уровня в сочетании с динамической типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (RAD, Rapid Application Development). Он также может использоваться в качестве сценарного языка для связи с программными компонентами. Синтаксис Python прост в изучении, подчеркивает удобочитаемость кода и снижает затраты на обслуживание программных продуктов [6].

Python поддерживает модули и пакеты, способствуя модульности и повторному использованию кода. Интерпретатор Python и большая стандартная библиотека доступны бесплатно как исходный и исполняемый код для всех важных платформ и могут свободно распространяться. «Python является универсальным языком программирования. У него есть свои преимущества и недостатки, а также его область применения. Python имеет обширную стандартную библиотеку для решения различных задач. Кроме того, Python имеет относительно простые инструменты для интеграции в C, C. ++ (и Java) языки, как путем встраивания интерпретатора в

программы на этих языках, так и с использованием библиотек, написанных на этих языках, в программах на Python.

Язык Python поддерживает различные парадигмы программирования: императивный (процедурный, структурный, модульный подходы), объектно-ориентированное и функциональное программирование. Можно предположить, что Python - это целая технология создания программных продуктов (и их прототипов). Он доступен практически на всех современных платформах (как 32-битных, так и 64-битных) с компилятором C и платформой Java [7].

Написание программы на чистом Python, особенно при использовании циклов, приводит к очень медленному выполнению. Вы можете справиться с этим и свести задачу к матричным операциям (векторизация), но далеко не все это возможно. Python становится своего рода связующим звеном между разными языками, когда обработка на нижнем уровне выполняется оптимизированными библиотеками, а взаимодействие между программными блоками и управлением заданиями происходит на более высоком уровне. Процесс написания прототипа программного пакета может быть проиллюстрирован следующим образом (см. Рисунок 1). Основные блоки (алгоритмы, тесты, графический интерфейс пользователя, ввод данных) могут быть достаточно быстро реализованы в Python.

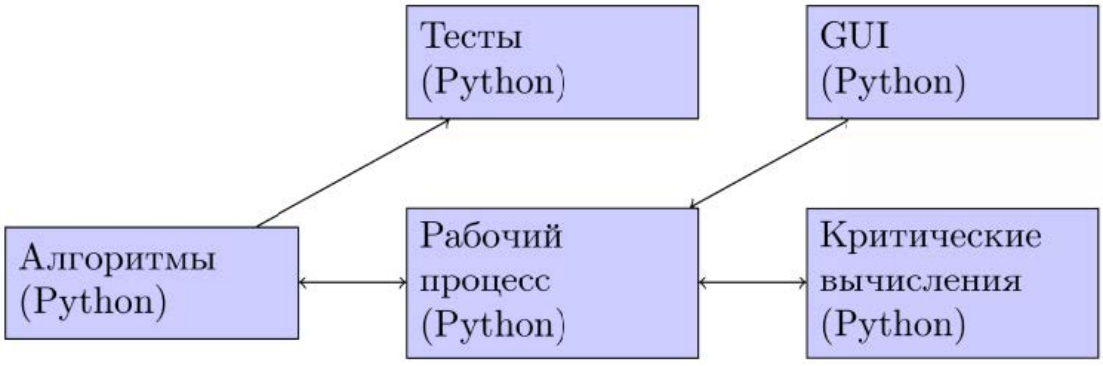

Рисунок 1 – Процесс прототипирования в программном комплексе Python

1.1.3 Программный комплекс Lazarus

Lazarus - это быстрая среда разработки для компилятора Free Pascal, похожая на Delphi. Этот проект основан на оригинальной кроссплатформенной библиотеке визуальных компонентов Lazarus Component Library (LCL). Кроссплатформенное программное обеспечение - это программное обеспечение, которое работает на нескольких аппаратных платформах и / или в операционной системе [8].

Free Pascal - это компилятор языка Pascal и Object Pascal, который работает в Windows, Linux, Mac OS X, FreeBSD и других операционных системах. Это означает, что разработанные приложения могут работать практически под любой операционной системой.

Все, что мы видим на экране во время работы различных приложений, все элементы (кнопки, ползунки, меню и т. Д.) Могут быть реализованы в Lazarus. Лазарь использует технологию визуального программирования. Для создания графического пользовательского интерфейса пользователь использует готовые компоненты, значки которых расположены в окне компонентов. После того, как он вставил компонент в форму, автоматически генерируется программный код для него. Только действия, которые будет выполнять это приложение, должны быть запрограммированы вручную.

Процесс создания приложения можно разделить на следующие этапы:

1) создание проекта. В результате на экране появляется пустая форма (окно будущего приложения);
2) создание графического интерфейса проекта – расположение необходимых элементов, задание размеров, изменение свойств;
3) написание программного кода, который определит, что будет делать наша программа.
4) отладка программы.

1.2 Обзор протоколов передачи данных

Обмен информацией между устройствами, которые являются частью автоматизированной системы (компьютеры, контроллеры, датчики, исполнительные механизмы), как правило, осуществляется через промышленную сеть (fieldbus, полевая шина). В настоящее время существует более 50 типов промышленных сетей (ModBus, PROFIBUS, DeviceNet, протокол HART, FDDI, LonWorks и другие). Однако распространена только часть из них. Большинство систем управления процессами в России используют следующие протоколы:

1) HART-протокол;
2) PROFIBUS – протокол;
3) ModBus – протокол;

Промышленная сеть связана с ее компонентами (устройствами, сетевыми узлами) через интерфейсы. Сетевой интерфейс относится к логической и (или) физической границе между устройством и средой передачи информации. Наиболее важными параметрами интерфейса являются полоса пропускания и максимальная длина подключаемого кабеля [9]. Промышленные интерфейсы обычно обеспечивают гальваническую развязку между подключенными устройствами. В промышленной автоматизации чаще всего используются последовательные интерфейсы, такие как RS-485, RS-232, RS-422, Ethernet, Can.

1.2.1 Протокол HART

В середине 1980-х годов американская компания Rosemount разработала протокол HART (Highway Addressable Remote Transducer). В начале 1990-х протокол был дополнен и стал коммуникационным стандартом открытого общения. Сначала он был нормализован только для использования в режиме двухточечного соединения, затем стало возможным использовать протокол в режиме многоточечного соединения («multidrop»). Основные технические параметры, определяемые стандартом на HART-протокол, представлены в таблице 1.

Таблица 1 – Технические параметры, определяемые стандартом на HART-протокол

Топология	«Точка-точка» (стандартная) или шина
Максимальное количество устройств	Одно подчиненное устройство и два ведущих устройства (стандартный режим); 15 подчиненных устройств, 2 ведущих устройства (многоточечный режим с удаленным питанием)
Максимальная протяженность линии связи	3 км (стандарт); 100 м (многоточечный режим)
Тип линии	Экранированная витая пара
Интерфейс	4-20 мА, токовая петля (аналоговый)
Скорость передачи	1,2 кбит/с
Метод обращения	Polling (механизм опроса с уникальной адресацией каждого устройства)
Максимальная длина пакета данных	0-25 байт
Время цикла обновления данных	Около 500 мс (в пакетном режиме — 330 мс)
Надежность передачи данных	1 ошибка на 105 бит, контроль по четности каждого байта, байт контрольной суммы для каждого пакета

Протокол HART имеет два основных режима работы: «точка-точка» и режим множественного доступа. В режиме «точка-точка» имеется одно ведущее устройство и одно ведомое. Преимущество этого режима заключается в том, что цифровые данные легко передаются по существующей линии 4…20 мА, что обеспечивает более детальный мониторинг устройства по существующей инфраструктуре сетей связи. В режиме множественного доступа к одной линии присоединяется несколько ведущих и ведомых устройств, поэтому могут передаваться только данные протокола HART FSK, а постоянный ток в линии фиксируется на уровне 4 мА. Режим множественного доступа может быть полезен, если много вынесенных устройств обменивается данными с единой системой управления, но в этом случае постоянный ток интерфейса «токовая петля» не может

быть использован для непрерывного отслеживания основного измеряемого значения [9].

HART – это гибкий способ коммуникации для различных приложений промышленной автоматизации. Сам метод предлагает множество преимуществ, которые снижают стоимость, упрощают проектирование и обеспечивают такие результаты, как [10]:

1) передача (сопутствующей) цифровой информации без прерывания основного аналогового сигнала;
2) простая реализация с использованием существующей двухпроводной инфраструктуры 4…20 мА;
3) гибкие способы работы для удовлетворения потребностей различных систем.

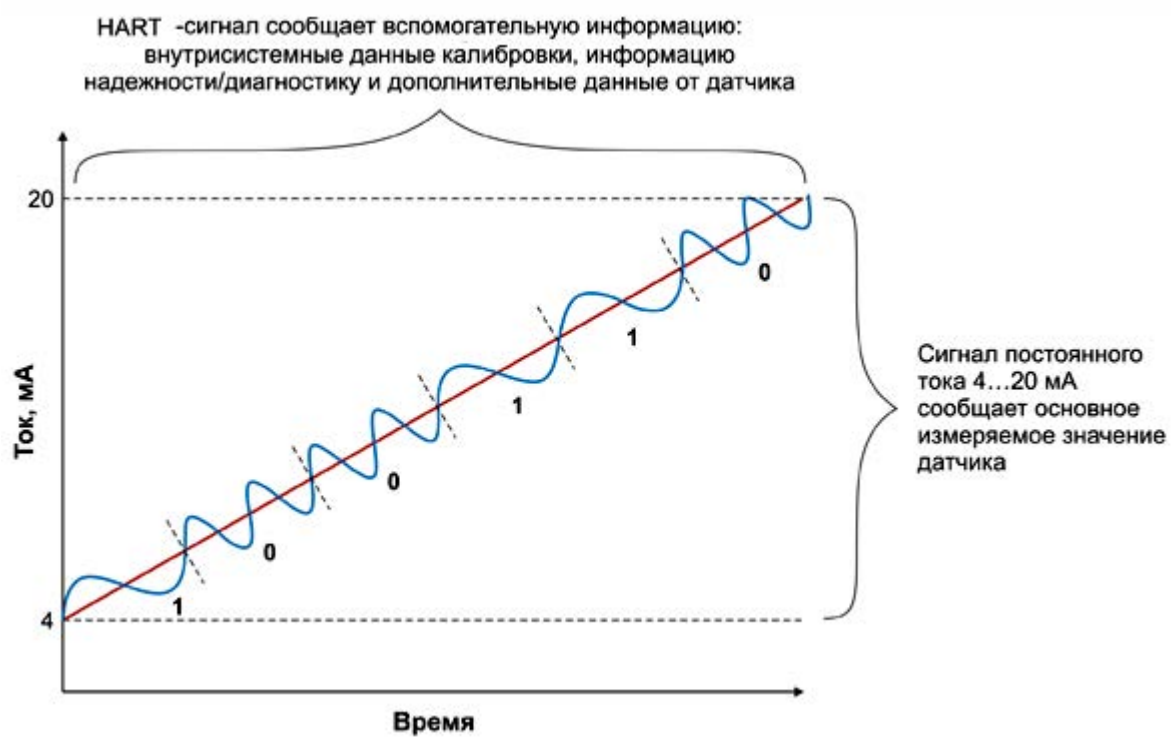

Рисунок 2 – Ток 4…20мА и информация HART

Протокол HART функционирует согласно принципу ведущий-ведомый, и все операции связи инициируются ведущим устройством. Протокол HART поддерживает два ведущих устройства: первичный ведущий, обычно система управления, и вторичный ведущий - блок управления, который используется в полевых условиях (ноутбук с HART модемом или ручной управляющий модуль - HART коммуникатор). Распределение активности ведущих устройств при управлении ведомым, выполняется на временной основе. После каждой транзакции одно из двух ведущих устройств может принимать на себя связь внутри определенного временного окна.

Для настройки датчиков с поддержкой HART используются либо модули ручного управления - коммуникаторы, либо специальный пакет программного обеспечения для модемов PC-HART. Несмотря на то, что протокол HART является стандартизированным и открытым протоколом, не все устройства можно настроить с помощью одного коммуникатора HART или модема со специальным программным обеспечением (например, PACTware). Большинство производителей выпускают коммуникаторы и модемы, которые в большинстве случаев предназначены только для устройств одного и того же производителя. Достоинствами протокола HART являются [11]:

1) простая настройка, сервис и техническое обслуживание устройств с поддержкой HART;
2) совместимость с обычными аналоговыми полевыми устройствами и датчиками;
3) открытый стандарт, доступный каждому изготовителю оборудования КИП;
4) достаточно высокая помехоустойчивость.

1.2.2 Протокол PROFIBUS

Технологии промышленных информационных сетей используются для обмена информацией между инструментами автоматизации, которые обычно могут

находиться на значительном расстоянии друг от друга. Одной из самых известных и широко используемых информационных технологий является промышленная сеть PROFIBUS [12].

Физический уровень. Этот уровень имеет дело с передачей битов по физическим каналам, таким как коаксиальные кабели, кабели витой пары или оптоволоконные кабели. Свойства физического носителя для передачи данных, такие как полоса пропускания, помехоустойчивость, импеданс и другие, связаны с этим уровнем. Свойства электрических сигналов определяются на одном уровне, например, требования к фронту импульсов, уровню напряжения или тока передаваемого сигнала, типу кодирования, скорости передачи сигнала. Типы разъемов и назначение каждого контакта также стандартизированы здесь.

Уровень канала. На физическом уровне биты просто отправляются. Не учитывается, что в некоторых сетях, в которых линии связи поочередно используются (совместно используются) несколькими парами взаимодействующих компьютеров, физическая среда передачи может быть занята. Поэтому одной из задач канального уровня является проверка доступности среды передачи. Другой задачей канального уровня является реализация механизмов обнаружения и исправления ошибок. Для этой цели биты на уровне канала группируются в числа, называемые кадрами. Уровень канала обеспечивает правильную передачу каждого кадра, помещая специальную последовательность битов в начало и конец каждого кадра, чтобы пометить его, а также вычисляет контрольную сумму, суммирует все байты кадра определенным образом и добавляет контрольную сумму. стать основой. Когда приходит кадр, получатель подсчитывает контрольную сумму полученных данных и сравнивает результат с контрольной суммой из кадра. Если они совпадают, структура считается действительной и принятой. Если контрольные суммы не совпадают, выдается ошибка.

Сетевой уровень. Этот уровень служит для формирования единой транспортной системы, которая объединяет несколько сетей с различными принципами передачи

информации между конечными узлами. Рассмотрим функции сетевого уровня на примере локальных сетей. Протокол канального уровня локальных сетей обеспечивает передачу данных между любыми узлами только в сети с соответствующей типичной топологией. Таким образом, в сети доставка данных регулируется канальным уровнем, а доставка данных между сетями - сетевым уровнем. Сообщения сетевого уровня называются пакетами. Понятие «номер сети» используется при организации доставки пакетов на уровне сети. В этом случае адрес получателя состоит из номера сети и номера компьютера в этой сети.

Транспортный уровень. Посылки могут быть искажены или потеряны на пути от отправителя к получателю. Хотя некоторые приложения имеют свои собственные инструменты для устранения неполадок, есть те, кто предпочитает иметь дело с надежными соединениями немедленно. Работа транспортного уровня заключается в обеспечении того, чтобы приложения или верхние уровни стека - приложение и сеанс - передавали данные с требуемым уровнем надежности. Модель OSI определяет пять классов обслуживания, которые предоставляются транспортным уровнем. Эти типы услуг отличаются качеством предоставляемых услуг: возможностью восстановления разорванного соединения, срочностью, доступностью средств мультиплексирования для множественных соединений между различными прикладными протоколами через общий транспортный протокол и, прежде всего, способностью исправлять и обнаруживать ошибки передачи, такие как искажение, дублирование и потеря пакетов.

Сеансовый уровень. Уровень сеанса обеспечивает управление диалогом для записи, которая в настоящее время активна, а также дает инструменты синхронизации. Последний позволяет нам вставлять контрольные точки в длинные передачи данных, чтобы в случае ошибки вы могли вернуться к последней контрольной точке, а не начинать заново. На практике только несколько приложений используют уровень сеанса, и он редко реализуется.

Уровень презентации. Этот уровень гарантирует, что информация, передаваемая прикладным уровнем, понимается прикладным уровнем в другой системе.

Прикладной уровень. На самом деле прикладной уровень состоит только из ряда различных протоколов, с помощью которых пользователи сети получают доступ к общим ресурсам, таким как файлы, принтеры или гипертекстовые веб-сайты, и организуют свое сотрудничество, например, с использованием протокола электронной почты [13].

Контроль доступа к шине в PROFIBUS. Существует два основных требования для контроля доступа к шине PROFIBUS. С одной стороны, для надежной связи между одноранговыми устройствами автоматизации или ПК каждому участнику необходимо иметь доступ к шине в течение определенного временного окна, чтобы решить свои проблемы со связью. С другой стороны, обмен данными между сложными устройствами автоматизации или ПК и простыми децентрализованными периферийными устройствами требует быстрого обмена данными с минимальными усилиями по протоколу. Оба требования выполняются благодаря гибридному контролю доступа к шине, который включает в себя: децентрализованный обмен токеном (токеном) между активными участниками (мастерами); Центральный обмен ведущий-ведомый для обмена данными между активными и пассивными узлами на шине PROFIBUS. Активный участник, которому принадлежит маркер, в настоящее время принимает на себя функции мастера на шине для связи с пассивными и активными участниками. Обмен сообщениями через автобус происходит путем обращения к участникам. Уникальный адрес присваивается каждому члену PROFIBUS. Адрес назначается от 0 до 126. Максимальное количество участников в автобусе не превышает 127 [14].

1.2.3 Протокол MudBus

Modbus - это протокол связи, разработанный системами Modicon. Проще говоря, это метод, используемый для передачи информации по последовательным линиям между электронными устройствами. Устройство, запрашивающее информацию, называется Modbus Master, а устройства, предоставляющие информацию, - это Modbus Slave. В стандартной сети Modbus есть один ведущий и до 247 ведомых, каждый с уникальным адресом ведомого от 1 до 247. Ведущий также может записывать информацию на ведомые устройства.

Modbus - это открытый протокол, означающий, что производители могут бесплатно встраивать свое оборудование без выплаты роялти. Это стало очень распространенным протоколом, широко используемым многими производителями во многих отраслях промышленности. Modbus обычно используется для передачи сигналов от контрольно-измерительных приборов и устройств управления обратно в главный контроллер или систему сбора данных.

Modbus передается по последовательным линиям между устройствами. Самой простой настройкой будет один последовательный кабель, соединяющий последовательные порты на двух устройствах: ведущем и подчиненном. Данные отправляются в виде серии единиц и нулей, называемых битами. Каждый бит отправляется как напряжение. Нули отправляются как положительные напряжения, а единицы как отрицательные. Биты отправляются очень быстро. Типичная скорость передачи составляет 9600 баит (бит в секунду).

Протокол Modbus разработан для использования в программируемых логических контроллерах, таких, как управление электроприводом. В настоящее время является очень распространенным протоколом, используемых в различных промышленных системах. К примеру, данный протокол используется в контроллерах шаговых двигателей Онитекс. Широко используется для передачи данных последовательные линии связи, основанных на интерфейсах RS-485, RS-422, RS-232. В начале развития применялся интерфейс RS-232, как один из наиболее простых промышленных

интерфейсов для последовательной передачи данных. В настоящее время протокол часто используется поверх интерфейса RS-485, что позволяет добиться высокой скорости передачи, больших расстояний и объединения нескольких устройств в единую сеть, тем более что протокол Modbus поддерживает адресацию. Широкая распространенность протокола Modbus, обусловленная его простотой и надежностью, позволяет легко интегрировать устройства, поддерживающие Modbus, в единую сеть [15].

Контроллеры могут быть настроены для связи в стандартных сетях Modbus с использованием одного из двух режимов передачи: ASCII или RTU.Режимы делятся на ASCII и RTU (Remote Terminal Unit) [16].

Таблица 2 – Характеристики режимов ASCII и RTU

Характеристика	ASCII (7-бит)	RTU(8-бит)
Система кодирования	Используются ASCII символы 0-9,A-F	8-битовая двоичная система
Число бит на символ		
Стартовые биты	1	1
Биты данных (LSB вперед)	7	8
Четность	Вкл./Выкл.	Вкл./Выкл.
Стоповые биты	1 или 2	1 или 2
Контрольная сумма	LRC (Longitudinal Redundancy Check). LRC	CRC (Cyclical Redundancy Check). CRC_16

Когда контроллеры настроены на обмен данными в сети Modbus с использованием режима ASCII (американский стандартный код для обмена информацией), каждый восьмибитный байт в сообщении отправляется в виде двух

символов ASCII. Основным преимуществом этого режима является то, что он допускает интервалы времени до одной секунды между символами без возникновения ошибки.

Когда контроллеры настроены для связи в сети Modbus с использованием режима RTU (Remote Terminal Unit), каждый восьмибитный байт в сообщении содержит два четырехбитных шестнадцатеричных символа. Основным преимуществом этого режима является то, что его более высокая плотность символов обеспечивает лучшую пропускную способность, чем ASCII, для той же скорости передачи данных. Каждое сообщение должно передаваться в непрерывном потоке.

В режиме ASCII сообщения начинаются с символа двоеточия (:) (ASCII 3A hex) и заканчиваются парой перевода строки перевода строки (CRLF) (ASCII 0D и 0A hex). Допустимые символы, передаваемые для всех остальных полей: шестнадцатеричные 0 ... 9, A ... F. Сетевые устройства непрерывно контролируют сетевую шину на наличие символа двоеточия. Когда оно получено, каждое устройство декодирует следующее поле (поле адреса), чтобы выяснить, является ли оно адресуемым устройством.

1.3 Выводы по разделу 1

В наши дни предприятия стремятся автоматизировать процессы своих производств. Это вызвано бурным развитием технологий и внедрением интеллектуальных систем управления технологическими процессами. Для этого устанавливается большое количество измерительных средств, особенно широкое применение нашли датчики измерения температуры. Но со временем есть риск получить недостоверные результаты измерений, что приводит к неправильному принятию решения по управлению технологическими процессами.

По результатам проведённого обзора были рассмотрены и изучены программы и протоколы создания архитектуры графических интерфейсов для датчиков

температур. Были выявлены плюсы и минусы пользования той или иной программой, либо протоколом.

Исходя из таблицы 3 можно сделать вывод, что для разработки архитектуры графического интерфейса оптимально использовать программный продукт Python. Также проведено сравнение характеристик протоколов передачи данных.

Таблица 3 – Сравнение программ для разработки графического интерфейса для датчика температуры

Программные продукты	Стоимость	Кроссплатформенная разработка	Языки программиро-вания
Delphy	250т.р	Присутствует	TurboPascal, Pascal, C++, C#
Python	Бесплатно (открытый код)	Присутствует	C#, C++, Java
Lazarus	Бесплатно (открытый код)	Присутствует (взаимодействие с продуктами Delphy)	FreePascal Object Pascal

Таблица 4 – Характеристики основных протоколов передачи данных

Протоколы	Максимальная скорость передачи данных	Максимальное количество подключаемых устройств	Максимальная длина кабеля (с репитером/без репитера)	Совместимые протоколы
Profibus	115,2 кбит/с	126 штук	7 км / 1,1 км	DP -V0, DP-V1, DP-V2
Modbus	1,7 Мбит/с	247 штук	10 км / 1,2 км	Modbus RTU
HART	1,2 кбит/с	15 подчиненных и 2 ведущих	3 км	HART

По техническому заданию необходимо разработать программу и алгоритм графического интерфейса для датчика температуры с функцией метрологического самоконтроля. Данный тип микроконтроллера осуществляет передачу данных через шину RS485 которая совместима с протоколами Modbus RTU, Ethernet, UID. Из этого следует, что необходимо выбрать протокол ModBus.

Задачи работы:

1) разработать алгоритм работы программы графического интерфейса для датчика температуры с метрологической функцией самоконтроля;
2) отладить программу с использованием рабочих образцов датчика;
3) провести корректировки программы по результатам испытаний.

2 РАЗРАБОТКА ПРОГРАММЫ И АЛГОРИТМА РАБОТЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ДЛЯ ДАТЧИКА ТЕМПЕРАТУРЫ С ФУНКЦИЕЙ МЕТРОЛОГИЧЕСКОГО САМОКОНТРОЛЯ

2.1 Разработка алгоритма графического интерфейса

Алгоритм программы – это совокупность последовательных действий (схема действий, точное предписание) которое определяет процесс перехода от первичных данных к желаемому результату. Существует две формы представления алгоритма: словесное описание и графическое представление (с помощью блок-схем). В среде программирование под алгоритмом решения задачи понимается совокупность процедур обработки событий.

Пользовательский интерфейс – это своеобразный коммуникационный канал, обеспечивающий взаимодействие пользователя с компьютером. Существует три принципа проектирования пользовательских интерфейсов:

1) структуризация;
2) простота;
3) обратная связь.

Под структуризацией понимают искусственно выделенные программистом взаимодействующие части программы. Использование структуризации устраняет проблему сложности разработки, повышает надежность работы программы.

Под простотой понимают простой программный продукт, в котором все функции программы удобны в использовании и взаимодействие с ними происходило на интуитивном уровне.

Обратная связь интерфейса - это краткие, понятные, написанные на понятном пользователю языке сообщения, которые должен получать пользователь о действиях системы и событиях внутри нее [17].

Пользовательский графический интерфейс представляет собой окно, в котором расположены различные элементы управления. Взаимодействие пользователя с программой осуществляется при помощи клавиатуры и мыши. Внутри интерфейса расположены кнопки и разделы меню.

Разработка интерфейса данного проекта осуществляется путём программирования окна на языке Python.

Последовательность создания алгоритма для Python:

1) полный набор исходных данных задачи;
2) разработка самого алгоритма;
3) написание программного кода;
4) отладка программы;
5) тестирование программы;
6) создание справочной системы;

Поясним каждый из шагов.

Полный набор исходных данных. Подразумевается, что для начала работы необходимо подготовить все данные которые будут определены в алгоритме. Если данные будут включены, например, после написания кода, то внести новые данные будет невозможно.

Разработка самого алгоритма. На данном этапе создания алгоритма необходимо определить последовательность действий, которые нужно получить для удовлетворительного результата (цели). Алгоритм может состоять из различных вариантов прохода алгоритма. Если задача может быть решена несколькими способами, то возможны различные варианты алгоритма. В таком случае программист выбирает наиболее подходящий вариант алгоритма.

Написание программного кода. После того как разработан алгоритм программы, можно приступать к написанию программного кода.

Отладка программы. Обычно программное обеспечение содержит ошибки и ошибки, которые обычно удаляются. Отладка - это процесс исправления ошибки в

программном обеспечении. Процесс отладки начинается, как только написан код программного обеспечения. Затем он продолжается последовательно, так как код объединяется с другими единицами программирования для формирования программного продукта. Отладка имеет много преимуществ, таких как [18]:

1) он немедленно сообщает об ошибке. Это позволяет более раннее обнаружение ошибки и делает процесс разработки программного обеспечения беззаботным и беспроблемным;

2) он также предоставляет максимально полезную информацию о структурах данных и позволяет легко интерпретировать.

Отладка помогает разработчику уменьшить количество ненужной и отвлекающей информации. Благодаря отладке разработчик может избежать сложного одноразового тестирования кода, чтобы сэкономить время и энергию при разработке программного обеспечения.

Тестирование программы. Процесс исследования программного обеспечения (в нашем случае, программ написанная в Python 3.4.0) с целью получения информации о качестве продукта.

2.2 Алгоритм создания интерфейса в Python

1) Подключаем базу данных, для этого создаем компоненту ADOConnection, в свойствах которой прописываем путь к нашей базе данных. Переносим содержимое каждой таблицы БД в наше приложение. Для этого, для каждой таблицы БД проделываем следующие операции:

а) создаем компоненту ADOTable. Изменяем свойства компоненты:

Connection:= ADOConnection1;

Table Name:= <Файл>;

Active:= True.

б) создаем компоненту DataSource. Изменяем свойства компоненты:

DataSet:= ADOTable1();

в) создаем компоненту DBGrid. Изменяем свойства компоненты:

DataSource:= DataSource1();

г) создаемкомпоненту DBNavigator. Изменяем свойства компоненты:

DataSource:= DataSource1().

2) создаем дополнительное окно, в котором будут выводиться результаты последующих запросов. Проделываем следующие операции:

а) создаем компоненту ADOQuery. Изменяем свойства компоненты:

Connection:= ADOConnection1;

б) создаемкомпоненту DataSource. Изменяем свойства компоненты:

DataSet:= ADOQuery1;

в) создаем компоненту DBGrid. Изменяем свойства компоненты:

DataSource:= DataSource<запрос>.

1) создаем кнопки, при нажатии на которые будут выполняться нужные нам запросы. В свойствах каждой кнопки изменяем свойство ShowHint. ShowHint:=true. В свойстве Hint прописываем описание каждой кнопки;

2) в процедуре каждой кнопки прописываем программный код каждого запроса;

3) добавляем подписи;

4) изменяем расположение объектов в программе;

5) проверяем код программы на наличие ошибок;

6) компилируем программу.

Далее мы приводим подробное описание результатов по каждым из пунктов нашей программы.

Первым делом разрабатывается меню программы. Так как программа должна обеспечить возможность управления датчиком температуры с метрологической функцией самоконтроля, необходимо создать окно настройки портов, необходимо разработать окно настройки портов датчика. Связь датчика температуры с

персональным компьютером осуществляется через микроконтроллер, в котором передача данных происходит через протокол Modbus, по COM порту.

Следовательно, в окне настройки портов должны присутствовать ячейки, где можно выбрать нужный нам вид порта, скорость передачи и все виды и значения данных. Путём программирования создаётся вкладка «Файл», в котором расположено окно «Установки COM порта». В этом окне, которое называется «Setup» мы располагаем все нужные нам настройки портов.

Процедура, написанная на языке «Python» и связанная с кнопкой "Файл", выполняет запрос на показ шторки с дополнительным меню:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
orm1.ADOQuery1.SQL.Clear;
form1.ADOQuery1.SQL.Text:='SELECT ;
'FROM  «Файл» INNER JOIN (Установка COM портов INNER JOIN (INNER JOIN
Setup ON '+
'Setup.[Settings] = Port.[Baud rate]) ON Data bits.[Stop bits] '+
'= Parity.[None]) ON parity.[Odd] = Parity.[Mark] '+
'GROUP BY Flow control.[None] '+
'HAVING (((Software.[hardware]) Between));';
form1.ADOQuery1.Active:=True;
end;
```

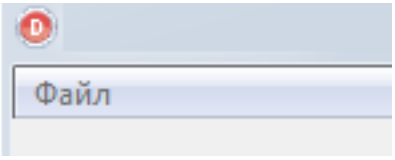

Рисунок 3 – Пункт меню «Файл»

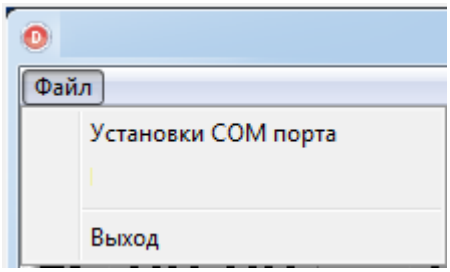

Рисунок 4 – Пункт «Установки COM порта»

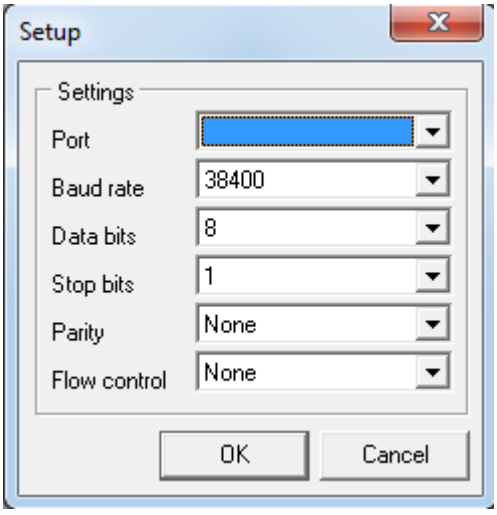

Рисунок 5 – Окно «Setup»

Далее необходимо разработать окно, в котором будет показано как работают датчики температуры. Назовём его «Чтение», так как здесь происходит вывод результатов. Путём программирования разместим в окне температуру, которую показывают датчики, и окно состояния, где показано, как изменяется температура с течением времени. Результат выводится в виде временных диаграмм, нажав на клавишу «Вкл.График», по необходимости можно изменять масштаб, нажимая на кнопки плюс либо минус. Также в этом окне можно посмотреть какую температуру показывает отдельно взятый датчик либо всех сразу, путём нажатия на кнопку выбора нужного датчика либо нажав на кнопку «Всё». По необходимости можно как включить, так и выключить отображения даты и времени, путям нажатия

соответствующей клавиши «Дата+время». Также расположим здесь кнопку включения связи, чтобы отслеживать показания температуры в реальном времени.

Процедура, написанная на языке «Python» и связанная с кнопкой "Чтение", выполняет запрос на показ шторки с дополнительным меню:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
orm1.ADOQuery1.SQL.Clear;
form1.ADOQuery1.SQL.Text:='SELECT ;
'FROM  «Чтение» INNER JOIN (ВКЛ.Связь INNER JOIN (INNER JOIN Setup ON '+
'ВКЛ.График.[Settings] = Port.[T1::E10]) ON Data bits.[Stop bits] '+
'= Parity.[None]) ON parity.[Data+time] = Parity.[Mark] '+
'GROUP BY Flow control.[None] '+
form1.ADOQuery1.Active:=True;
end;
```

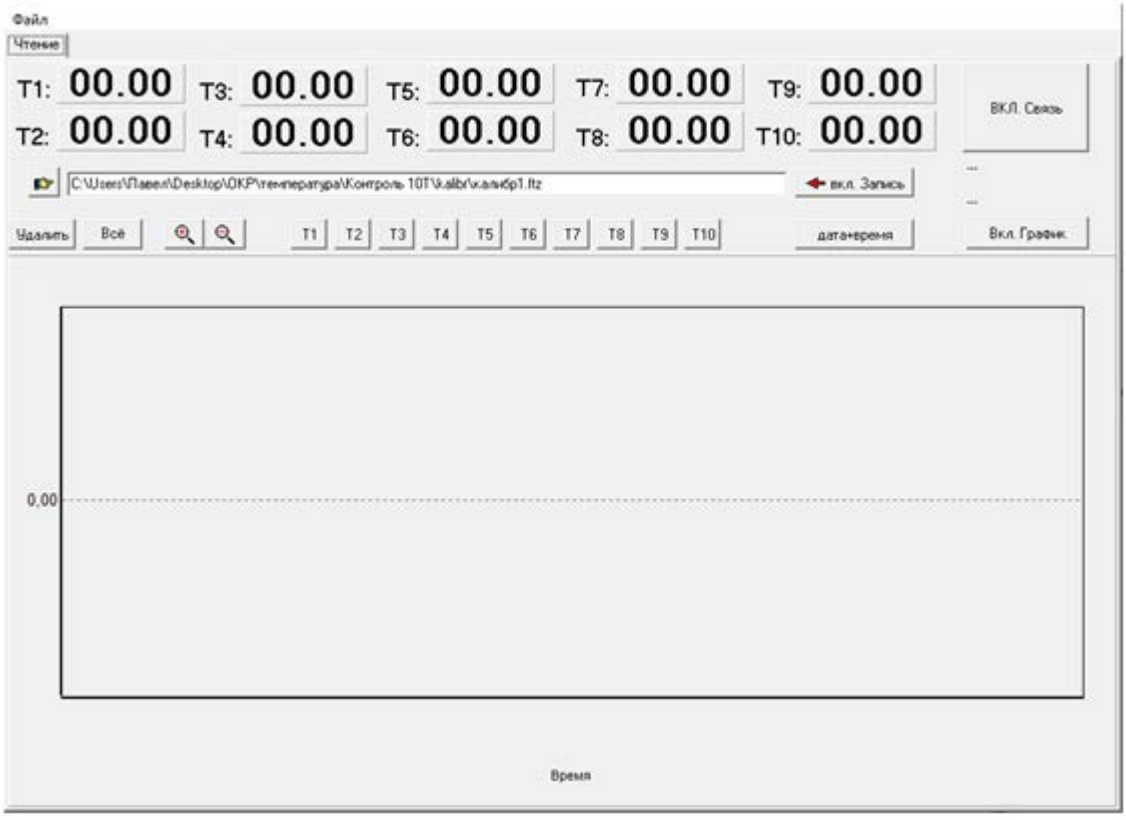

Рисунок 6 – Окно «Чтение»

Для удобства понимания необходимо создать окно, где будут отображены температуры всех датчиков. Для этого путём программирования создаётся окно «График». Здесь будут показаны температуры всех датчиков относительно друг друга. Здесь также, как и в окне «Чтение», расположены кнопки управления графиками и выбора датчиков, но в отличие от прошлого окна, здесь можно посмотреть работу датчиков за определённый отрезок времени. Для этого здесь добавлены кнопки дробления «/1», «/10», «/100».

Процедура, написанная на языке «Python» и связанная с кнопкой "График", выполняет запрос на показ шторки с дополнительным меню:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
orm1.ADOQuery1.SQL.Clear;
form1.ADOQuery1.SQL.Text:='SELECT ;
'FROM  «График» INNER JOIN (Прочитать из файла INNER JOIN (INNER JOIN Setup ON '+
Дата+время.[Settings] = Port.[T1::E10]) ON Data bits.[Stop bits] '+
'= Parity.[None]) ON parity.[Data+time] = Parity.[Mark] '+
'GROUP BY Flow control.[None /1; /10; /100] '+
'HAVING (((Software.[hardware]) Between));';
form1.ADOQuery1.Active:=True;
end;
```

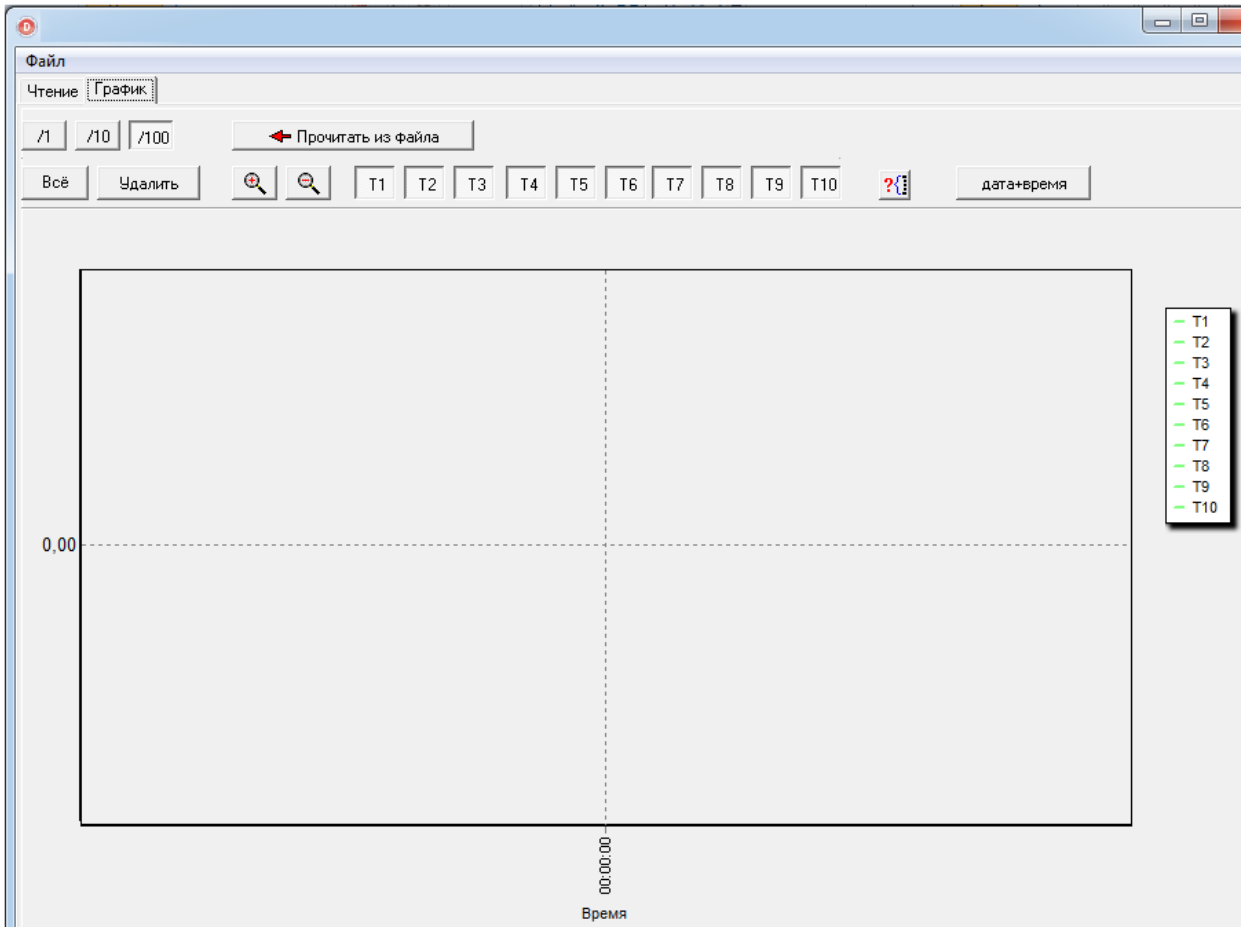

Рисунок 7 – Окно «График»

Чтобы датчики температуры работали корректно необходимо их откалибровать. Для этого, путём программирования, создано окно «Калибровки». Для калибровки нужно читать значения дополнительных параметров датчика: код АЦП, код температуры окружающей среды, сопротивление ТС, передавать код ЦАП, передавать и читать коэффициенты полиномов математической модели датчика, а также передавать команды для реализации калибровки: запись во флэш-память, включение фильтра, отключение фильтра, передача кода в ЦАП, отключение передачи кода в ЦАП.

Процедура, написанная на языке «Python» и связанная с кнопкой "Калибровки", выполняет запрос на показ шторки с дополнительным меню:

Procedure TForm1.Button1Click(Sender: TObject);

```
Begin
orm1.ADOQuery1.SQL.Clear;
form1.ADOQuery1.SQL.Text:='SELECT ;
'FROM  «Калибровки» INNER JOIN (Установленный файл калибровки INNER JOIN
(INNER JOIN Setup ON '+
Записать.[Settings] = Port.[T1::T10]) ON Data bits.[Stop bits] '+
'= Parity.[None]) ON parity.[Data+time] = Parity.[Mark] '+
'GROUP BY Flow control.[None] '+
'HAVING (((Software.[hardware]) Between));';
form1.ADOQuery1.Active:=True;
end;
```

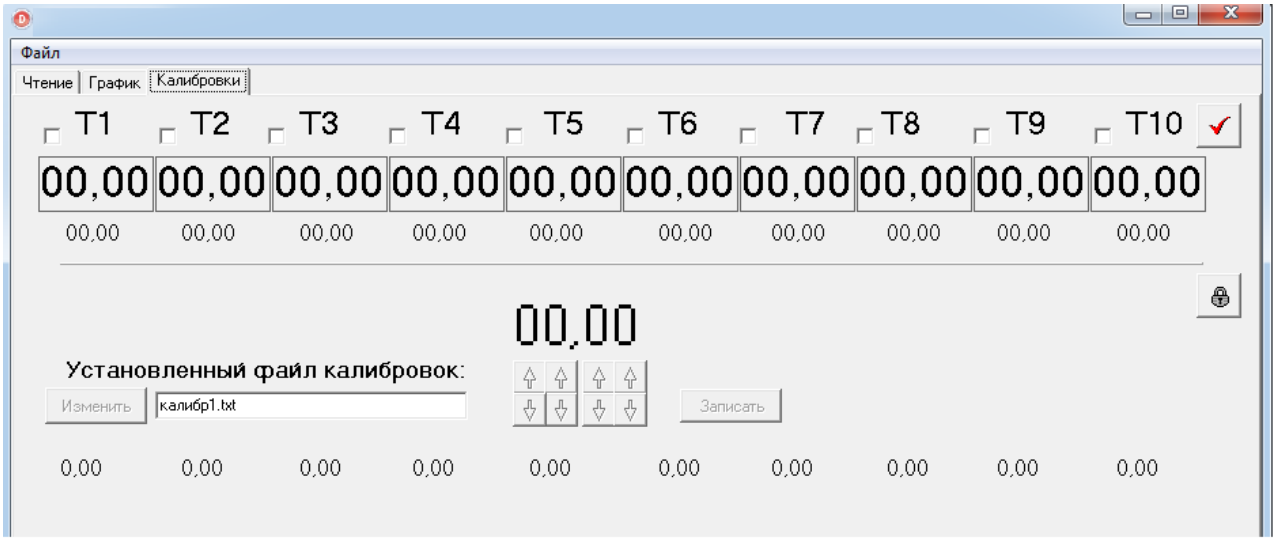

Рисунок 8 – Окно «Калибровки»

По техническому заданию необходимо разработать интерфейс для датчика температуры с функцией метрологического самоконтроля. Значит нужно добавить в наш интерфейс вкладку, где будет происходить диагностика. Путём программирования создаём окно «Диагностика». Здесь мы расположили панель состояния, панель выбора порта, панели для настройки диагностики: передача

команд: тест, запись функции диагностики, чтение функции диагностики, включение режима коррекции, выключение режима коррекции.

Процедура, написанная на языке «Python» и связанная с кнопкой "Диагностика", выполняет запрос на показ шторки с дополнительным меню:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
orm1.ADOQuery1.SQL.Clear;
form1.ADOQuery1.SQL.Text:='SELECT ;
'FROM  «Диагностика» INNER JOIN (Порт термодатчика INNER JOIN (INNER JOIN
Setup ON '+
Установить.[Settings] = Port.[T1::E10]) ON Data bits.[Период опроса датчика] '+
'= Parity.[None]) ON parity.[Время опроса] = Parity.[Mark] '+
'GROUP BY Flow control.[Опрос датчика] '+
'HAVING (((Software.[Очистка окна]) Between));';
end;
```

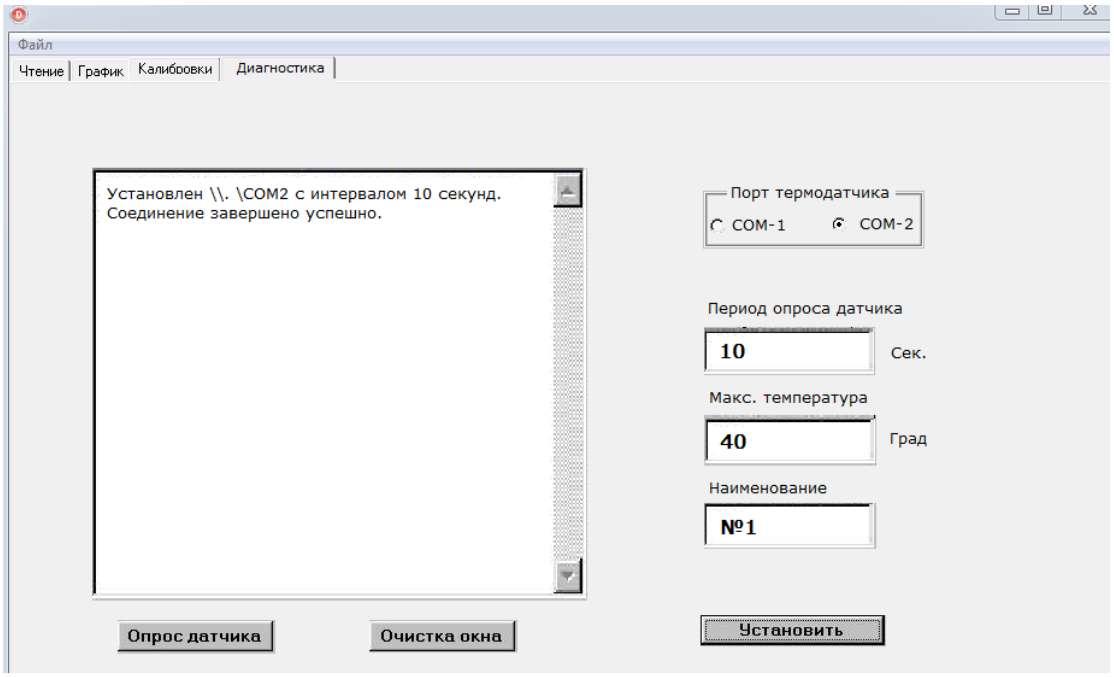

Рисунок 9 – Окно «Диагностика»

2.3 Разработка алгоритма работы интерфейса с датчиком

Первое действие в программе - основные настройки. Мы устанавливаем указатель стека на последнюю ячейку ОЗУ, исходное состояние каналов связи с датчиками температуры и скорость обмена данными по протоколу ModBus, разрешаются прерывания от таймера / счетчика и от ModBus, записываются номер и индивидуальные адреса датчиков температуры в ОЗУ, необходимые константы записываются в реестр. Когда начальные настройки завершены, начинается часть программы, в которой опрашивается датчик температуры. Он повторяется циклически во время подачи питания на датчик или до тех пор, пока не возникнет запрос на прерывание. Зонд начинается с сигнала сброса, за которым следует команда игнорировать адрес датчика температуры [19]. Команда запуска «CONVERT T» для измерения температуры позволяет преобразовать значения температуры в цифровой вид датчика. Аналого-цифровое преобразование значений температуры занимает от 750 мс до 800 мс. Чтобы получить правильное значение температуры, следует ожидать паузу в 800 мс. Пауза поддерживается таймером / счетчиком 0. Мы можем выполнять другие действия во время перерыва (например, обмениваться данными с компьютером). Датчик запрашивается после паузы. Вызов датчика начинается с сигнала сброса на линии связи с датчиком. После сигнала сброса и ответного сигнала от датчика следует команда «ПЗУ». Эта команда сообщает датчику, что на линии связи устанавливается один 64-битный адрес датчика. После того, как адрес был установлен в строке, датчик температуры сравнивает установленный адрес со своим собственным адресом, и тогда датчик готов [20].

Значение температуры считывается в блоке 7 (Рисунок 10) и записывается в соответствующие ячейки ОЗУ. Программа ветвится в блоке 8: если измеренная температура не превышает заданные предельные значения, результат отображается

на экране ПК (блок 9). Если температуру необходимо измерить дальше (блок 11), перейдите к блоку 4. Если нет, программа заканчивается.

Рисунок 10 – Блок-схема алгоритма работы графического интерфейса и датчика

2.3.1 Разработка алгоритма работы интерфейса с несколькими датчиками температуры

Для правильной работы разработанной системы датчиков необходимо разработать алгоритмы первичной обработки информации с нескольких датчиков. Эти алгоритмы должны обеспечивать генерацию аварийных сообщений для оператора в случае нарушения нормального режима работы и ситуации до возникновения аварийной ситуации.

Сигналы от датчиков отправляются на АЦП по физическим линиям. Все типы помех накладываются на эти сигналы (импульсный шум; промышленные помехи частоты; помехи из-за ошибок датчика). Сигнал от АЦП обрабатывается в компьютере, и для устранения влияния помех разрабатываются алгоритмы управления. Это включает в себя алгоритмы проверки, фильтрации и проверки технологических ограничений [21].

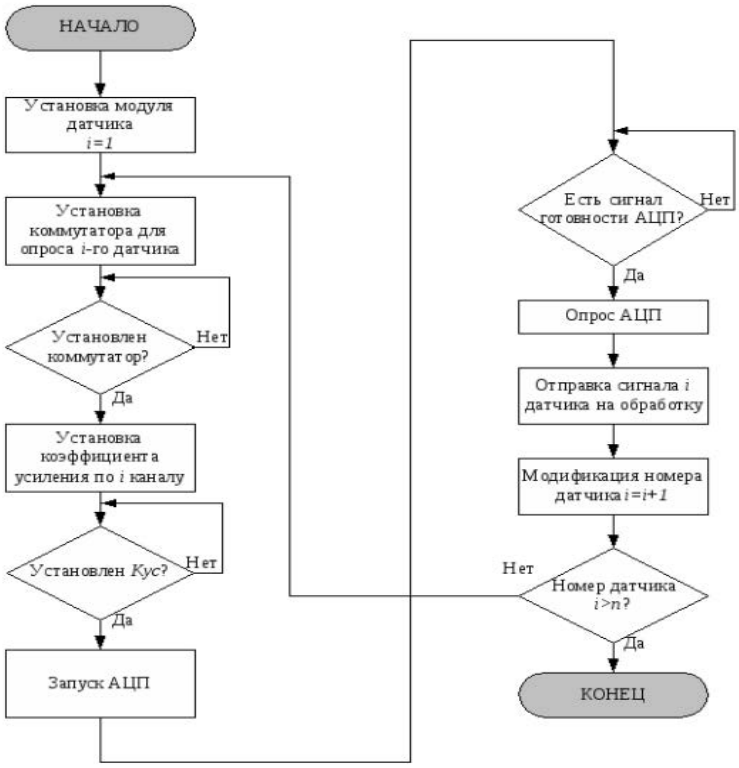

Рисунок 11 – Блок-схема работы интерфейса с несколькими датчиками

2.4 Разработка алгоритма работы интерфейса с протоколом ModBus

Чтобы датчик мог передавать данные на персональный компьютер, необходимо, чтобы компьютер мог принимать эти данные. Для этого создается алгоритм, с помощью которого протокол передачи данных ModBus будет «общаться» с нашим компьютером. В упрощенном виде, структура запросов Modbus состоит из кода функции (чтение/запись), и данных, которые нужно считать или записать. При этом, коды функции различаются для разных типов данных.

Рисунок 12 – Регистры и функции ModBus

Рассмотрим каждый их них.

1) discrete inputs — дискретные входы устройства, доступны только для чтения. Диапазон адресов регистров: с 10001 по 19999. Имеют функцию «02» — чтение группы регистров;

2) coils — дискретные выходы устройства, или внутренние значения. Доступны для чтения и записи. Диапазон адресов регистров: с 20001 по 29999. Имеет функции: «01» — чтения группы регистров, «05» — запись одного регистра, «15» — запись группы регистров;

3) input registers — 16-битные входы устройства. Доступны только для чтения. Диапазон адресов регистров: с 30001 по 39999. Имеют функцию: «04» — чтение группы регистров;

4) holding registers — 16-битные выходы устройства, либо внутренние значения. Доступны для чтения и записи. Диапазон адресов регистров: с 40001 по 49999 [22].

Несмотря на названия, входы и выходы могут на самом деле являться внутренними переменными, хранить счетчики, флаги, или быть управляющими триггерами. В разных устройствах могут быть задействованы разные диапазоны регистров, или же все сразу, но мы будем использовать только один.

Сформулируем рабочий порядок ведущего (MASTER) и подчиненного (SLAVE) устройств в краткой форме. Ведущее устройство, котое в данный момент выполняет целевую программу, запрашивает данные удаленного модуля или устанавливает выходы удаленного модуля в определенное состояние. Ведущее устройство (в дальнейшем называемое ведущим или MASTER) генерирует сообщение Modbus, которое обычно содержит следующее: адрес ведомого устройства, код команды, дополнительные данные и контрольную сумму.

После выполнения команды подчиненное (ведомое) устройство генерирует ответное сообщение, которое обычно содержит адрес подчиненного устройства (то есть его собственный адрес), код команды, дополнительные данные и контрольную сумму. Сгенерированное сообщение передается ведущему устройству (состояние 2.4), и после успешного получения ответа от ведомого устройства (состояние 1.4) ведущее устройство продолжает работать в соответствии с основными программами, а ведомое устройство продолжает ждать следующую команду. Транзакция Modbus завершена.

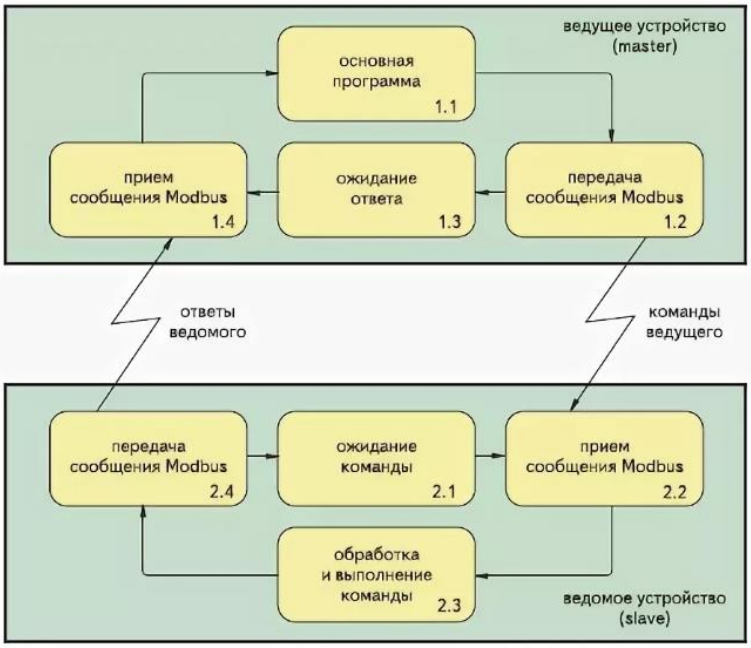

Рисунок 13 – Обобщенный алгоритм обмена данными между ведущим и ведомым устройством

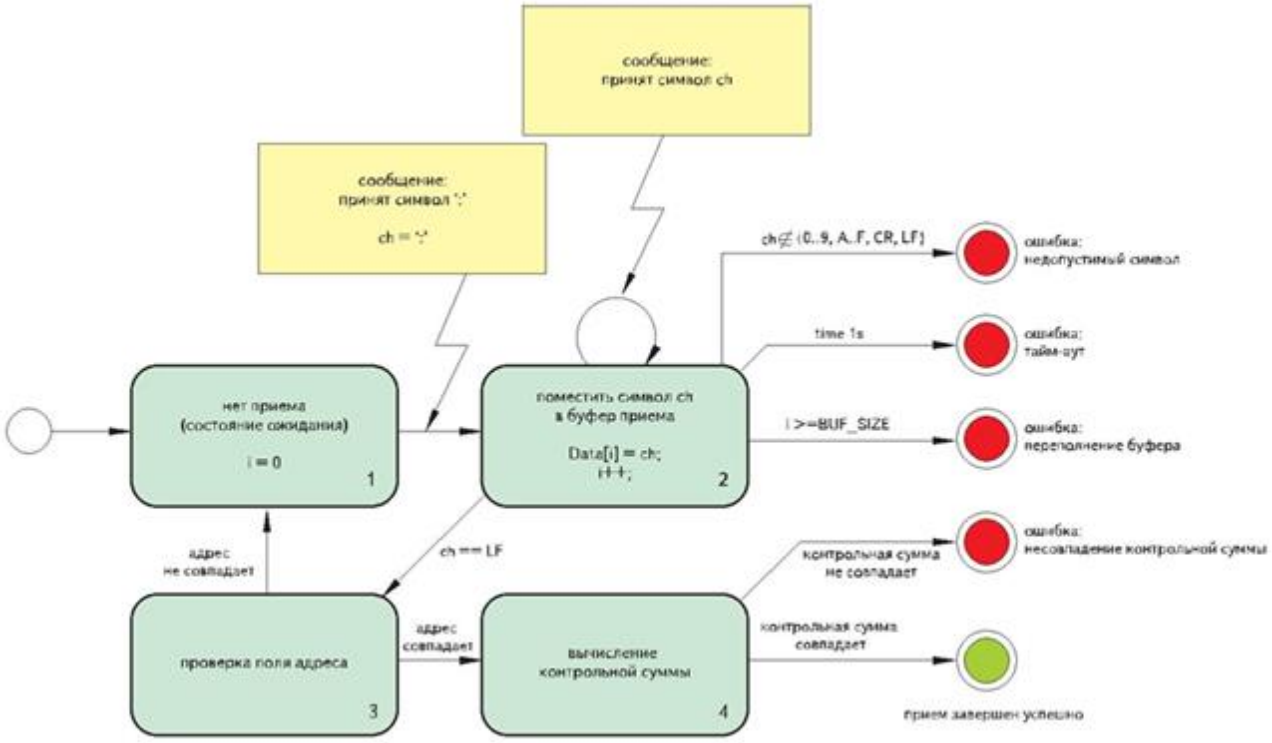

Рисунок 14 – Конечный алгоритм приема сообщений от ModBus

2.5 Выводы по разделу 2

В данном разделе были разработаны алгоритмы работы графического интерфейса и протокола передачи данных с графическим интерфейсом. Также было приведено подробное описание всех действующих окон и их функций. Были приведены результаты компилирования графического интерфейса.

Для реализации разработанного графического интерфейса для датчика температуры с функцией метрологического самоконтроля необходимо протестировать программный продукт на опытных образцах датчиков.

3 ОТЛАДКА ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ ОПЫТНЫХ ОБРАЗЦОВ ДАТЧИКОВ ТЕМПЕРАТУРЫ

3.1 Отладка графического интерфейса на корректность работы кода, функционала и визуальной составляющей

Отладка программного обеспечения – это процесс поиска и устранения дефектов или проблем в компьютерной программе, которые мешают правильной работе программного обеспечения или системы. Тактика отладки может включать интерактивную отладку, анализ потока управления, модульное тестирование, интеграционное тестирование, анализ файла журнала, мониторинг на уровне приложения или системы, дамп памяти и профилирование [23].

Тестирование – это процесс, позволяющий оценить функциональность программного приложения с целью выяснить, соответствует ли разработанное программное обеспечение указанным требованиям или нет, и выявить дефекты, чтобы убедиться, что продукт не имеет дефектов для производства качественного продукта [24].

Ручное тестирование - это процесс тестирования программного обеспечения вручную, чтобы узнать больше о нем, найти то, что работает, а что нет. Обычно это включает в себя проверку всех функций, указанных в документах с требованиями, но часто также включает в себя тестировщиков, которые пробуют программное обеспечение с точки зрения конечного пользователя. Планы ручного тестирования варьируются от сценариев с полностью написанными сценариями, дающими тестировщикам подробные шаги и ожидаемые результаты, вплоть до руководств высокого уровня, которые управляют сеансами исследовательского тестирования. На рынке имеется множество сложных инструментов, помогающих в ручном тестировании, но если вы хотите начать с простого и гибкого места.

Автоматическое тестирование - это процесс тестирования программного обеспечения с использованием средства автоматизации для поиска дефектов. В этом процессе тестировщики выполняют сценарии тестирования и автоматически генерируют результаты теста с помощью средств автоматизации.

Для проверки работоспособности разработанного мною интерфейса мы будем использовать внутренний отладчик приложений, встроенный в среду разработки Python. Отладка программы будет производиться в пошаговом режиме, чтобы в случае обнаружения ошибки сразу же её исправить. Приведенные коды являются лишь частью основного кода графического интерфейса [26]. Весь программный код будет приведен в Приложении А.

Первым шагом будет проверка корректного запуска программы, чтобы понять сможет ли интерфейс запуститься на устройстве, котором будет эксплуатироваться. Для этого мы используем ярлык, который появиться на рабочем столе после установки нашего интерфейса на персональный компьютер. Результатом успешного запуска будет открывшееся окно программы, с основными его функциями.

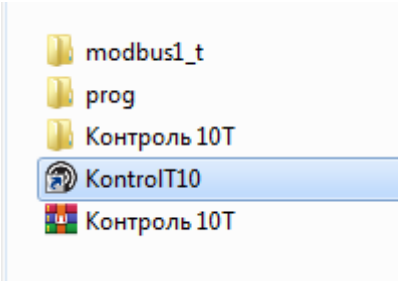

Рисунок 15 – Ярлык запуска программы

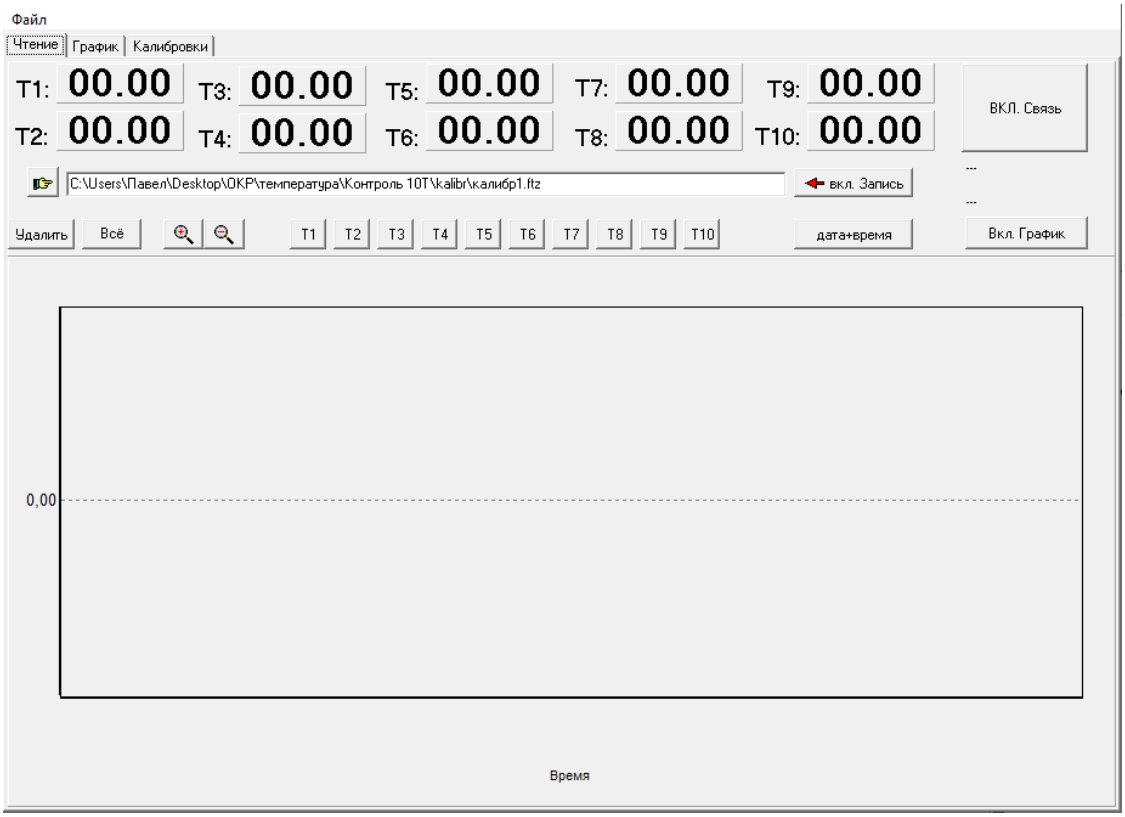

Рисунок 16 – Результат правильного запуска

Следующим этапом будет проверка работоспособности вкладки «Файл» в котором будет производиться основная настройка портов, скоростей передачи данных, основных констант и переменных. При нажатии на вкладку «Файл» будет открываться шторка для выбора режима настройки портов, а затем при нажатии на «Установки COM порта» будет открываться окно «Setup» для подробной настройки.

Если данный программный код, после проверки и компоновки внутри встроенного отладчика приложений среды Python, работает корректно, мы получим работоспособное окно «Файл» со всеми функциями описанными выше.

```
#pragma package(smart_init)
#pragma link "UTPSV_COMport"
#pragma resource "*.dfm"
TForm1 *Form1;
bool Slave[16];//подключенные устройства
bool process; //переменная,показывающая , что выполняется запрос
```

```
const KolReg=200;//Количество регистров , которыми можно оперировать
union {int t; float f; char byte[4];} Digit;
union { float f; char byte[4];} Digit_p[65];
 unsigned char KolSlave;//Количество подключенных слэйвов
 unsigned char Data[25]; //Массив для отправки данных на сом-порт
 unsigned char Dat[25];
 unsigned short a ;
 unsigned char kolerr,par[60];
 unsigned long kolrd;
 unsigned char rdbuf[300];//Массив для принятия данных с сом-порта
 unsigned short CRC16(unsigned char puchMsg[256],int usDataLen);
 int  RKolByte(unsigned short Kol,unsigned char Fun);
 int i,j,fl,fl1,parol;
 unsigned char pi,lim,ph,pp=1,k;
 AnsiString str;
 TPSV_COMport *cp1;
```

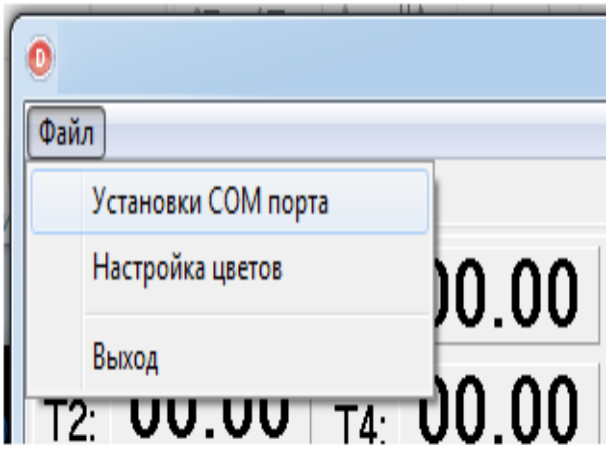

Рисунок 17 – Шторка с выбором функции «Установки COM порта»

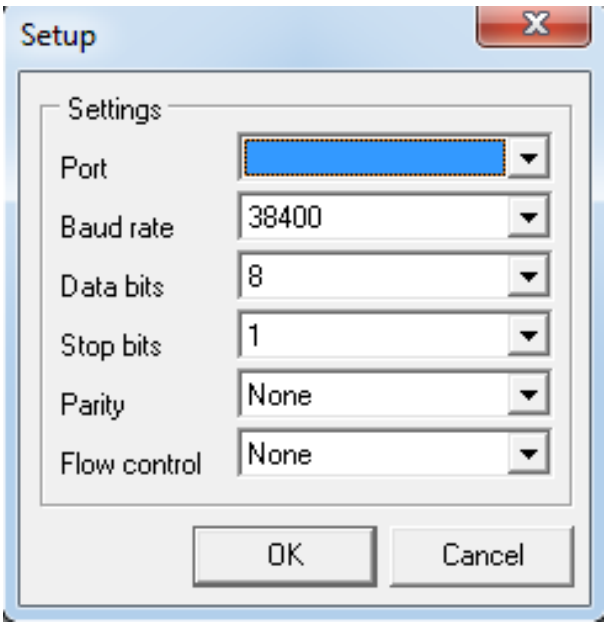

Рисунок 18 – Окно «Setup»

После настройки портов следует перейти во вкладку «Калибровка» и указать файл с установленными параметрами калибровки температуры на датчике. При успешном выполнении данного этапа установятся необходимые нам значения температур.

Если данный программный код, после проверки и компоновки внутри встроенного отладчика приложений среды Python, работает корректно, мы получим работоспособное окно «Калибровка» со всеми функциями, описанными выше.

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
  cp1->ComPCreate();
  cp1->ComPClearBuff();
  KolSlave=1;
  Timer2->Enabled=true;
}
//---------------------------------------------------------------------------
//Процедура принятия данных с порта и проверка пришедших данных
void __fastcall TForm1::WriteData(unsigned char KByte,unsigned char Funk )
{
```

```
 char *p;
 AnsiString *p1;
 cp1->ComPClearBuff();  //Очистка буфера ком-порта
 cp1->RTScontrol=Disable_;
 cp1->ComPSetParam();
 cp1->DelayMs(200); //Тайм-аут 0,2 сек
 try
 {
 cp1->ComPRead(rdbuf,KByte);
 }
 catch(...) //Обработка исключения - ответ не пришел => неадрессованных
устройств нет
 {
  ShowMessage(" Нет ответа ");
    kolerr++;
   return;
 }
 if(rdbuf[0]!=Data[0]){kolerr++;
 ShowMessage("  Slave другой ");
  goto wd1;}
 else
    {if(rdbuf[1]==Data[1]+0x80)
      {a=CRC16(rdbuf,4);
       if (a!=0)
         {ShowMessage(" не совпадение контрольной суммы CRC16 приема 5байт ");
          kolerr++; goto wd1;
         }
       else
         {ShowMessage(" ТВ: ошибка при приеме ");
          kolerr++; goto wd1;
         }
      }
      if(rdbuf[1]==Data[1])
      {
      a=CRC16(rdbuf,KByte-1);
      if (a!=0)
      {
       ShowMessage(" не совпадение контрольной суммы CRC16 при приеме");
       kolerr++; goto wd1;
      }
      else
      {
```

```
        if(rdbuf[0]>=KolSlave)KolSlave=rdbuf[0]+1;
       kolerr=0;
       switch (Funk)  //Выбор выполненной функции
        {
        case 16 :  // запись в регистр
        if
(rdbuf[0]==Data[0]&&rdbuf[1]==Data[1]&&rdbuf[2]==Data[2]&&rdbuf[3]==Data[3]&
&rdbuf[4]==Data[4]&&rdbuf[5]==Data[5])

        break;
        case 3 :  //чтение из регистра
           p=&rdbuf[3];
     //Запись пришедших данных в массив Digit_p (float)
     // rdbuf[2] -  количество байт данных в ответе (без 2х первых байт и без CRC16)
           for (i=0;i<=(rdbuf[2]/4);i++)
            for (j=3;j>=0;j--)
             {
              Digit_p[i].byte[j]=*p++;
             } ;
     //Вывод пришедших данных
           Memo1->Lines->Add("");
           Memo1->Lines->Add("Транзакция от :"+str+" Slave "+IntToStr(rdbuf[0]));
//     ShowMessage(Data[3]);
           p1=&Edinicy[((Data[3]-1)/2)];
           for(i=0; i<=(rdbuf[2]/4-1); i++)
             {
               Memo1->Lines-
>Add((*p1++)+FloatToStr(Digit_p[i].f));//+Edinicy[n].c_str());
             }
          break;
        }
       }
      }
        }
```

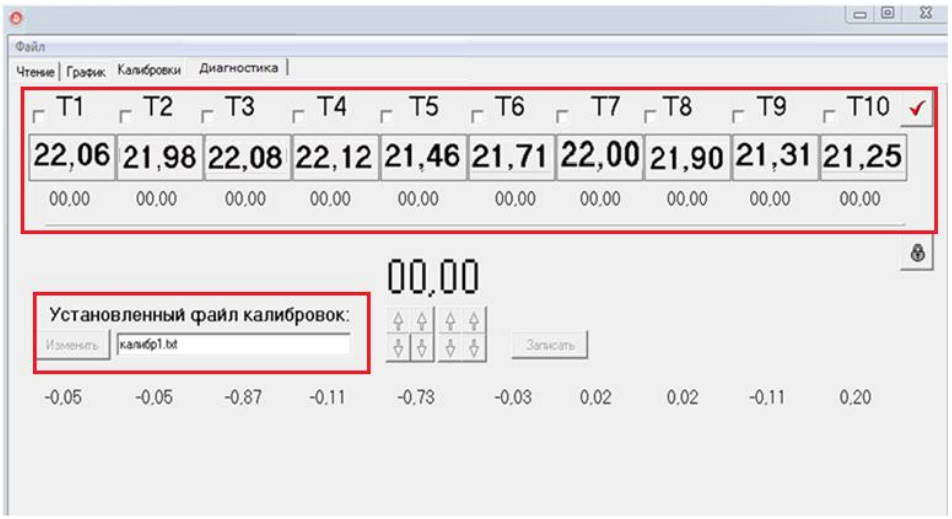

Рисунок 19 – Корректная работа вкладки «Калибровка»

Следующим этапом будет установка связи с датчиком температуры. Для этого в интерфейсе нужно перейти во вкладку «Чтение» и нажать кнопку «ВКЛ.Связь». В случае правильной работы программы, в рабочем пространстве появятся графики отображающие температуру и её изменение с течением времени, а также значения температур на одном или нескольких датчиках.

Если данный программный код, после проверки и компоновки внутри встроенного отладчика приложений среды Python, работает корректно, мы получим работоспособное окно «Чтение» со всеми функциями описанными выше.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 unsigned short n;
 fl=0;
else
 {
 if (!process)   //Если не обрабатывается запрос
 {
 process=true;
 StrCopy(Data,"");
 try
 {
 //Data[...]-данные,отправляемые в запросе
```

```
 // Data[0] - адрес слэйва  8 бит
 Data[0]=(ComboBox4->ItemIndex);
  //Data[1] – Функция ВКЛ.Связь
 Data[1] = 16;
 if((ComboBox1->ItemIndex)<34)n=(ComboBox1->ItemIndex)*2+21;
 else n=((ComboBox1->ItemIndex)-34)*2+117;
   // Data[2&3] - номер  регистра , в который происходит вызов графиков
 Data[2]=n>>8;
 Data[3]=n;
Digit.f = StrToFloat(Edit2->Text.c_str());
  cp1->RTScontrol=Enable_;
  cp1->ComPSetParam();
  cp1->ComPClearBuff();
  for(k=0; k<13; k++)
        {cp1->ComPWrite(Dat,1);
         Dat[0]=Dat[k+1];
         }
//  cp1->ComPWrite(Data,13);  //Отправка запроса на сом-порт
  }
  catch (eComPortExcept& E  {
    cp1->ComPortExceptShow(E);
    fl=1;
  };
   if (fl!=1)
  {  cp1->DelayMs( 3);
  WriteData(8,16);//Вызов процедуры принятия ответа
  if(kolerr==0)process=false;
    else
      {//if(kolerr<3)goto err7;
       //else
       //ShowMessage ("Попытки связаться с датчиком не удались");
      }
  }
 process=false;
         }
```

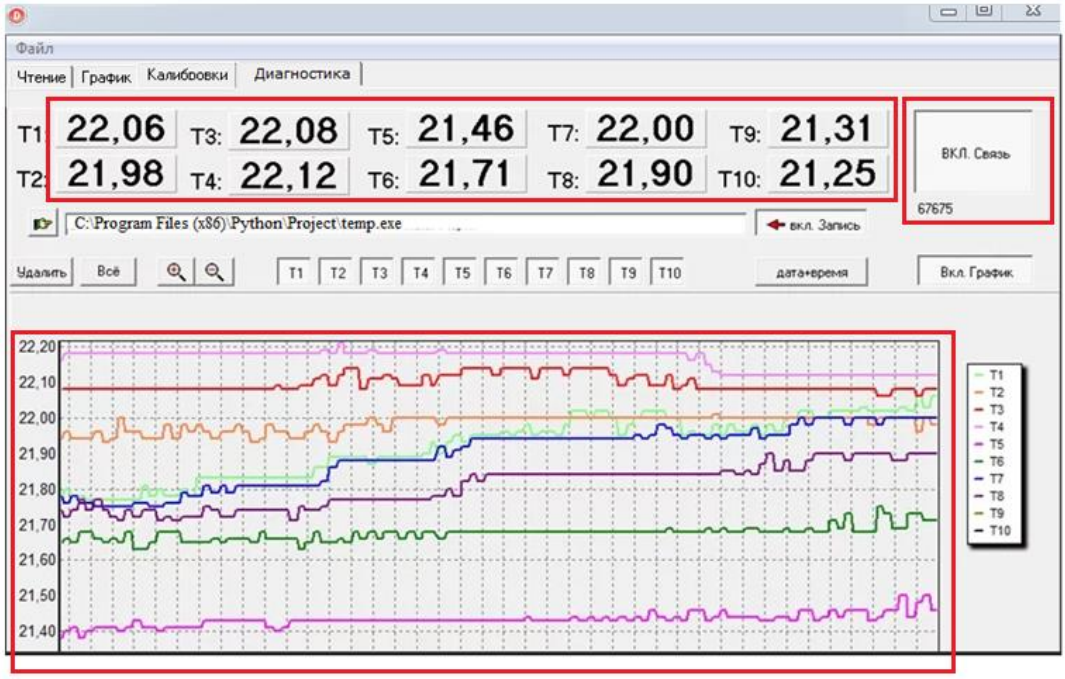

Рисунок 20 – Отображение успешной связи с датчиком температуры

Далее проведем отладку окна «График». При правильной работе программы, в этой вкладке должны отображаться совмещенные графики температур, как они изменяются со временем относительно друг друга.

Если данный программный код, после проверки и компоновки внутри встроенного отладчика приложений среды Python, работает корректно, мы получим работоспособное окно «График» со всеми функциями, описанными выше.

```
AnsiString Edinicy[]={"Ur1= ","Ur2= ","r1= ","r2= ","t1= ","t2= ","t= ","Ut= ",
 "diag= ","error= ","a0= ","a1= ","a2= ","a3= ","a4= ","a5= ","a6= ","a7= ",
 "a8= ","a9= ","b0= ","b1= ","b2= ","b3= ","b4= ","b5= ","b6= ","b7= ","b8= ",
 "b9= ","t10= ","t11= ","t20= ","t21= ","dr= ","n0= ","n10= ","rmin= ","rmax= ",
 "tdop= ","muct= ","mbr= ","mmct= ","mslave= "};
static unsigned
 #define VTIMEOUT 500
```

```
#define APPID_STR_LONG 25   // максимальное число символов в строке имени
приложения, использующего COM-порт
#define NUMBYTES 1          // количество байт по умолчанию
#define STRVERSION "Версия компонента 1.11"
#define VERSION 11;
static inline void ValidCtrCheck(TPSV_COMport *)
        new TPSV_COMport(NULL);
__fastcall TPSV_COMport::TPSV_COMport(TComponent* Owner)
        : TComponent(Owner)              // Конструктор
{
  lpWriteBuff = NULL;
  lpReadBuff = NULL;
  FpName = "COM2";      // Значения полей по умолчанию
  FBaudRate = 2400;
  FByteSize = 8;
  FParity = 0;
  FStopBits = 2;
  FParityEnabled = FALSE;
  FDTRcontrol = DTR_CONTROL_ENABLE;
  FRTScontrol = RTS_CONTROL_ENABLE;
  FReadIntervalTimeout = 0;
  FReadTotalTimeoutMultiplier = TIMEOUT;
  FReadTotalTimeoutConstant = 0;
  FNumBytesToWrite = NUMBYTES;
  FNumBytesToRead = NUMBYTES;
  FNumBytesWritten = 0;
  FNumBytesRead = 0;
}
```

Рисунок 21 – Правильная работа вкладки «График»

Последним шагов в отладке нашего графического интерфейса будет проверка работоспособности вкладки «Диагностика». При правильной работе данного окна нам будет отображаться выбор порта, период опроса датчика, максимальная температура, наименование функции, где сохранены все данные по датчику, включая текущую температуру. Также, должно отображаться окно, где будут показаны текущие температуры, код от АЦП, коэффициент изменения сопротивления.

Если данный программный код, после проверки и компоновки внутри встроенного отладчика приложений среды Python, работает корректно, мы получим работоспособное окно «График» со всеми функциями, описанными выше.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 unsigned short n;
 fl=0;
 if (!process)  //Если не обрабатывается запрос
 {
 process=true;
 StrCopy(Data,"");
 try
 {
 //Data[...]-данные,отправляемые в запросе
 // Data[0] - адрес слэйва  8 бит
 Data[0]=(ComboBox3->ItemIndex);
 // Data[1] - номер функции  8 бит
 Data[1] = 3;
 Data[3]=(ComboBox2->ItemIndex)*2+1;
 Data[2]=0;
 Data[4]=0;
 try
 {
 Data[5] = StrToInt(Edit4->Text.c_str());
 }
 catch(...)
 {
  ShowMessage("Количество значений введено неправильно");
  fl=1;
 };
 // Data[6&7] - контрольная сумма 16 бит
 a=CRC16(Data,5);
 Data[6]=a >> 8;
 Data[7]=a;
err3:
  for(k=0; k<8; k++) Dat[k]=Data[k];
 if (Data[5]>KolReg)
  ShowMessage("Количество значений введено неправильно");
 else
 { kolerr=0;
  try
  {
  cp1->RTScontrol=Enable_;
  cp1->ComPSetParam();
  cp1->ComPClearBuff();
```

```
  for(k=0; k<8; k++)
        {cp1->ComPWrite(Dat,1);
         Dat[0]=Dat[k+1];
         }
  //  cp1->ComPWrite(Data,8);  //Отправка запроса на сом-порт (8 байт)
  }
  catch (eComPortExcept& E)//Обработка исключения в случае неудачной записи в
сом-порт
  {
    cp1->ComPortExceptShow(E);
    fl=1;
  };
  if (fl!=1) //Если не было ошибки записи на сом-порт
  {
    cp1->DelayMs( 3);
    WriteData(RKolByte(Data[5],3),3);//Вызов процедуры принятия ответа
    if(kolerr==0)process=false;
    else
      {//if(kolerr<10)goto err3;
       //else
      // ShowMessage ("Попытки прочитать не удались");
       }
  };
 };
  process=false;
}
catch (...) //Обработка исключения в случае неправильного ввода числа
{
 ShowMessage("Ошибка ввода числа , повторите ввод") ;
 process=false;
 } ;
 }
 else
  ShowMessage("Происходит обработка запроса,попробуйте чуть позже");
}
```

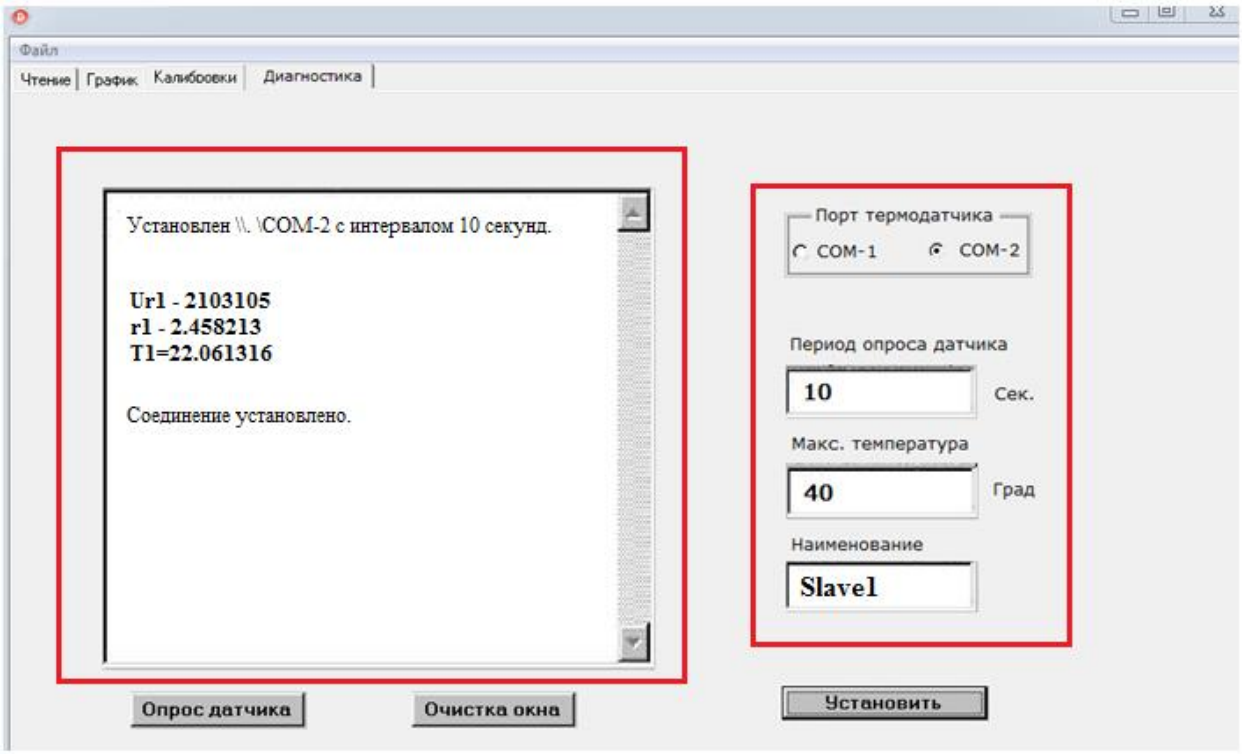

Рисунок 22 – Успешная работа вкладки «Диагностика»

3.2 Выводы по разделу 3

В данном разделе была произведена отладка разработанного графического интерфейса с его функциональным наполнением, а также проверка всех её частей на ошибки, включая программный код. Данный раздел показывает, что в коде программы или его визуальной части ошибок не обнаружено, все составляющие и функционал работают исправно, и интерфейс полностью готов к использованию.

4 ПРОВЕДЕНИЕ КОРРЕКТИРОВКИ ПРОГРАММЫ ПО РЕЗУЛЬТАТАМ ИСПЫТАНИЙ

4.1 Описание проведения тестирования программы

После успешной отладки графического интерфейса и исправления всех возникших ошибок необходимо переходить к тестированию созданной программы на разработанных проектной командой датчиков температуры с функцией метрологического самоконтроля. Для этого поставим цели какие мы хотим достичь в ходе тестирования:

1) отображение в программе всех подключенных датчиков;
2) корректная передача данных из программы в АЦП датчиков;
3) корректный приём данных поступающих с датчиков;
4) корректное отображение всех данных во всех вкладках («Чтение», «Графики») графического интерфейса.

Первый пункт подразумевает за собой отображение в программе нужного нам количества подключенных к компьютеру датчиков, ведь наша программа позволяет производить настройку и калибровку, а также считывание данных сразу с нескольких устройств.

Пункт под номером два подразумевает за собой правильное отображение всех и вывод всех запрашиваемых данных (кодов, сопротивления, температуры и т.д.).

Третий пункт подразумевает за собой корректную передачу данных из нашего графического интерфейса в АЦП для калибровки датчиков.

Под последним пунктом будем понимать, что вся информация, которая будет поступать с датчиков, будет отображаться в отведенных для этого вкладках разработанного программного обеспечения.

4.2 Выполнение тестирования программного обеспечения

4.2.1 Отображение в программе всех подключенных датчиков

Для работы с датчиками температуры нам необходимо убедиться, что разработанный нами графический интерфейс может определять количество подключенных к нему устройств. Для этого, сначала, необходимо запустить программу нажатием на ярлык, находящемся на рабочем столе, и после этого подключить в USB-порты необходимое количество устройств.

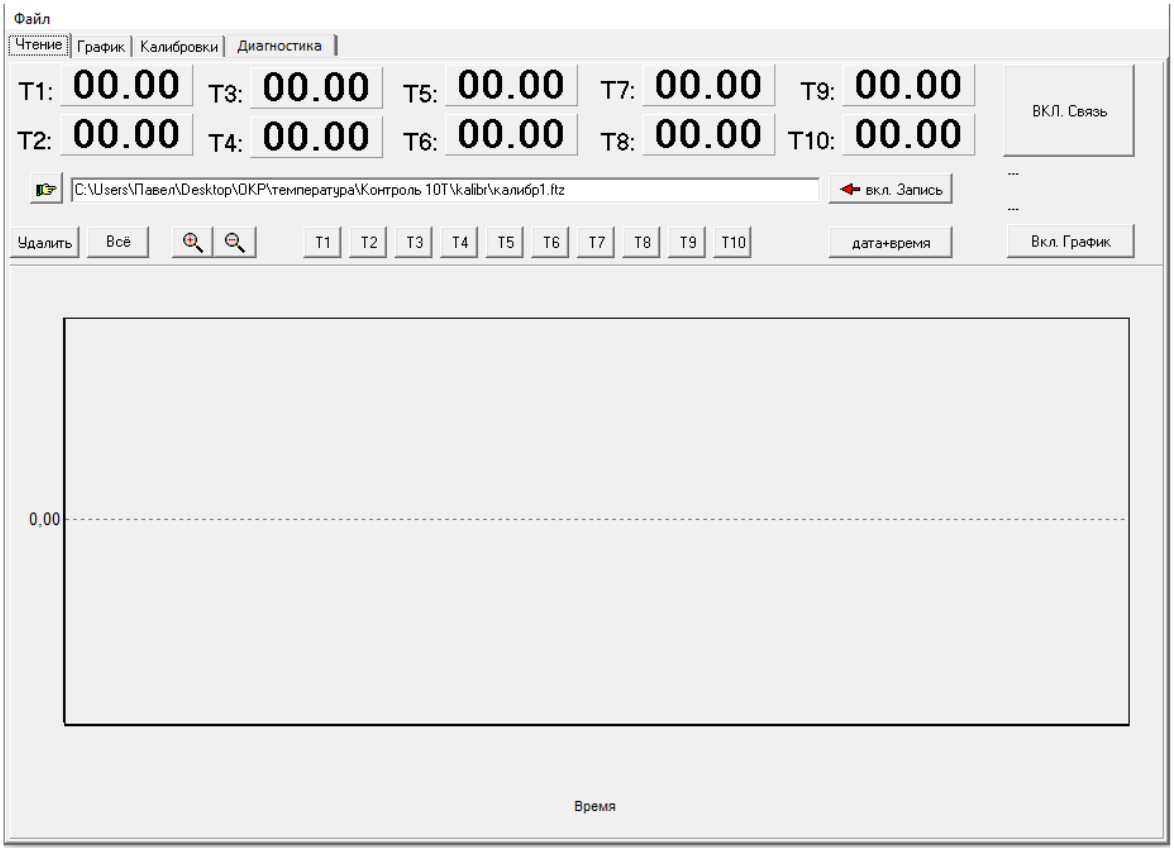

Рисунок 23 – Программа без подключенных устройств

Можем заметить, что до подключения датчиков вкладка остаётся практически пустой, за исключением встроенных кнопок и рабочей области.

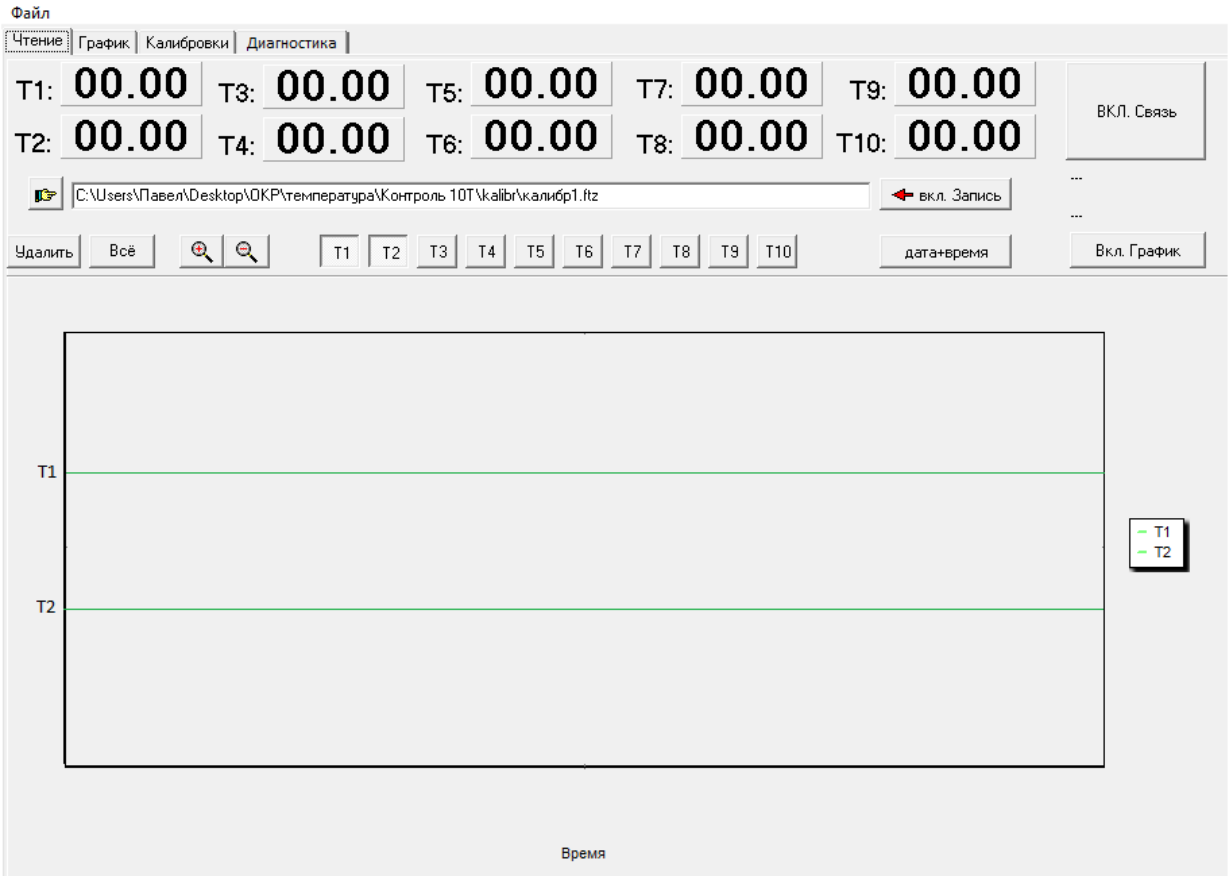

Рисунок 24 – Программа после подключения 2-х датчиков, вкладка «Чтение»

Как мы можем заметить, после подключения двух датчиков в рабочей области появилось две полосы отвечающие за отображение температуры каждого из датчиков, но так как датчики еще не калиброваны, то наш интерфейс показывает нулевую активность, то есть сплошные линии. Если мы перейдем в другую вкладку, то программа должна показать аналогичный результат. Для этого попробуем перейти в каждую из вкладок.

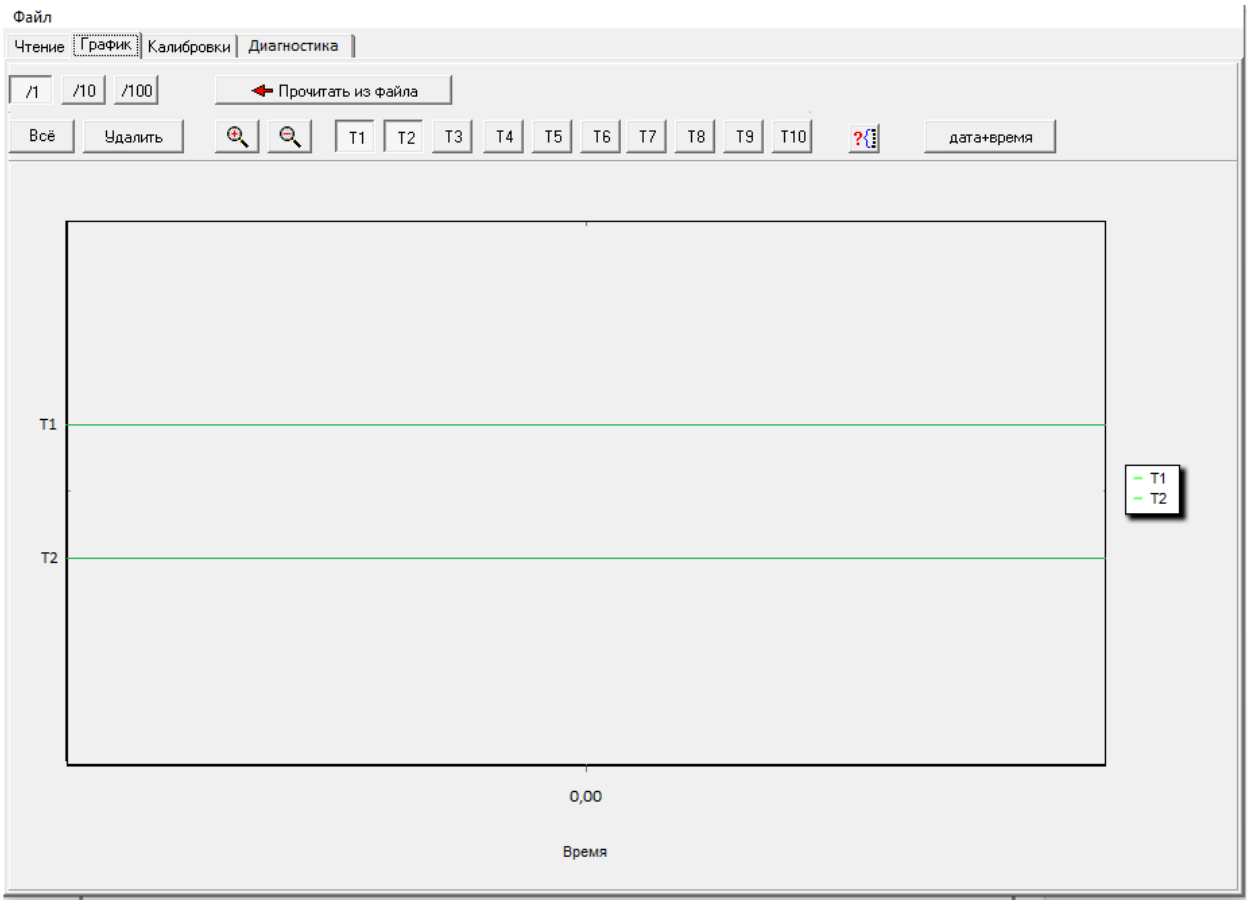

Рисунок 25 – Вкладка «График» после подключения 2-х датчиков

Во вкладке «Калибровки» подключенное количество датчиков обозначается галочками автоматически.

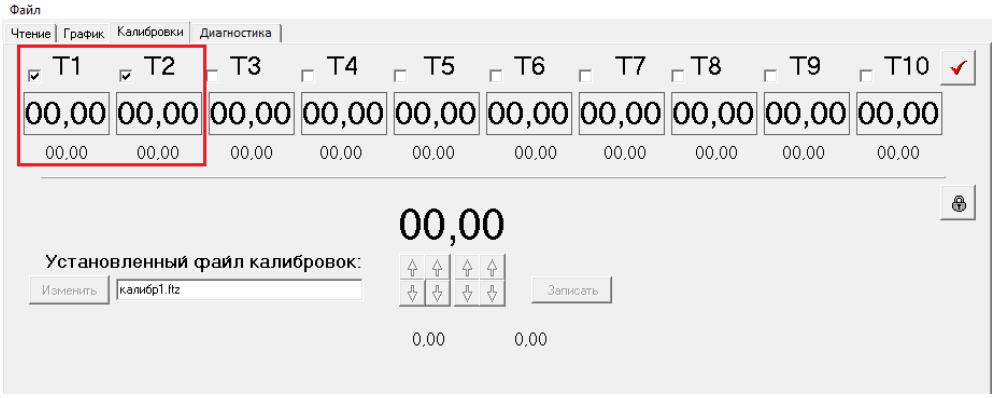

Рисунок 26 – Вкладка «Калибровки» после подключения 2-х датчиков

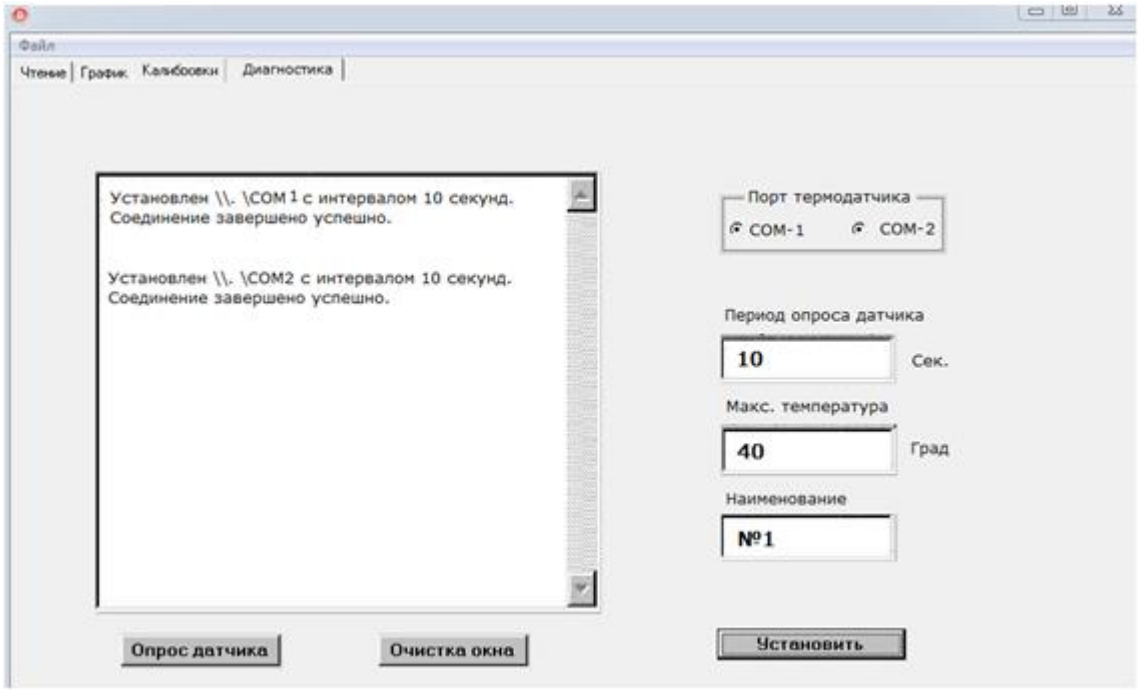

Рисунок 27 – Вкладка «Диагностика» после подключения 2-х датчиков

Как мы можем заметить в рабочем окне вкладки описано лишь то, что связь с датчиками установлена и период опроса. Это связано с тем что датчики еще не калиброваны, в дальнейшем здесь будет описана вся нужная нам информация. Также можно заметить, что в месте выбора порта термодатчика выбраны 2 порта одновременно. Это связано с тем, что в случае подключения нескольких датчиков выбираются сразу два порта, но если подключить только один, то и выберется только один.

4.2.2 Корректная передача данных из программы в АЦП датчиков

После того как мы удостоверились, что наша программа корректно отображает количество подключенных датчиков, перейдём к настройке датчиков. Для этого зайдем во вкладку «Калибровки» и нажмем на кнопку изменить в графе «Установочный файл калибровок»

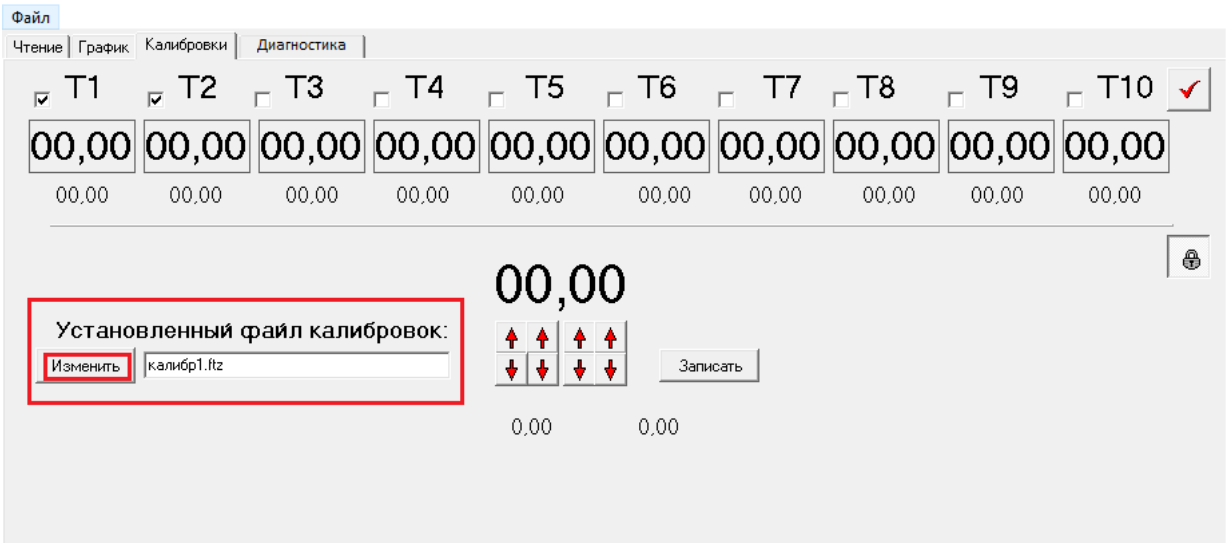

Рисунок 28 – Начало калибровки

После этого перед нами откроется окно, где мы сможем полностью настроить наши датчики. Чтобы не записывать каждую переменную отдельно, для удобства была создана подпрограмма, которая после компиляции сформирует файл формата « .ftz» для каждого датчика.

Подпрограмма имеет две вкладки «Запись данных», которая откроется автоматически, и «Чтение данных», но также предусмотрена возможность переключаться между вкладками внутри подпрограммы. Рассмотрим первую вкладку «Запись данных». Здесь мы выбираем какой датчик необходимо откалибровать, для этого используется графа «Slave». Также мы можем заметить графу «Переменная для записи», в которой можно выбрать коэффициенты для полиномов, коды ЦАП и так далее, и место для записи значений тех или иных переменных. Нажав на клавишу «Отправить данные» все значения в микроконтроллер.

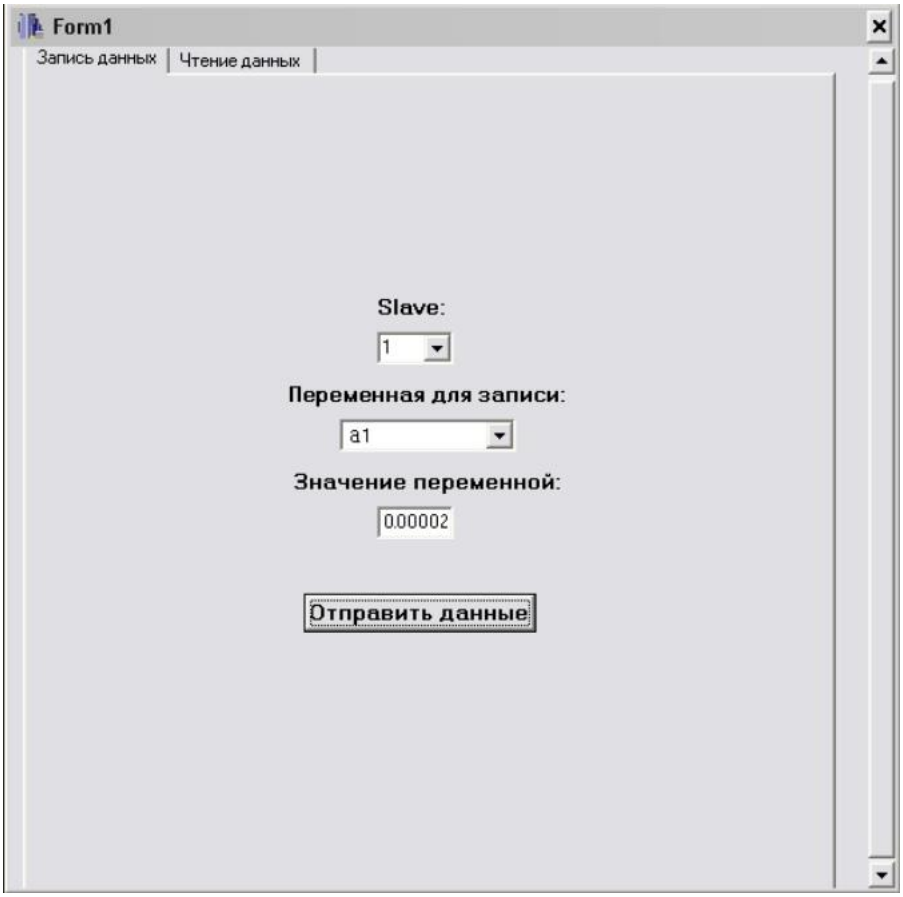

Рисунок 29 – Корректная работа вкладки «Запись данных» подпрограммы

Описание работы вкладки подпрограммы «Чтение данных» будет описано позже.

4.2.3 Корректный приём данных поступающих с датчиков

После настройки и калибровки датчиков, необходимо проверить достоверность поступаемых с них данных в нашей программе. Для этого переходим во вкладку «Диагностика» и, указав все необходимые нам параметры, нажать на кнопку «Установить». Так как в программе предусмотрена работа сразу с несколькими датчиками, то опрос будет происходить поочерёдно и результат будет описан по каждому в отдельности. Для тестирования мы используем 2 датчика.

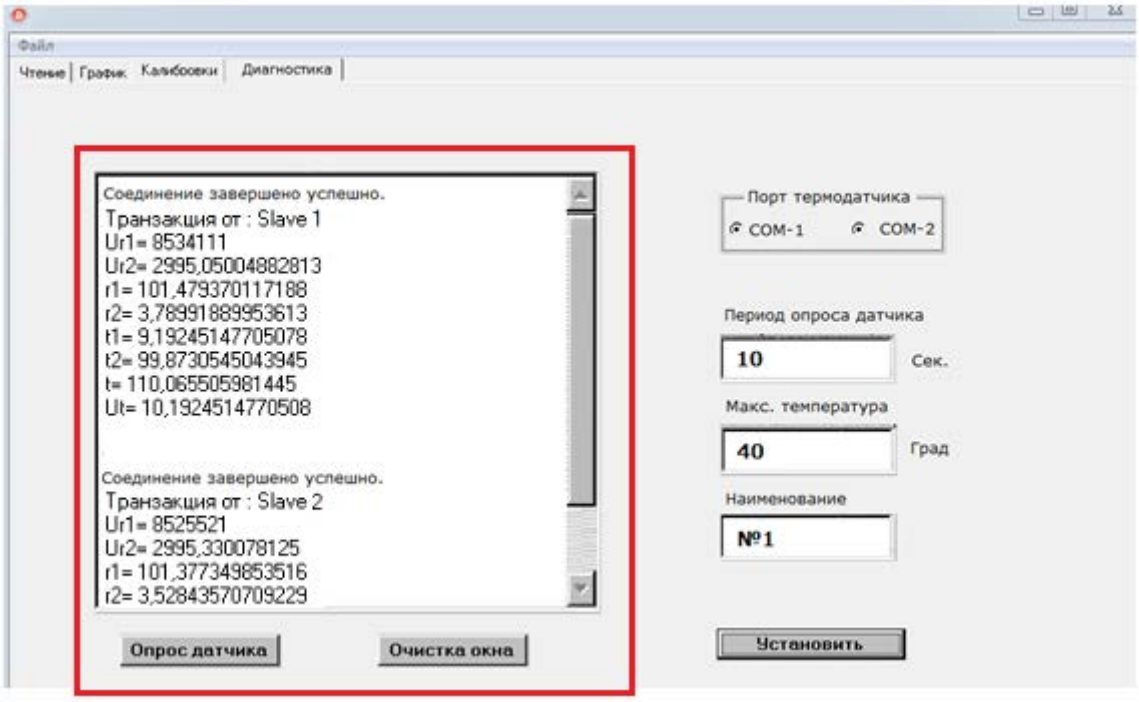

Рисунок 30 – Результат опроса двух датчиков

В окне показаны все, опрашиваемые программой, данные с датчика, где: Ur1 – выходной код сигма-дельта АЦП от измерения сопротивления ТС, Ur2 – выходной код встроенного в микроконтроллер АЦП от преобразования температуры окружающей ЭП среды, r1 – вычисленное по полиному сопротивление ТС, Ом, r2 – вычисленная температура ТС, ºС. t1 – вычисленная функция диагностики γ, t2 – значение сопротивления ТС в момент подачи тока саморазогрева, Ом, t – значение сопротивления ТС в момент отключения тока саморазогрева, Ом, Ut – сопротивление саморазогрева (t - t2), Ом.

Но, если нам необходимо посмотреть результат работы, к примеру, отдельно взятого датчика, и посмотреть, как будут вести себя данные во время калибровки, то мы можем это сделать, вызвав подпрограмму, описанную выше, в которой будет отображено только информация про выбранный датчик. Для этого мы нажимаем на клавишу «Опрос датчика» и откроется вкладка «Чтение данных». Данная вкладка включается автоматически, при вызове подпрограммы из окна «Диагностика», но

также предусмотрена возможность переключаться между вкладками внутри подпрограммы.

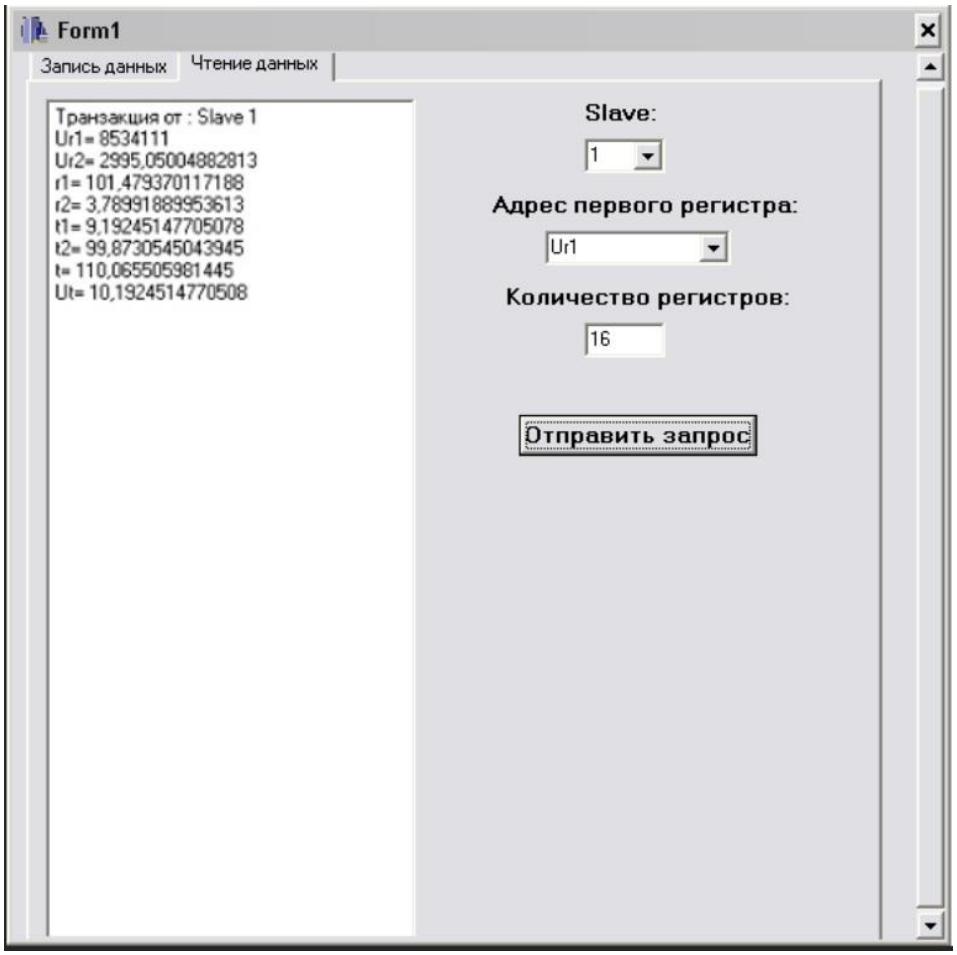

Рисунок 31 – Корректная работа вкладки «Чтение данных» подпрограммы

В рабочем окне мы можем заметить графу «Slave», которая, мы уже знаем, нужна для выбора датчика на опрос, графу «Адрес первого регистра», где можно выбрать, вручную, какие регистры мы будем опрашивать и их значения. Нажав на клавишу «Отправить запрос», нам придет информация, которую мы запрашивали.

4.2.4 Корректное отображение всех данных во всех вкладках «Чтение» и «Графики» графического интерфейса

После диагностики мы можем наблюдать все данные которые поступают с датчиков температуры. На основе их проверим корректность отображаемой информации в рабочих окнах нашего интерфейса. Перейдя во вкладку «Чтение» и нажав на клавишу «ВКЛ.Связь» мы получим графики изменения температуры каждого датчика и сопоставив данные, мы видим, что рабочее окно отображает температуру верно.

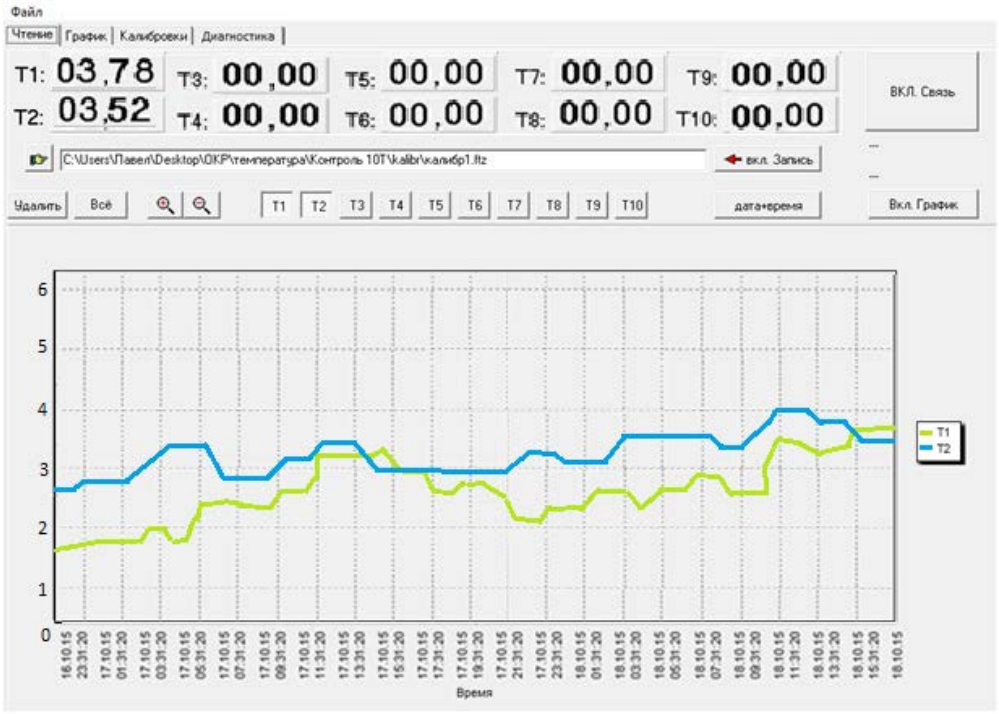

Рисунок 31 – Корректное отображение температур от двух датчиков во вкладке «Чтение»

Также, перейдя во вкладку «Графики» мы можем наблюдать, что при правильной работе программы, в окне должны отображаться совмещенные графики температур, как они изменяются со временем относительно друг друга.

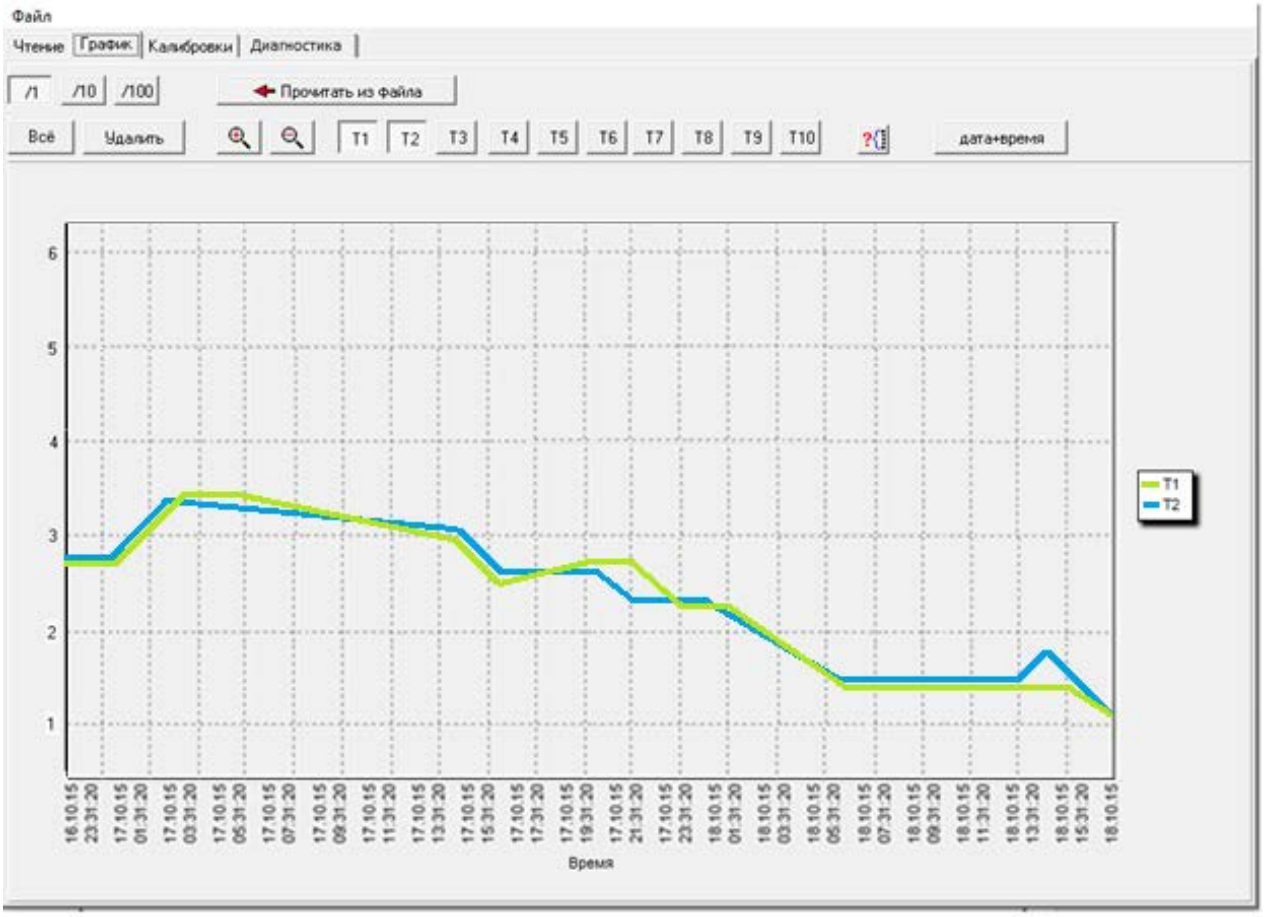

Рисунок 32 – Корректное отображение графиков температур в зависимости друг от друга во вкладке «Графики»

4.3 Выводы по разделу 4

В данном разделе было произведено тестирование разработанного графического интерфейса с его полностью рабочим функционалом для готовых датчиков температуры с функцией метрологического самоконтроля. Раздел показывает, что все части программы работают корректно и без ошибок.

В результате тестирования был сделан вывод о том, что разработанное программное обеспечение имеет ряд преимуществ:

а) удобный и наглядный интерфейс пользователя, не требующий изменения исходного кода для корректировки исходных параметров эксперимента;

б) удобная форма представления параметров датчиков температуры с помощью задания временных отсечений на графиках, увеличение графиков, возможность получения количественных параметров в указанный момент времени.

Графический интерфейс является полностью работоспособным и готовым к эксплуатации.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы были выполнены следующие задачи:

В первом разделе был проведен аналитический обзор литературы и патентов по теме исследования, а также произведен сравнительный анализ существующих способов написания алгоритмов, кодов и создания графических интерфейсов для датчиков температур, был произведен подбор программ. По результатам проведения сравнительного анализа было выявлено, что программный комплекс Python, и протокол передачи данных ModBus являются наиболее подходящим для создания программного обеспечения датчиков температуры для серийного производства.

Во втором разделе были разработаны алгоритмы работы графического интерфейса и протокола передачи данных с графическим интерфейсом. Также было приведено подробное описание всех действующих окон и их функций. Были приведены результаты компилирования графического интерфейса.

В третьем разделе произведена отладка разработанного графического интерфейса с его функциональным наполнением, а также проверка всех её частей на ошибки, включая программный код.

В четвертом разделе был составлен перечень функций, подлежащих тестированию, а именно:

1) отображение в программе всех подключенных датчиков;
2) корректная передача данных из программы в АЦП датчиков;
3) корректный приём данных поступающих с датчиков;
4) корректное отображение всех данных во всех вкладках («Чтение», «Графики») графического интерфейса.

Практическим результатом проделанной работы является разработка графического интерфейса, который даёт возможность пользователю взаимодействовать с датчиком температуры. Созданное программное обеспечение

было протестировано на рабочих образцах датчиков температуры с метрологической функцией самоконтроля. Результаты проведенного тестирования показали, что интерфейс работает корректно и соответствует установленным в ходе разработки требованиям.

Графический интерфейс позволяет:

1) полностью подготовить и настроить датчики температуры к их работе (произвести калибровку);
2) отслеживать в реальном времени показания температуры, которые поступают с датчиков в удобной форме (в виде графиков);
3) контролировать состояние датчиков, путём диагностики.

В результате тестирования был сделан вывод о том, что разработанное программное обеспечение имеет ряд преимуществ:

а) удобный и наглядный интерфейс пользователя, не требующий изменения исходного кода для корректировки исходных параметров эксперимента;

б) удобная форма представления параметров датчиков температуры с помощью задания временных отсечений на графиках, увеличение графиков, возможность получения количественных параметров в указанный момент времени.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1.Агафонов, К.А. Реализация надёжного протокола данных / К.А. Агафонов, Д.С. Порох, А.А. Шалыто. – СПб.: Научно-технический вестник СПбГУ ИТМО. Автоматное программирование, 2006. – 74 с.

2.Бакшеева, Ю.В. Резистивные датчики температуры с метрологическим самоконтролем / Ю.В. Бакшеева, К.В. Сапожникова, Р.Е. Тайманов // Датчики и системы. – 2011. – №4. – С. 62–70.

3.Бен-Ари, М. Языки программирования. Практический сравнительный анализ / М. Бен-Ари. – М.: Изд-во Мир, 2000. – 279 с.

4.Бородин, А.В. Б833 Средства разработки графических интерфейсов пользователя: учебное пособие / А.В. Бородин, А.В. Бородина. – Петрозаводск: Изд-во ПетрГУ, 2012. – 77 с.

5.Интегрированная среда разработки Delphi. – https://vuzlit.ru/817262/integrirovannaya_sreda_razrabotki_delphi.

6.Python – высокоуровневый язык программирования.– https://www.python.org/

7.Программирование на языке Python.– https://books.google.ru/books?hl=ru&lr=&id=mh0bU6NXrBgC&oi=fnd&pg=PR1&dq=python+programming&ots=XAozBvc_hb&sig=mkXzWT8L4v5g5qsFceFCpd2vtC4&redir_esc=y#v=onepage&q=python%20programming&f=false

8.Lazarus как бесплатная среда по разработке ПО.– https://lazarus-rus.ru/

9.Справочный материал по среде Lazarus. – http://goo.kz/loader/load/23969.

10.Денисенко, В. HART-протокол: общие сведения и принципы построения сетей на его основе / В. Денисенко // Записная книжка инженера. – Изд-во СТА-ПРЕСС, 2010. – №3. – С. 8–16.

11.Основы HART-протокола. - https://www.compel.ru/lib/articles/nachnem-s-osnovnogo-chto-takoe-protokol-hart-i-kak-on-rabotaet

12.Методическое пособие по протоколу PROFIBUS. – http://window.edu.ru/resource/005/75005/files/Методичка_profibus.pdf.

13.Протоколы обмена данными. – https://www.master-prom.ru/articles/usefull/control-means/protokoly-obmena-dannymi-profibus-modbus-etc.

14.Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона + CD / Н. Вирт. – М.: Изд-во ДМК Пресс, 2010. – 272 с.

15.Денисенко, В. ModBus-протокол: общие сведения и принципы построения сетей на его основе / В. Денисенко // Записная книжка инженера. – Изд-во СТА-ПРЕСС, 2010. – №4. – С. 5–10.

16.Подключение устройств по протоколу ModBus. – http://doc.rapidscada.net/content/ru/use-cases/modbus-protocol.html.

17.Вонг, У. Основы программирования для "чайников" / У. Вонг. – М.: Изд-во Диалектика, 2007. – 336 с.

18.Вирт, Н. Систематическое программирование. Введение / Н. Вирт. – М.: Изд-во Мир, 1977. – 177 с.

19.Жданов, А.Д. Реализация протокола ModBus / А.Д. Жданов, Т.М. Коломейцева, А.А. Шалыто // Компоненты и технологии. СПб.: Изв-во Файнстрит, 2006. – №4. – С. 78–82

20..Голубцов, М.С. Микроконтроллеры AVR: от простого к сложному / М.С. Голубцов, А.В. Кириченкова. – 2-е изд. – М.: Изд-во СОЛОН-Пресс. Серия «Библиотека инженера» 2006. – 304 с.

21.ГОСТ Р 8.673-2009. Государственная система обеспечения единства измерений. Датчики интеллектуальные и системы измерительные интеллектуальные. Основные термины и определения. – М.: Стандартинформ, 2010. – 7 с.

22.Китаев, Ю.В. Основы программирования микроконтроллеров. Учебное пособие / Ю.В. Китаев. – СПб.: Изд-во СпбГУ ИТМО, 2007. – 107 с.

23.Компьютерное программное обеспечение. –https://kompkimi.ru/sovety/eto-polezno-znat/chto-takoe-kompyuternoe-programmnoe-obespechenie.

24.Компьютерное моделирование физических процессов. – https://studfiles.net/preview/2653995/page:46/.

25.Программирование микроконтроллеров. – http://easyelectronics.ru/avr-uchebnyj-kurs-programmirovanie-na-si-chast-1.html.

26.Программное обеспечение для микроконтроллера. – http://www.chip-dip.ru/product0/874599444.aspx.

27.Прототипирование программных комплексов И.В. Оселедец. – http://agora.guru.ru/abrau2012/pdf/404.pdf .

28.Рафикумазан, М. Микропроцессоры и машинное проектирование микропроцессорных систем / М, Рафикумазан; пер. с англ. – М.: Мир, 1988. – 288 с.

29.РМГ 74-2004. Рекомендации по межгосударственной стандартизации. Государственная система обеспечения единства измерений. Методы определения межповерочных и межкалибровочных интервалов средств измерений. – М.: Изд-во Стандартинформ, 2006. – 21 с.

30.Сетевые технологии. –http://сетиэвм.рф/index.php/lektsii/protokoly/kanalnyj-uroven/protokoly/80-profibus.

31.Сопряжение компьютеров с внешними устройствами. – http://www.pcports.ru/articles/avr4.php.

32. Специализированное программное обеспечение для автоматизации деятельности метрологической службы. – https://www.dipaul.ru/upload/iblock/721/dipaul_fluke_metcal_2015_100dpi.pdf.

33.Геращенко, О.А. Температурные измерения. Справочник / О.А. Геращенко, А.Н. Гордов, А.К. Еремина и др. – Киев: Изд-во Наук. думка, 1989. – 704 с.

34.Rev, H. Low Voltage Temperature Sensors. TMP35/TMP36/TMP37 / H. Rev // Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners., 2015. – 19 p.

ПРИЛОЖЕНИЕ А

Листинг А1. – Код графического интерфейса

```
#include "math.h"
#include <vcl.h>
#pragma hdrstop
#include "UTPSV_COMport.h"
#include "Server_p1.h"
#include <fstream.h>
#include "stdio.h"
#include <stdlib.h>
#include "Dialogs.hpp"
#include "Ar1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "UTPSV_COMport"
#pragma resource "*.dfm"
#define FNAME "test.$$$"
TForm1* Form1;
TForm2* Form2;
//Tfrm_View1 *frm_View1;

int bbb = 1;//счетчик для вывода в memo в одной строке всех n=...
bool Slave[16];//подключенные устройства
bool process; //переменная,показывающая , что выполняется запрос
const KolReg = 500;//Количество регистров , которыми можно оперировать
union { int t; float f; char byte[4]; } Digit;
union { float f; char byte[4]; } Digit_p[48000];
```

```
unsigned char KolSlave;//Количество подключенных слэйвов
unsigned char Data[25]; //Массив для отправки данных на ком-порт
unsigned char Dat[25];
unsigned short a;
unsigned char kolerr, par[60];
unsigned long kolrd;
unsigned char rdbuf[15000];//Массив для принятия данных с ком-порта
unsigned short CRC16(unsigned char puchMsg[256], int usDataLen);
int  RKolByte(unsigned short Kol, unsigned char Fun);
int i, j, fl, fl1, parol;
unsigned char pi, lim, ph, pp = 1, k;
unsigned long m, nmax, nmin;
unsigned char buf[2], ft = 0;
unsigned long n1;
float step, d, rbuf[5000], dmax, dmin;
unsigned char n = 30;
union { signed long f; char byte[4]; } inf;
float fmax, fmin, sum, fstep, mo;
unsigned long kolg[20];

AnsiString str;
TPSV_COMport* cp1;
AnsiString FName;

AnsiString Edinicy[] = { "p1= ","p2= ","p3= ","h= ","t1= ","t2= ","t3= ","fun1= ",
"fun2= ","fun3= ","g= ","h= ","v= ","m= ","qv= ","qm= ","qvt= ","qmt= ",
"Nrm1= ","Ndr1= ","Nrt1= ","brm1= ","bdr1= ","brt1= ",
"Nrm2= ","Ndr2= ","Nrt2= ","brm2= ","bdr2= ","brt2= ",
```

```
"Nrm3= ","Ndr3= ","Nrt3= ","brm3= ","bdr3= ","brt3= ",
"Nrtm= ","brtm= ","Nrtmf= ","rm1= ","dr1= ","rt1= ","rm2= ","dr2= ","rt2= ",
"rm3= ","dr3= ","rt3= ","p1nk= ","p2nk= ","p3nk= ",
"rg01_1= ","rgd1_1= ","rg02_1= ","rgd2_1= ","rg03_1= ","rgd3_1= ",
"rg01_2= ","rgd1_2= ","rg02_2= ","rgd2_2= ","rg03_2= ","rgd3_2= ",
"rg01_3= ","rgd1_3= ","rg02_3= ","rgd2_3= ","rg03_3= ","rgd3_3= ",
"a0= ","a1= ","a2= ","a3= ","a4= ","a5= ","a6= ","a7= ","a8= ","a9= ",
"a10= ","a11= ","a12= ","a13= ","a14= ","a15= ","a16= ","a17= ","a18= ","a19= ",
"a20= ","a21= ","a22= ","a23= ","a24= ","a25= ","a26= ","a27= ","a28= ","a29= ",
"a30= ","a31= ","a32= ","a33= ","a34= ","a35= ","a36= ","a37= ","a38= ","a39= ",
"a40= ","a41= ","a42= ","a43= ","a44= ","a45= ","a46= ","a47= ","a48= ","a49= ",
"a50= ","a51= ","a52= ","a53= ","a54= ","a55= ","a56= ","a57= ","a58= ","a59= ",
"a60= ","a61= ","a62= ","a63= ","a64= ","a65= ","a66= ","a67= ","a68= ","a69= ",
"a70= ","a71= ","a72= ","a73= ","a74= ","a75= ","a76= ","a77= ","a78= ","a79= ",
"a80= ","a81= ","a82= ","a83= ","a84= ","a85= ","a86= ","a87= ","a88= ","a89= ",
"a90= ","a91= ","a92= ","a93= ","a94= ","a95= ","a96= ","a97= ","a98= ","a99= ",
"a100= ","a101= ","a102= ","a103= ","a104= ","a105= ","a106= ","a107= ","a108=
","a109= ",
"a110= ","a111= ","a112= ","a113= ","a114= ","a115= ","a116= ","a117= ","a118=
","a119= ",
"a120= ","a121= ","a122= ","a123= ","a124= ","a125= ","a126= ","a127= ","a128=
","a129= ",
"a130= ","a131= ","a132= ","a133= ","slave= ","speed= ","p10= ","p20= ","p30= ",
"kan1= ","kan2= ","limtm= ","limrm= ","limdr= ","usr= ","hk1= ","hk2= ","dph=
","dpk1= ",
"n= ","b0= ","b1= ","b2= ","b3= ","b4= ","b5= ","b6= ","b7= ","b8= ",
"b9= ","b10= ","флэш ","0р ","калибр ","динамика "
};
```

```
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
```

```
    0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01,
    0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00,
0xC1, 0x81,
    0x40
    };
    static unsigned char auchCRCLo[] = {
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05,
0xC5, 0xC4,
    0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B,
0xC9, 0x09,
    0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF,
0x1F, 0xDD,
    0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12,
0x13, 0xD3,
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36,
0xF6, 0xF7,
    0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE,
0xFA, 0x3A,
    0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A,
0xEA, 0xEE,
    0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7,
0xE6, 0x26,
```

```
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63,
0xA3, 0xA2,
    0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D,
0xAF, 0x6F,
    0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9,
0x79, 0xBB,
    0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74,
0x75, 0xB5,
    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50,
0x90, 0x91,
    0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C,
0x5C,
    0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58,
0x98, 0x88,
    0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D,
0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81,
0x80,
    0x40
    };
    //---------------------------------------------------------------------------
    __fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
    {
        cp1->ComPCreate();
        cp1->ComPClearBuff();
        KolSlave = 1;
        Timer2->Enabled = true;
```

```
}
//---------------------------------------------------------------------------
//Процедура принятия данных с порта и проверка пришедших данных
void __fastcall TForm1::WriteData(unsigned char KByte, unsigned char Funk)
{
   char* p;
   AnsiString* p1;
   cp1->ComPClearBuff();  //Очистка буфера ком-порта
   cp1->RTScontrol = Disable_;
   cp1->ComPSetParam();
   cp1->DelayMs(200); //Тайм-аут 0,2 сек
   try
   {
      cp1->ComPRead(rdbuf, KByte);
   }
   catch (...) //Обработка исключения - ответ не пришел => неадрессованных устройств нет
  {
      ShowMessage(" Нет ответа ");
      kolerr++;
      return;
    }
    if (rdbuf[0] != Data[0]) {
      kolerr++;
      ShowMessage("  Slave другой ");
      goto wd1;
    }
    else
```

```
    {
      if (rdbuf[1] == Data[1] + 0x80)
      {
        a = CRC16(rdbuf, 4);
        if (a != 0)
        {
          ShowMessage(" не совпадение контрольной суммы CRC16 приема 5байт ");
          kolerr++; goto wd1;
        }
        else
        {
          ShowMessage(" ТВ: ошибка при приеме ");
          kolerr++; goto wd1;
        }
      }
      if (rdbuf[1] == Data[1])
      {
        a = CRC16(rdbuf, KByte - 1);
        if (a != 0)
        {
          ShowMessage(" не совпадение контрольной суммы CRC16 при приеме");
          kolerr++; goto wd1;
        }
        else
        {
          if (rdbuf[0] >= KolSlave)KolSlave = rdbuf[0] + 1;
```

```
kolerr = 0;
switch (Funk)  //Выбор выполненной функции
{
case 16:  // запись в регистр
   if (rdbuf[0] == Data[0] && rdbuf[1] == Data[1] && rdbuf[2] == Data[2]
&& rdbuf[3] == Data[3] && rdbuf[4] == Data[4] && rdbuf[5] == Data[5])
      ShowMessage("Данные успешно записаны в регистр");
   else  ShowMessage("Ошибка записи в регистр");
   break;
case 3:  //чтение из регистра
   p = &rdbuf[3];
   //Запись пришедших данных в массив Digit_p (float)
   // rdbuf[2] -  количество байт данных в ответе (без 2х первых байт и
без CRC16)
   for (i = 0; i <= (rdbuf[2] / 4); i++)
      for (j = 3; j >= 0; j--)
      {
         Digit_p[i].byte[j] = *p++;
      };
   //Вывод пришедших данных
   Memo1; ->Lines->Add("");
   bbb; ++;
   Memo1;  ->Lines->Add("Транзакция  от  :" + str + " Slave " +
IntToStr(rdbuf[0]));
   bbb; ++;
   //    ShowMessage(Data[3]);
   p1 = &Edinicy[((Data[3] - 1) / 2)];
   for (i = 0; i <= (rdbuf[2] / 4 - 1); i++)
```

```
                {
                    Memo1->Lines->Add((*p1++)                                    +
FloatToStr(Digit_p[i].f));//+Edinicy[n].c_str());
                    bbb++;
                }
                break;
              }
            }
          }
        }
    wd1: i = 0;         //холостая команда
    }

    //---------------------------------------------------------------------------
    //Функция подсчета контрольной суммы
    unsigned short CRC16(unsigned char puchMsg[256], int usDataLen)
    {
      unsigned char uchCRCHi = 0xFF;
      unsigned char uchCRCLo = 0xFF;
      unsigned char uIndex;
      int i = 0;
      while (i <= usDataLen)
      {
        uIndex = uchCRCHi ^ puchMsg[i];
        uchCRCHi = (uchCRCLo ^ auchCRCHi[uIndex]);
        uchCRCLo = auchCRCLo[uIndex];
        i++;
      }
```

```
    return (uchCRCHi << 8 | uchCRCLo);
}

//Отправка запроса - запись данных  --------------------------------
void __fastcall TForm1::Button1Click(TObject* Sender)
{
    unsigned short n;
    fl = 0;
    if (pp == 0)ShowMessage("Введите пароль");
    else
    {
       if (!process)   //Если не обрабатывается запрос
       {
          process = true;
          StrCopy(Data, "");
          try
          {
             //Data[...]-данные,отправляемые в запросе
             // Data[0] - адрес слэйва  8 бит
             Data[0] = (ComboBox4->ItemIndex);
             //Data[1] - номер функции  8 бит
             Data[1] = 16;
             if ((ComboBox1->ItemIndex) > 68)n = (ComboBox1->ItemIndex) * 2;
             else ShowMessage("Нет записи в этот регистр");
             // Data[2&3] - номер  регистра , в который происходит запись 16 бит
             Data[2] = n >> 8;
             Data[3] = n;
```

```
//Data[4&5] -  количестов регистров для записи. Мы здесь записываем данные в два регистра
Data[4] = 0;
Data[5] = 2;
//Data[6] - счетчик байт. Мы записываем 4 байта
Data[6] = 4;
// Data[7&8&9&10] - число , записываемое в 2 регистра 32 бита
Digit.f = StrToFloat(Edit2->Text.c_str());
Data[7] = Digit.byte[3];
Data[8] = Digit.byte[2];
Data[9] = Digit.byte[1];
Data[10] = Digit.byte[0];
// Data[11&12] - контрольная сумма 16 бит
a = CRC16(Data, 10);
Data[11] = a >> 8;
Data[12] = a;
kolerr = 0;
for (k = 0; k < 13; k++) Dat[k] = Data[k];
err7: try
{
cp1->RTScontrol = Enable_;
cp1->ComPSetParam();
cp1->ComPClearBuff();
for (k = 0; k < 13; k++)
{
cp1->ComPWrite(Dat, 1);
Dat[0] = Dat[k + 1];
}
```

```
            //  cp1->ComPWrite(Data,13);  //Отправка запроса на сом-порт
        }
        catch (eComPortExcept & E)//Обработка исключения в случае неудачной записи в сом-порт
        {
            cp1->ComPortExceptShow(E);
            fl = 1;
        };
        if (fl != 1)
        {
            cp1->DelayMs(3);
            WriteData(8, 16);//Вызов процедуры принятия ответа
            if (kolerr == 0)process = false;
            else
            {//if(kolerr<3)goto err7;
             //else
             //ShowMessage ("Попытки записать не удались");
            }
        }
        process = false;
        }
        catch (...) //Обработка исключения в случае неправильного ввода числа
        {
            ShowMessage("Ошибка ввода числа , повторите ввод");
            process = false;
        };
    }
    else
```

```
            ShowMessage("Происходит обработка запроса,попробуйте чуть позже");
        }
    }

    //Функция подсчета количества байт данных , которые должны прийти в ответ с контроллера на запрос сервера
    int  RKolByte(unsigned short Kol, unsigned char Fun)
    {
        int p, k;
        //Kol - количество запрашиваемых регистров или ячеек
        switch (Fun)  //выбор функции
        {
        case 3: p = 5 + Kol * 2; break;
        case 16: p = 8; break;
        }
        return(p); // p - количество байт , которое должно прийти в ответе
    }
    //---------------------------------------------------------------------------


    void __fastcall TForm1::Exit(TObject* Sender, bool& CanClose)
    {
        if (MessageDlg("Прекратить работу?", mtConfirmation, TMsgDlgButtons() << mbOK << mbCancel, 0) == mrCancel)
            CanClose = false;
    }
```

```
void __fastcall TForm1::TabSheet3ContextPopup(TObject* Sender,
   TPoint& MousePos, bool& Handled)
{
   a = 0;
}
//------------------Запись номера slave   -----------------

void __fastcall TForm1::Button3Click(TObject* Sender)
{
   //Data[0]-адрес слэйва 8 бит
   Data[0] = (ComboBox5->ItemIndex);
   //Data[1]-номер функции. 20 - пользовательская функция 8 бит
   Data[1] = 20;
   //Data[2]-отсылаемый новый адрес 8 бит
   Data[2] = StrToInt(Edit1->Text.c_str());
   a = CRC16(Data, 2);
   // Data[3&4] - контрольная сумма   16 бит
   Data[3] = a >> 8;
   Data[4] = a;
   for (k = 0; k < 5; k++) Dat[k] = Data[k];
   try
   {
      cp1->RTScontrol = Enable_;
      cp1->ComPSetParam();
      cp1->ComPClearBuff();
      for (k = 0; k < 5; k++)
```

```
        {
            cp1->ComPWrite(Dat, 1);
            Dat[0] = Dat[k + 1];
        }
    }
    catch (eComPortExcept & E)  //Обработка исключения в случае неудачной записи
    {
        cp1->ComPortExceptShow(E);
        ShowMessage("Ошибка записи");
        return;
    };
    cp1->DelayMs(3);
    cp1->ComPClearBuff();  //Очистка буфера
    cp1->RTScontrol = Disable_;
    cp1->ComPSetParam();
    cp1->DelayMs(200);    //Тайм-айт 0,2 сек
    try
    {
        cp1->ComPRead(rdbuf, 5);  //Чтение данных с сом-порта в массив rdbuf
    }
    catch (...) //Обработка исключения - ответ не пришел => неадрессованных устройств нет
    {
        ShowMessage(" Устройство не обнаружено ");
        return;
    }
```

```
        if (rdbuf[0] == Data[0] && rdbuf[1] == Data[1] && rdbuf[2] == Data[2] &&
rdbuf[3] == Data[3] && rdbuf[4] == Data[4])
        {
           Slave[KolSlave] = true;
           ShowMessage(" Устройству присвоен новый адрес ");
        }
     }

     //---------------------------------------------------------------------------
     //------------------Динамика-----------------------------------
     void __fastcall TForm1::Button4Click(TObject* Sender)
     {
        char* p;
        unsigned long i1 = 0;
        char i2;
        unsigned int n, Nz, kolerr1 = 0;
        process = true;
        StrCopy(Data, "");
        Memo1->Lines->Add("Старт приема данных");
        bbb++;
        //Data[...]-данные,отправляемые в запросе
        // Data[0] - адрес слэйва  8 бит
        Data[0] = (ComboBox3->ItemIndex);
        // Data[1] - номер функции  8 бит
        Data[1] = 3;
        Data[3] = 1;
        Data[2] = 0;
        Data[4] = 0;
```

```
Data[5] = 8;   //кол-во 16-р регистров (p1,p2,p3,h)
// Data[6&7] - контрольная сумма 16 бит
a = CRC16(Data, 5);
Data[6] = a >> 8;
Data[7] = a;
Nz = StrToInt(Edit3->Text.c_str());
for (n = 0; n < Nz; n++)
{

   if (n) { Memo1->Lines->Delete(bbb); bbb--; }
   Memo1->Lines->Add("n=" + IntToStr(n));
   bbb++;

   try
   {
      cp1->RTScontrol = Enable_;
      cp1->ComPSetParam();
      cp1->ComPClearBuff();
      for (k = 0; k < 8; k++) Dat[k] = Data[k];
      for (k = 0; k < 8; k++)
      {
         cp1->ComPWrite(Dat, 1);
         Dat[0] = Dat[k + 1];
      }
   }
   catch (eComPortExcept & E)  //Обработка исключения в случае неудачной записи
   {
```

```
        cp1->ComPortExceptShow(E);
        ShowMessage("Ошибка записи");
        return;
      };
      cp1->DelayMs(2);
      cp1->ComPClearBuff();  //Очистка буфера ком-порта
      cp1->RTScontrol = Disable_;
      cp1->ComPSetParam();
      cp1->DelayMs(100); //Тайм-аут 0,1 сек
      try
      {
        cp1->ComPRead(rdbuf, 21);
      }
      catch (...) //Обработка исключения - ответ не пришел => неадрессованных устройств нет
      {
        kolerr1++;
        goto start;
      };
      //    p=&rdbuf[3];
      i2 = 0;
      for (i = 0; i < 4; i++)
      {
        for (j = 3; j >= 0; j--)
        {
          Digit_p[i1].byte[j] = rdbuf[3 + i2];
          i2++;
        }
```

```
            i1++;
        }
    start: a = 0;
    }
    Memo1->Lines->Add("Прием данных закончен");
    bbb++;
    Memo1->Lines->Add("err=" + IntToStr(kolerr1));
    bbb++;
    process = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject* Sender)
{
    unsigned int Nz, i1;
    unsigned long j1 = 0;
    int b = 0;
    int k;
    Nz = StrToInt(Edit3->Text.c_str());
    Form2->Table1->First();

    while (!Form2->Table1->Eof)
    {
        b++;
        Form2->Table1->Next();
    }
    Form2->Table1->First();
    for (k = 0; k < b; k++)
```

```
    {
      Form2->Table1->Delete();
      Form2->Table1->Next();
    }

    for (i1 = 0; i1 < Nz; i1++)
    {
      Form2->Table1->First();
      Form2->Table1->Insert();
      for (j = 0; j < 4; j++)Form2->Table1->Fields->Fields[j]->Clear();
      Form2->Table1->FieldByName("p1")->AsString = FloatToStr(Digit_p[j1].f);
      j1++;
      Form2->Table1->FieldByName("p2")->AsString = FloatToStr(Digit_p[j1].f);
      j1++;
      Form2->Table1->FieldByName("p3")->AsString = FloatToStr(Digit_p[j1].f);
      j1++;
      Form2->Table1->FieldByName("h")->AsString = FloatToStr(Digit_p[j1].f);
      Form2->Table1->Post();
      j1++;
      //  Form2->Table1->Next();
    }
    Form2->Table1->First();
    Form2->Show();
    Form2->Table1->ApplyUpdates();
    Form2->Table1->CommitUpdates();
  }
  //---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button6Click(TObject* Sender)
{
   unsigned int Nz;
   Nz = StrToInt(Edit3->Text.c_str());
   FILE* F;
   if (SaveF_Dr->Execute())
   {
      FName = SaveF_Dr->FileName;
      if ((F = fopen(FName.c_str(), "wb")) == NULL)
      {
         ShowMessage("Не удается создать файл для архива");
         return;
      }
      fwrite(Digit_p, sizeof(float), Nz * 4, F);
      fclose(F);
      Memo1->Lines->Add("Архив сохранен в файле");
      bbb++;
   }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject* Sender)
{
   unsigned int Nz;
   Nz = StrToInt(Edit3->Text.c_str());
   FILE* F;
   if (OpenF_Dr->Execute())
   {
```

```
        FName = OpenF_Dr->FileName;
        if ((F = fopen(FName.c_str(), "rb")) == NULL)
        {
           ShowMessage("Не удается открыть файл");
           return;
        }
        fread(Digit_p, sizeof(float), Nz * 4, F);
        fclose(F);
        Memo1->Lines->Add("Архив открыт");
        bbb++;
     }

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button8Click(TObject* Sender)
{
   unsigned long kolrd1;
   char ff = 0;

   kolrd1 = 3 * StrToInt(Edit5->Text.c_str());
   cp1->RTScontrol = Enable_;
   cp1->ComPSetParam();
   try
   {
      Data[0] = (ComboBox6->ItemIndex) + 1;
      cp1->ComPClearBuff();
      cp1->ComPWrite(Data, 1);  //Отправка запроса на ком-порт
```

```
    }
    catch (eComPortExcept & E)//Обработка исключения в случае неудачной записи
в ком-порт
    {
      cp1->ComPortExceptShow(E);
    }
    Memo2->Lines->Add("Команда отправлена");
    if ((ComboBox6->ItemIndex) < 7)
    {
      cp1->DelayMs(2);
      cp1->RTScontrol = Disable_;
      cp1->ComPSetParam();
      cp1->ComPClearBuff();  //Очистка буфера ком-порта
      Memo2->Lines->Add("Подождите, идет прием архива");
      try
      {
        for (m = 0; m < kolrd1; m++)
        {
          cp1->ComPRead(buf, 1);
          rdbuf[m] = buf[0];
        }
      }
      catch (...)
      {
        ff = 1;
      }
      if (ff == 0) Memo2->Lines->Add("Архив принят");
      else Memo2->Lines->Add("Архив не принят");
```

```
    }
    cp1->ComPSetParam();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button9Click(TObject* Sender)
{
    unsigned long k1 = 0;
    unsigned long kolr;

    kolr = StrToInt(Edit5->Text.c_str()) - 1;
    for (m = 0; m < kolr; m++)
    {
        for (i = 0; i < 3; i++)
        {
            inf.byte[i] = rdbuf[k1];
            k1++;
        }
        inf.byte[3] = 0;
        rbuf[m] = inf.f * 1.0;
    }
    dmax = rbuf[0] * 1.0;
    for (m = 1; m < kolr; m++)
    {
        if (rbuf[m] > (dmax * 1.0))
        {
            dmax = rbuf[m] * 1.0;
            nmax = m;
```

```
    }
}
dmin = rbuf[0] * 1.0;
for (m = 1; m < kolr; m++)
{
   if (rbuf[m] < (dmin * 1.0))
   {
      dmin = rbuf[m] * 1.0;
      nmin = m;
   }
}
Memo2->Lines->Add("nmax= " + IntToStr(nmax) + " max= " + FloatToStr(dmax));
Memo2->Lines->Add("nmin= " + IntToStr(nmin) + " min= " + FloatToStr(dmin));
sum = 0;
for (m = 0; m < kolr; m++)
{
   sum = sum + rbuf[m];
}
mo = sum / kolr;
sum = 0;
for (m = 0; m < kolr; m++)
{
   sum = sum + (rbuf[m] - mo) * (rbuf[m] - mo);
}
d = sum / (kolr - 1);

Memo2->Lines->Add("d= " + FloatToStr(d));
Memo2->Lines->Add("mo= " + FloatToStr(mo));
```

```
        for (i = 0; i < n; i++) kolg[i] = 0;
        step = (dmax - dmin) / n;
        Memo2->Lines->Add("step= " + FloatToStr(step));
        for (i = 0; i < n; i++)
        {
           for (m = 0; m < kolr; m++)
           {
              if (rbuf[m] >= (dmin + step * i))
              {
                 if (rbuf[m] < (dmin + step * (i + 1)))
                    kolg[i]++;
              }
           }
           Memo2->Lines->Add(FloatToStr(dmin + step * i + step / 2) + "        " +
IntToStr(kolg[i]));
        }
     }
     //---------------------------------------------------------------------------
```

Листинг А2. – Вкладка «Чтение»

```
     //Отправка запроса - чтение данных-----------------------------
     void __fastcall TForm1::Button2Click(TObject* Sender)
     {
        unsigned short n;
        fl = 0;
        if (!process)  //Если не обрабатывается запрос
        {
           process = true;
           StrCopy(Data, "");
```

```
try
{
   //Data[...]-данные,отправляемые в запросе
   // Data[0] - адрес слэйва  8 бит
   Data[0] = (ComboBox3->ItemIndex);
   // Data[1] - номер функции  8 бит
   Data[1] = 3;
   Data[3] = (ComboBox2->ItemIndex) * 2 + 1;
   Data[2] = 0;
   Data[4] = 0;
   try
   {
      Data[5] = StrToInt(Edit4->Text.c_str());
   }
   catch (...)
   {
      ShowMessage("Количество значений введено неправильно");
      fl = 1;
   };
   // Data[6&7] - контрольная сумма 16 бит
   a = CRC16(Data, 5);
   Data[6] = a >> 8;
   Data[7] = a;
err3:
   for (k = 0; k < 8; k++) Dat[k] = Data[k];
   if (Data[5] > KolReg)
      ShowMessage("Количество значений введено неправильно");
   else
```

```
{
   kolerr = 0;
   try
   {
      cp1->RTScontrol = Enable_;
      cp1->ComPSetParam();
      cp1->ComPClearBuff();
      for (k = 0; k < 8; k++)
      {
         cp1->ComPWrite(Dat, 1);
         Dat[0] = Dat[k + 1];
      }
      //  cp1->ComPWrite(Data,8);  //Отправка запроса на сом-порт (8 байт)
   }
   catch (eComPortExcept & E)//Обработка исключения в случае неудачной записи в сом-порт
   {
      cp1->ComPortExceptShow(E);
      fl = 1;
   };
   if (fl != 1) //Если не было ошибки записи на сом-порт
   {
      cp1->DelayMs(3);
      WriteData(RKolByte(Data[5], 3), 3);//Вызов процедуры принятия ответа
      if (kolerr == 0)process = false;
      else
      {//if(kolerr<10)goto err3;
```

```
                //else
                // ShowMessage ("Попытки прочитать не удались");
                }
            };
        };
        process = false;
    }
    catch (...) //Обработка исключения в случае неправильного ввода числа
    {
        ShowMessage("Ошибка ввода числа , повторите ввод");
        process = false;
    };
  }
  else
    ShowMessage("Происходит обработка запроса,попробуйте чуть позже");

}
//---------------------------------------------------------------------------
```

Листинг А.2 – Код подпрограммы

```
#include <vcl.h>
#pragma hdrstop

#include "UTPSV_COMport.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
#define TIMEOUT 2000        // времен'ной интервал по умолчанию, мс
```

```
#define VTIMEOUT 500
#define APPID_STR_LONG 25   // максимальное число символов в строке имени приложения, использующего COM-порт
#define NUMBYTES 1          // количество байт по умолчанию
#define STRVERSION "Версия компонента 1.11"
#define VERSION 11;

static inline void ValidCtrCheck(TPSV_COMport *)
{
        new TPSV_COMport(NULL);
}
//---------------------------------------------------------------------------
__fastcall TPSV_COMport::TPSV_COMport(TComponent* Owner)
        : TComponent(Owner)              // Конструктор
{
  lpWriteBuff = NULL;
  lpReadBuff = NULL;
  FpName = "COM2";      // Значения полей по умолчанию
  FBaudRate = 2400;
  FByteSize = 8;
  FParity = 0;
  FStopBits = 2;
  FParityEnabled = FALSE;
  FDTRcontrol = DTR_CONTROL_ENABLE;
  FRTScontrol = RTS_CONTROL_ENABLE;
  FReadIntervalTimeout = 0;
  FReadTotalTimeoutMultiplier = TIMEOUT;
  FReadTotalTimeoutConstant = 0;
```

```
  FWriteTotalTimeoutMultiplier = 0;
  FWriteTotalTimeoutConstant = 0;
  FNumBytesToWrite = NUMBYTES;
  FNumBytesToRead = NUMBYTES;
  FNumBytesWritten = 0;
  FNumBytesRead = 0;
}
//---------------------------------------------------------------------------
__fastcall TPSV_COMport::~TPSV_COMport()      // Деструктор
{
  if(hCom)
    CloseHandle(hCom);    //Закрывает открытый дескриптор ресурсов порта
}
//---------------------------------------------------------------------------
namespace Utpsv_comport
{
        void __fastcall PACKAGE Register()
        {
                TComponentClass classes[1] = {__classid(TPSV_COMport)};
                RegisterComponents("METRAN_C", classes, 0);
        }
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetPortName(ePortName eValue)
{                       // метод записи в поле FpName
  switch (eValue)
   {
```

```
    case COM1 : FpName = "COM1"; break;
    case COM2 : FpName = "COM2"; break;
    case COM3 : FpName = "COM3"; break;
    case COM4 : FpName = "COM4"; break;
    default : { throw(errPSetPortName); }// вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
ePortName __fastcall TPSV_COMport::GetPortName()
{                    // метод чтения из поля FpName
  ePortName eValue;
  bool sc=False;
  if(FpName=="COM1") {eValue = COM1; sc=True;}
  if(FpName=="COM2") {eValue = COM2; sc=True;}
  if(FpName=="COM3") {eValue = COM3; sc=True;}
  if(FpName=="COM4") {eValue = COM4; sc=True;}
  if(sc==False)
    throw(errPGetPortName); // вариант обработки исключения

  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetBaudRate(eBaudRate eValue)
{                    // установка скорости порта
  switch (eValue)
  {
    case _110    : FBaudRate = 110;    break;
```

```
    case _150    : FBaudRate = 150;    break;
    case _300    : FBaudRate = 300;    break;
    case _600    : FBaudRate = 600;    break;
    case _1200   : FBaudRate = 1200;   break;
    case _2400   : FBaudRate = 2400;   break;
    case _4800   : FBaudRate = 4800;   break;
    case _9600   : FBaudRate = 9600;   break;
    case _19200  : FBaudRate = 19200;  break;
    case _38400  : FBaudRate = 38400;  break;
    case _57600  : FBaudRate = 57600;  break;
    case _115200 : FBaudRate = 115200; break;
    default : { throw(errPSetBaudRate); }// вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
eBaudRate __fastcall TPSV_COMport::GetBaudRate()
{                    // чтение текущей скорости порта
  eBaudRate eValue;
  switch (FBaudRate)
  {
    case 110    : eValue = _110;    break;
    case 150    : eValue = _150;    break;
    case 300    : eValue = _300;    break;
    case 600    : eValue = _600;    break;
    case 1200   : eValue = _1200;   break;
    case 2400   : eValue = _2400;   break;
    case 4800   : eValue = _4800;   break;
    case 9600   : eValue = _9600;   break;
```

```
    case 19200  : eValue = _19200;  break;
    case 38400  : eValue = _38400;  break;
    case 57600  : eValue = _57600;  break;
    case 115200 : eValue = _115200; break;
    default : { throw(errPGetBaudRate); }// вариант обработки исключения
   }
  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetByteSize(eByteSize eValue)
{                    // Установка числа бит передаваемых данных в байте
  switch (eValue)
  {
    case _5 : FByteSize = 5;    break;
    case _6 : FByteSize = 6;    break;
    case _7 : FByteSize = 7;    break;
    case _8 : FByteSize = 8;    break;
    default : { throw(errPSetByteSize); }// вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
eByteSize __fastcall TPSV_COMport::GetByteSize()
{                    // Чтение числа бит данных в передаваемом байте
  eByteSize eValue;
  switch(FByteSize)
  {
    case 5 : eValue = _5; break;
```

```
    case 6 : eValue = _6; break;
    case 7 : eValue = _7; break;
    case 8 : eValue = _8; break;
    default : { throw(errPGetByteSize); }// вариант обработки исключения
  }
  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetParity(eParity eValue)
{                    // Установка режима проверки на четность
  switch(eValue)
  {
    case _no    : FParity = 0; break;
    case _odd   : FParity = 1; break;
    case _even  : FParity = 2; break;
    case _mark  : FParity = 3; break;
    case _space : FParity = 4; break;
    default : { throw(errPSetParity); }// вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
eParity __fastcall TPSV_COMport::GetParity()
{                    // Чтение режима проверки на четность
  eParity eValue;
  switch(FParity)
  {
    case 0 : eValue = _no;    break;
```

```
    case 1 : eValue = _odd;   break;
    case 2 : eValue = _even;  break;
    case 3 : eValue = _mark;  break;
    case 4 : eValue = _space; break;
    default : { throw(errPGetParity); }// вариант обработки исключения
   }
  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetStopBits(eStopBits eValue)
{                    // Установка числа стоп-битов
  switch(eValue)
  {
    case _1   : FStopBits = 0; break;
    case _1p5 : FStopBits = 1; break;
    case _2   : FStopBits = 2; break;
    default : { throw(errPSetStopBits); }// вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
eStopBits __fastcall TPSV_COMport::GetStopBits()
{                    // Чтение числа стоп-битов
  eStopBits eValue;
  switch(FStopBits)
  {
    case 0 : eValue = _1;   break;
    case 1 : eValue = _1p5; break;
```

```
    case 2 : eValue = _2;   break;
    default : { throw(errPGetStopBits); } // вариант обработки исключения
   }
   return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetParityEnabled(bool value)
{                   // Установка разрешения на проверку четности
  FParityEnabled=value;
}
//---------------------------------------------------------------------------
bool __fastcall TPSV_COMport::GetParityEnabled()
{                   // Чтение разрешения проверки на четность
  bool value;
  value=FParityEnabled;
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetDTRcontrol(eDTRcontrol eValue)
{                   // чтение типа DTR контроля
  switch(eValue)
  {
    case Disable   : FDTRcontrol = DTR_CONTROL_DISABLE;   break;
    case Enable    : FDTRcontrol = DTR_CONTROL_ENABLE;    break;
    case Handshake : FDTRcontrol = DTR_CONTROL_HANDSHAKE; break;
    default : { throw(errPSetDTRcontrol); }// вариант обработки исключения
```

```
  }
}
//---------------------------------------------------------------------------
eDTRcontrol __fastcall TPSV_COMport::GetDTRcontrol()
{                    // установка типа DTR контроля
  eDTRcontrol eValue;
  switch(FDTRcontrol)
  {
    case DTR_CONTROL_DISABLE   : eValue = Disable;   break;
    case DTR_CONTROL_ENABLE    : eValue = Enable;    break;
    case DTR_CONTROL_HANDSHAKE : eValue = Handshake; break;
    default : { throw(errPGetDTRcontrol); } // вариант обработки исключения
  }
  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetRTScontrol(eRTScontrol eValue)
{                    // чтение типа RTS контроля
  switch(eValue)
  {
    case Disable_   : FRTScontrol = RTS_CONTROL_DISABLE;   break;
    case Enable_    : FRTScontrol = RTS_CONTROL_ENABLE;    break;
    case Handshake_ : FRTScontrol = RTS_CONTROL_HANDSHAKE; break;
    case Toggle     : FRTScontrol = RTS_CONTROL_TOGGLE;    break;
    default : { throw(errPSetRTScontrol); }// вариант обработки исключения
  }
}
```

```
//---------------------------------------------------------------------------
eRTScontrol __fastcall TPSV_COMport::GetRTScontrol()
{                   // установка типа RTS контроля
  eRTScontrol eValue;
  switch(FRTScontrol)
  {
    case RTS_CONTROL_DISABLE   : eValue = Disable_;   break;
    case RTS_CONTROL_ENABLE    : eValue = Enable_;    break;
    case RTS_CONTROL_HANDSHAKE : eValue = Handshake_; break;
    case RTS_CONTROL_TOGGLE    : eValue = Toggle;     break;
    default : { throw(errPGetRTScontrol); }// вариант обработки исключения
  }
  return eValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetReadIntervalTimeout(int value)
{                   // Установка Max временной задержки между двумя принимаемыми
байтами
  if(value>=0)
    FReadIntervalTimeout = value;
  else
   { throw(errPSetRIntervalTimeout); }   // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetReadIntervalTimeout()
{                   // Чтение Max временной задержки между двумя принимаемыми
байтами
```

```
  int value;
  if(FReadIntervalTimeout < MaxInt)
    value = FReadIntervalTimeout;
  else
    { throw(errPGetRIntervalTimeout); }   // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetReadTotalTOMultiplier(int value)
{                    // Установка общей временной задержки для чтения 1 байта
  if(value>=0)
    FReadTotalTimeoutMultiplier = value;
  else
    { throw(errPSetRTotalTOMultiplier); } // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetReadTotalTOMultiplier()
{                    // Чтение общей временной задержки для чтения 1 байта
  int value;
  if(FReadTotalTimeoutMultiplier < MaxInt)
    value = FReadTotalTimeoutMultiplier;
  else
    { throw(errPGetRTotalTOMultiplier); } // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TPSV_COMport::SetReadTotalTOConstant(int value)
{                    // Установка общей временной задержки для чтения 1 байта
  if(value>=0)
    FReadTotalTimeoutConstant = value;
  else
    { throw(errPSetRTotalTOConstant); } // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetReadTotalTOConstant()
{                    // Чтение общей временной задержки для чтения 1 байта
  int value;
  if(FReadTotalTimeoutConstant < MaxInt)
    value = FReadTotalTimeoutConstant;
  else
    { throw(errPGetRTotalTOConstant); } // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetWriteTotalTOMultiplier(int value)
{                    // Установка общей временной задержки для записи 1 байта
  if(value>=0)
    FWriteTotalTimeoutMultiplier = value;
  else
    { throw(errPSetWTotalTOMultiplier); } // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetWriteTotalTOMultiplier()
```

```
{                    // Чтение общей временной задержки для записи 1 байта
  int value;
  if(FWriteTotalTimeoutMultiplier < MaxInt)
    value = FWriteTotalTimeoutMultiplier;
  else
    { throw(errPGetWTotalTOMultiplier); } // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetWriteTotalTOConstant(int value)
{                    // Установка общей временной задержки для записи 1 байта
  if(value>=0)
    FWriteTotalTimeoutConstant = value;
  else
    { throw(errPSetWTotalTOConstant); } // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetWriteTotalTOConstant()
{                    // Чтение общей временной задержки для записи 1 байта
  int value;
  if(FWriteTotalTimeoutConstant < MaxInt)
    value = FWriteTotalTimeoutConstant;
  else
    { throw(errPGetWTotalTOConstant); } // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------
```

```
void __fastcall TPSV_COMport::SetNumBytesToWrite(int value)
{                    // Кол-во байтов для вывода
  if(value>0)
    FNumBytesToWrite = value;
  else
    { throw (errPSetNumBytesToWrite); }  // вариант обработки исключения
}
//---------------------------------------------------------------------------
int __fastcall TPSV_COMport::GetNumBytesToWrite()
{
  int value;
  if(FNumBytesToWrite < MaxInt)
    value = FNumBytesToWrite;
  else
    { throw (errPGetNumBytesToWrite); }  // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetNumBytesToRead(int value)
{                    // Кол-во байтов для ввода
  if(value>0)
    FNumBytesToRead = value;
  else
    { throw (errPSetNumBytesToRead); }  // вариант обработки исключения
}
//---------------------------------------------------------------------------
```

```
int __fastcall TPSV_COMport::GetNumBytesToRead()
{
  int value;
  if(FNumBytesToRead < MaxInt)
    value = FNumBytesToRead;
  else
    { throw (errPGetNumBytesToRead); }  // вариант обработки исключения
  return value;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetAppIdString(AnsiString sValue)
{
  if(sValue.Length() < APPID_STR_LONG)
    FAppIdString = sValue;
  else
    throw (errPSetAppIdStringLong);
}
//---------------------------------------------------------------------------
AnsiString __fastcall TPSV_COMport::GetAppIdString()
{
  AnsiString sValue;
  sValue = FAppIdString;
  return sValue;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::SetExeptionProcessing(bool value)
```

```
{                    // Разрешение обработки исключений внутри компонента
  FEnExeption = value;
}
//---------------------------------------------------------------------------

void __fastcall DelayMs(unsigned long delayms)
{
  DWORD tbegin,tend;
  tbegin = GetTickCount();
  do
    tend = GetTickCount();
  while((tend - tbegin) < delayms);
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::DelayMs(unsigned long delayms)
{
  DWORD tbegin,tend;
  tbegin = GetTickCount();
  do
    tend = GetTickCount();
  while((tend - tbegin) < delayms);
}
//---------------------------------------------------------------------------
void __fastcall TPSV_COMport::ComPCreate()
{                    // Создание связанного с портом потока данных
  lpWriteBuff = NULL;
  lpReadBuff = NULL;
```

```
  hCom = CreateFile(FpName.c_str(),
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL);
  if (hCom == INVALID_HANDLE_VALUE)
    throw(errComPCreateFile);

  ComPSetParam();
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPSetParam()
{                       // Установка параметров порта (полей DCB)
  if (!ComPIsInit())
    throw(errComPNoHandle);

  // Проверка текущих настроек DCB
  if (!GetCommState(hCom, &dcb))
    throw(errGetCommState);

  // Загрузка в DCB новых параметров
  dcb.BaudRate = FBaudRate;
  dcb.ByteSize = FByteSize;
  dcb.Parity = FParity;
  dcb.StopBits = FStopBits;
```

```
dcb.fParity = FParityEnabled;
dcb.fDtrControl = FDTRcontrol;
dcb.fRtsControl = FRTScontrol;
// Неизменяемые параметры
dcb.fBinary = TRUE;
dcb.fOutX = FALSE;
dcb.fInX = FALSE;
dcb.fOutxCtsFlow = FALSE;
dcb.fOutxDsrFlow = FALSE;

dcb.fDsrSensitivity = FALSE;
dcb.fNull = FALSE;
dcb.fErrorChar = FALSE;
dcb.fAbortOnError = FALSE;

// Установка новых параметров DCB
if (!SetCommState(hCom, &dcb))
  throw(errSetCommState);

// Загрузка новых временн'ых задержек
timeouts.ReadIntervalTimeout = FReadIntervalTimeout;
timeouts.ReadTotalTimeoutMultiplier = FReadTotalTimeoutMultiplier;
timeouts.ReadTotalTimeoutConstant = FReadTotalTimeoutConstant;
timeouts.WriteTotalTimeoutMultiplier = FWriteTotalTimeoutMultiplier;
timeouts.WriteTotalTimeoutConstant = FWriteTotalTimeoutConstant;

// Установка новых временн'ых задержек
if (!SetCommTimeouts(hCom, &timeouts))
```

```
    throw(errSetCommTimeouts);

}
//---------------------------------------------------------------------------

bool __fastcall TPSV_COMport::ComPIsInit(void)
{                   // Проверка создания связанного с портом потока
  if (hCom == INVALID_HANDLE_VALUE)
    return false;
  else
    return true;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPClose(void)
{                   //Закрывает открытый дескриптор ресурсов порта
  try
  {
  if(hCom)
    CloseHandle(hCom);
  }
  catch(...)
  {
    throw(errComPClose);
  }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TPSV_COMport::ComPClearBuff(void)
{                    // Очистка буфера порта
  if(ComPIsInit())
    PurgeComm(hCom, PURGE_TXCLEAR | PURGE_RXCLEAR);
}
//---------------------------------------------------------------------------

int __fastcall TPSV_COMport::ComPVersion(void)
{                    // Возвращает целое число - порядковый номер версии компонента
  return VERSION;
}
//---------------------------------------------------------------------------

AnsiString __fastcall TPSV_COMport::ComPVersionStr(void)
{                    // Возвращает строку с номером версии компонента
  return STRVERSION;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPWrite_(unsigned char wrBuf)
{                    // Запись данных в порт
  lpWriteBuff = wrBuf;
  if (!WriteFile(hCom, lpWriteBuff, FNumBytesToWrite, &FNumBytesWritten, NULL))
  {
    throw(errMComPWrite);   // вариант обработки исключения
  }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TPSV_COMport::ComPWrite(char *wrBuf, DWORD wrNumBytes)
{                    // Запись данных в порт
  lpWriteBuff = wrBuf;
  if (!WriteFile(hCom, lpWriteBuff, wrNumBytes, &FNumBytesWritten, NULL))
  {
    throw(errMComPWrite);   // вариант обработки исключения
  }
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPWriteV_echo(char *wrBuf, DWORD
wrNumBytes)
{                    // Запись данных в порт с ожиданием эхо-ответа [Вольтметр]
  DWORD j;
  char *buff = NULL;
  bool VoltmRestart = false, VRDelay = false;
  ComPClearBuff();
  buff = wrBuf;
  char bufferOut, bufferIn;
  for (DWORD i=0; i<wrNumBytes; i++)
  {
    bufferOut = buff[i];
    // организуем паузу 6 сек для перезапуска вольтметра

    if (VoltmRestart) VRDelay = true;
    if (bufferOut == 'J') VoltmRestart =true;
    // организовываем паузу 0.05 сек
```

```
    DelayMs(50);

    ComPWrite(&bufferOut, 1);
    ComPRead(&bufferIn, 1);        // Ожидание эхо-ответа
    j=0;
    while (bufferOut != bufferIn)
    {
     j++;
     if (j>100)
     {
       throw(errMComPV_echo);     // Не пришел эхо-ответ
     }
     ComPRead(&bufferIn, 1);
    }
    if (VRDelay)
    {
     DelayMs(6000);
     VoltmRestart = VRDelay = false;
    }
  }
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPRead_(char *rdBuf)
{                    // Чтение полученных данных
  lpReadBuff = rdBuf;
  if (!ReadFile(hCom, lpReadBuff, FNumBytesToRead, &FNumBytesRead, NULL))
  {
```

```
    throw(errMComPRead);   // вариант обработки исключения
  }
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPRead(char *rdBuf, DWORD rdNumBytes)
{                    // Чтение заданного числа полученных данных
  lpReadBuff = rdBuf;
  if (!ReadFile(hCom, lpReadBuff, rdNumBytes, &FNumBytesRead, NULL))
  {
    throw(errMComPRead);   // вариант обработки исключения
  }
  if(rdNumBytes != FNumBytesRead)
    throw(errMComPReadNum);
}
//---------------------------------------------------------------------------

AnsiString __fastcall TPSV_COMport::ComPReadPacket(DWORD rdNumBytes)
{              // Побайтное чтение указанного числа символов
  try
  {
    AnsiString bufStr;
    //char readbuf[2] = "\0\0";
    char readbuf;
    for(DWORD i=0;i<rdNumBytes;i++)
    {
      if(i > rdNumBytes)
      {
```

```
      break;
     }
     ComPRead(&readbuf,1);
     bufStr += readbuf;
    }
    //bufStr.c_str();

    return bufStr;
  }
  catch(eComPortExcept& E)
  {
    switch(E)
    {
     case errMComPRead    :  throw(errMComPRead);
     case errMComPReadNum :  throw(errMComPReadNum);
    }
  }
  catch(...)
  {
    throw(errMComPReadPacket);
  }
}
//---------------------------------------------------------------------------

bool __fastcall TPSV_COMport::IntegerExcept(const& IntValue, const& eEx)
{
bool checkV=true;             // not working
```

```
  // Checking integer value to correctness

  if(checkV==false)
    throw(eEx);
  return checkV;
}
//---------------------------------------------------------------------------

void __fastcall TPSV_COMport::ComPortExceptShow(eComPortExcept& E)
{                    // формирует окно с именем ошибки
AnsiString S="Ошибка ";
AnsiString AppIdStr = GetAppIdString();
  AppIdStr += " .Работа с COM портом";
  switch(E)
  {
    case errComPCreateFile   : S += "создания потока";              break;
    case errGetCommState     : S += "чтения установок DCB";         break;
    case errSetCommState     : S += "установки параметров DCB";     break;
    case errSetCommTimeouts  : S += "установки временных задержек"; break;
    case errComPClose        : S += "закрытия дескриптора порта";   break;
    case errComPNoHandle     : S =  "Отсутствует дескриптор порта"; break;
    case errMComPWrite       : S += "записи данных в порт";         break;
    case errMComPWriteNum    : S += "указанного числа символов для записи"; break;
    case errMComPV_echo      : S += "ожидания эхо сигнала [вольтметр]"; break;
    case errMComPRead        : S += "чтения данных с порта";        break;
    case errMComPReadNum     : S += "чтения указанного числа символов"; break;
    case errMComPReadPacket  : S += "циклического побайтного чтения"; break;
    // расшифровка исключений для функций чтения и записи полей
```

```
    case errPSetPortName     : S += "установки имени COM порта";      break;
    case errPGetPortName     : S += "чтения имени COM порта";         break;
    case errPSetBaudRate     : S += "установки скорости обмена данными"; break;
    case errPGetBaudRate     : S += "чтения скорости обмена данными"; break;
    case errPSetByteSize     : S += "установки размера байта (бит)";  break;
    case errPGetByteSize     : S += "чтения размера байта (бит)";     break;
    case errPSetParity       : S += "установки типа проверки на четность"; break;
    case errPGetParity       : S += "чтения типа проверки на четность"; break;
    case errPSetStopBits     : S += "установки числа стоп-битов";     break;
    case errPGetStopBits     : S += "чтения числа стоп-битов";        break;
    case errPSetDTRcontrol   : S += "установки типа DTR-контроля";    break;
    case errPGetDTRcontrol   : S += "чтения типа DTR-контроля";       break;
    case errPSetRTScontrol   : S += "установки типа RTS-контроля";    break;
    case errPGetRTScontrol   : S += "чтения типа RTS-контроля";       break;
    case errPSetRIntervalTimeout   : S += "установки интервальной задержки на
чтение данных"; break;
    case errPGetRIntervalTimeout   : S += "чтения интервальной задержки на чтение
данных";  break;
    case errPSetRTotalTOMultiplier : S += "установки общей задержки на чтение 1
байта данных";  break;
    case errPGetRTotalTOMultiplier : S += "чтения общей задержки на чтение 1 байта
данных"; break;
    case errPSetRTotalTOConstant   : S += "установки добавочной задержки на чтение
данных"; break;
    case errPGetRTotalTOConstant   : S += "чтения добавочной задержки на чтение
данных";    break;
    case errPSetWTotalTOMultiplier : S += "установки общей задержки на запись 1
байта данных"; break;
```

```
   case errPGetWTotalTOMultiplier : S += "чтения общей задержки на запись 1 байта данных"; break;
   case errPSetWTotalTOConstant   : S += "установки добавочной задержки на запись данных"; break;
   case errPGetWTotalTOConstant   : S += "чтения добавочной задержки на запись данных"; break;
   case errPSetNumBytesToWrite    : S += "установки числа байт для записи данных"; break;
   case errPGetNumBytesToWrite    : S += "чтения числа байт для записи данных"; break;
   case errPSetNumBytesToRead     : S += "установки числа байт для чтения данных";   break;
   case errPGetNumBytesToRead     : S += "чтения числа байт для чтения данных"; break;
   case errPSetAppIdStringLong    : S  = "Превышено число символов в строке с именем приложения"; break;

   default : {}
  }
  Application->MessageBox(S.c_str(),AppIdStr.c_str(),MB_OK);
}
```