МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	Кафедра
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС ИЗМЕРЕНИЯ И КОНТРОЛЯ МИКРОКЛИМАТА

Студент 24.05.2026 (подпись, дата) Куракин Никита Николаевич (ФИО полностью)

Руководитель 24.05.2026 (подпись, дата) Федюнин Р.Н. (фамилия, инициалы)

Нормоконтролёр 25.05.2026 (подпись, дата) Бычков А.С. (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой _____ (подпись) Митрохин М.А. (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 02 июня 2026 г. № 2)

Секретарь ГЭК _____ (подпись) Гурин Е.И. (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«07» апреля 2026 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Куракин Никита Николаевич гр. 22ВВП2 факультета ФВТ

2. Тема работы Программно-аппаратный комплекс измерения и контроля микроклимата

Тема утверждена приказом ПГУ № 347/0 от "07" апреля 2026 г.

3. Руководитель бакалаврской работы Федюнин Р. Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.) Разработать программно-аппаратный комплекс для измерения и контроля параметров микроклимата (температура и относительная влажность воздуха, концентрация углекислого газа в воздухе). Программная часть комплекса должна быть спроектирована с применением клиент-серверной архитектуры. Клиентская часть программы должна обладать графическим интерфейсом. Серверная часть программы должна работать в режиме службы операционной системы Windows. Аппаратная часть должна включать в себя датчики для измерения параметров микроклимата и открытую аппаратную платформу для управления датчиками. Комплекс должен осуществлять измерение и анализ параметров микроклимата, установленных ГОСТ 30494-2011 «Межгосударственный стандарт. Здания жилые и общественные. Параметры микроклимата в помещениях». Контроль параметров микроклимата должен выполняться на клиенте с помощью текстовой и цветовой индикации.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметных областей: изучить существующие аналоги комплексов измерения и контроля микроклимата, выделить их достоинства и недостатки; изучить и выбрать датчики для измерения параметров микроклимата; сравнить технологии открытых аппаратных платформ и выбрать инструменты для разработки программного и аппаратного обеспечения комплекса. Необходимо изучить и настроить выбранные инструменты разработки и языки программирования, используемые при разработке комплекса. Также требуется проанализировать протоколы связи между компонентами аппаратной части и взаимодействия между элементами комплекса.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	5%	08.04.26	
2	Анализ требований	5%	13.04.26	
3	Выбор технологий, применяемых при проектировании и разработке	10%	16.04.26	
4	Описание архитектуры системы	10%	24.04.26	
5	Проектирование и реализация комплекса	45%	05.05.26	
6	Демонстрация работы и тестирование комплекса	15%	15.05.26	
7	Оформление выпускной работы	10%	20.05.26	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы _____ 07 апреля 2026 г.
(подпись, дата)

Задание к исполнению принял студент _____ 07 апреля 2026 г.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Л.Ф. Фесенова
27.05.2026 *(подпись, дата)*

Реферат

Бакалаврская работа содержит 168 листов, 96 рисунков, 5 таблиц, 14 источников и 5 приложений.

ПРОГРАММНО-АППАРАТНЫЙ КОМПЛЕКС, МИКРОКЛИМАТ, ДИАПАЗОН ИЗМЕРЕНИЯ, АППАРАТНАЯ ЧАСТЬ, ПРОГРАММНАЯ ЧАСТЬ, ИНТЕРФЕЙС СВЯЗИ, ДАТЧИКИ, ЭЛЕКТРОННЫЙ КОМПОНЕНТ УПРАВЛЕНИЯ

Объектом исследования являются средства измерения параметров микроклимата в закрытых помещениях.

Цель работы – разработка программно-аппаратного комплекса измерения и контроля микроклимата.

В процессе работы реализован анализ предметной области, нормативных требований и существующих аналогичных решений. Проведена разработка программно-аппаратного комплекса в соответствие с ТЗ.

В результате выполненной работы выполнены следующие задачи:

– разработка клиент-серверное приложение для взаимодействия пользователя с комплексом;

– проектирование электронного компонента управления для связи программной и аппаратной частей комплекса;

– выбор датчиков для измерения необходимых параметров микроклимата – температуры, относительной влажности воздуха и концентрации углекислого газа в воздухе;

– разработка аппаратной части комплекса измерения и контроля микроклимата.

Содержание

Перечень сокращений и обозначений 5
Введение 6
1 Анализ предметной области 7
1.1 Постановка задачи 9
1.2 Анализ аналогов 12
1.3 Выбор технологий разработки 14
1.3.1 Выбор датчиков 14
1.3.2 Выбор открытой аппаратной платформы 16
1.3.3 Выбор инструментов разработки аппаратной платформы 18
1.3.4 Установка и настройка Xilinx ISE Design Suite 20
1.3.5 Выбор инструментов разработки программной части 28
1.3.6 Установка и настройка Microsoft Visual Studio 29
2 Архитектура системы 33
2.1 Архитектура аппаратной части 34
2.2 Архитектура программной части 35
3 Описание разработки модулей 36
3.1 Разработка аппаратных модулей 37
3.2 Разработка серверных модулей 46
3.3 Разработка клиентских модулей 57
4 Демонстрация работы проекта 67
4.1 Описание работы 67
4.2 Тестирование системы 74
5 Безопасность жизнедеятельности 83
5.1 Правила безопасности при работе с ПК 83
5.2 Правила безопасности при работе с аппаратной частью 84
Заключение 86
Список использованных источников 87

Приложение А – UML-диаграммы 89
Приложение Б – Электрическая схема 94
Приложение В – Листинг программ 96
Приложение В – Презентация 162

Перечень сокращений и обозначений

В настоящей ВКР применяют следующие сокращения и обозначения

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

ПЛИС – программируемая логическая интегральная схема

ТЗ – техническое задание

CLB (Configurable Logic Block) – конфигурируемый логический блок

I2C (Inter-Integrated Circuit) – последовательный ассиметричный интерфейс связи

ISE (Integrated Synthesis Environment) – интегрированная система автоматизированного проектирования

PWM (pulse-width modulation) – широтно-импульсная модуляция

RH (Relative Humidity) – относительная влажность

RX (Receive) – линия приёма данных

SCL (Serial Clock) – линия последовательного тактового сигнала в интерфейса I2C

SDA (Serial Data) – линия последовательного сигнала данных в интерфейсе I2C

SMD-резистор (Surface Mount Device) – резистор для поверхностного монтажа

SPI (Serial Peripheral Interface) – последовательный периферийный интерфейс

TX (Transmit) – линия отправки данных

UART (Universal Asynchronous Receiver-Transmitter) – последовательный асинхронный приёмопередатчик

USB (Universal Serial Bus) – универсальная последовательная шина

VHDL (Very high-speed integrated circuits Hardware Description Language) – язык описания аппаратуры интегральных схем

Wi-Fi (Wireless Fidelity) – беспроводная связь

WLAN (Wireless Local Area Network) – беспроводная локальная сеть

Введение

Человек проводит значительную часть своего времени в закрытых помещениях – учится, работает, занимается физической или умственной активностью. Эффективность такой деятельности напрямую зависит от здоровья, работоспособности и общего самочувствия человека. Одним из ключевых факторов, определяющих состояние человека в помещении, является качество окружающего воздуха и микроклимата в целом. Отклонение одного из параметров микроклимата от нормы может привести к чувству дискомфорта, начиная от головной боли и заканчивая респираторными заболеваниями.

Нормы и требования к параметрам микроклимата в закрытых помещениях различной категории регламентируется ГОСТ 30494-2011 «Межгосударственный стандарт. Здания жилые и общественные. Параметры микроклимата в помещениях» [1]. Для контроля соблюдения этих требований может использоваться либо профессиональное оборудование, либо оборудование для домашнего использования. Первое настроено только на определённый набор датчиков, а масштабирование функционала может быть затруднительным или даже невозможным. Второе часто не обеспечивает необходимой точности и возможности интеграции в более крупные системы.

Для решения недостатков первого способа контроля микроклимата вместо закрытого блока управления можно использовать открытую аппаратную платформу, выполняющую аналогичный функционал, но с возможностью изменения логики управления. Возможность масштабирования и использование открытых протоколов связи решают недостатки второго способа контроля микроклимата, позволяя подобрать датчики с необходимой точностью и в дальнейшем интегрировать комплекс в более крупные системы.

Целью данной работы является разработка программно-аппаратного комплекса для измерения и контроля параметров микроклимата – температуры воздуха, относительной влажности воздуха и содержания углекислого газа.

1 Анализ предметной области

Предметной областью выполняемой работы являются программно-аппаратные системы мониторинга микроклимата в закрытых помещениях.

Микроклимат – состояние внутренней среды помещения, оказывающее воздействие на человека [1]. Микроклимат характеризуется следующими параметрами:

– температура воздуха – величина, отражающая степень нагрева воздуха;

– относительная влажность воздуха – величина, характеризующая соотношение текущего содержания водяного пара в воздухе к максимально возможному при данной температуре воздуха;

– концентрация углекислого газа в воздухе – величина, отражающая объем углекислого газа в единице объема воздуха;

– скорость движения воздуха – величина, показывающее пройденное расстояние воздушного потока в единицу времени;

– результирующая температура помещения – комплексный показатель радиационной температуры помещения и температуры воздуха помещения [1];

– локальная асимметрия результирующей температуры – разность результирующих температур в точке помещения [1].

В помещениях жилых и общественных зданий следует обеспечивать оптимальные или допустимые параметры микроклимата. Допустимые параметры микроклимата – сочетания значений показателей микроклимата, которые при длительном и систематическом воздействии на человека могут вызвать общее и локальное ощущение дискомфорта, ухудшение самочувствия и понижение работоспособности при усиленном напряжении механизмов терморегуляции и не вызывают повреждений или ухудшения состояния здоровья [1]. Оптимальные параметры микроклимата – сочетание значений показателей микроклимата, которые при длительном и систематическом воздействии на человека обеспечивают нормальное тепловое состояние организма при минимальном

напряжении механизмов терморегуляции и ощущение комфорта не менее чем у 80% людей, находящихся в помещении [1]. Оптимальные и допустимые параметры микроклимата регламентируются ГОСТ 30494-2011 «Межгосударственный стандарт. Здания жилые и общественные. Параметры микроклимата в помещениях».

Требуемые параметры микроклимата (оптимальные, допустимые или их сочетания) следует устанавливать в зависимости от назначения помещения и периода года [1]. По назначению помещения распределены по следующим категориям:

– помещения 1-й категории – помещения, в которых люди в положении лежа или сидя находятся в состоянии покоя и отдыха;

– помещения 2-й категории – помещения, в которых люди заняты умственным трудом, учебой;

– помещения 3а категории – помещения с массовым пребыванием людей, в которых люди находятся преимущественно в положении сидя без уличной одежды;

– помещения 3б категории – помещения с массовым пребыванием людей, в которых люди находятся преимущественно в положении сидя в уличной одежде;

– помещения 3в категории – помещения с массовым пребыванием людей, в которых люди находятся преимущественно в положении стоя без уличной одежды;

– помещения 4-й категории – помещения для занятий подвижными видами спорта;

– помещения 5-й категории – помещения, в которых люди находятся в полураздетом виде (раздевалки, процедурные кабинеты, кабинеты врачей и т.п.);

– помещения 6-й категории – помещения с временным пребыванием людей (вестибюли, гардеробные, коридоры, лестницы, санузлы, курительные, кладовые);

– помещения с постоянным пребыванием людей – помещение, в котором люди находятся не менее 2 ч непрерывно или 6 ч суммарно в течение суток.

В данной главе сформулированы требования и поставлена задача дипломной работы, выполнен анализ существующих систем мониторинга микроклимата в закрытых помещениях.

1.1 Постановка задачи

Необходимо разработать программно-аппаратный комплекс, позволяющий измерять и анализировать параметры микроклимата.

Комплекс должен отвечать следующим требованиям:

– комплекс должен осуществлять измерение и анализ параметров микроклимата, установленных межгосударственным стандартом параметров микроклимата в помещениях – температура воздуха, относительная влажность воздуха, содержание углекислого газа [1];

– диапазон измерений должен соответствовать следующим требованиям:

а) диапазон измерения температуры внутреннего воздуха от 5°С до 40°С;

б) диапазон измерения относительной влажности воздуха от 10% до 90%;

в) диапазон измерения концентрации углекислого газа от 400 см3/м3 до 2000 см3/м3;

– критерии анализа параметров микроклимата должны соответствовать нормам, установленным межгосударственным стандартом параметров микроклимата в помещениях:

а) в холодный период года:

1) помещения 1-й категории. Оптимальная температура воздуха – 20-22°С. Допустимая температура воздуха – 18-24°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

2) помещения 2-й категории. Оптимальная температура воздуха – 19-21°С. Допустимая температура воздуха – 18-23°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

3) помещения 3а категории. Оптимальная температура воздуха – 20-21°С. Допустимая температура воздуха – 19-23°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

4) помещения 3б категории. Оптимальная температура воздуха – 14-16°С. Допустимая температура воздуха – 12-17°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

5) помещения 3в категории. Оптимальная температура воздуха – 18-20°С. Допустимая температура воздуха – 16-22°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

6) помещения 4-й категории. Оптимальная температура воздуха – 17-19°С. Допустимая температура воздуха – 15-21°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

7) помещения 5-й категории. Оптимальная температура воздуха – 20-22°С. Допустимая температура воздуха – 20-24°С. Оптимальная относительная влажность – 30-45%. Допустимая относительная влажность – не более 60%;

8) помещения 6-й категории. Оптимальная температура воздуха – 16-18°С. Допустимая температура воздуха – 14-20°С;

б) в теплый период года:

1) помещения с постоянным пребыванием людей. Оптимальная температура воздуха – 23-25°С. Допустимая температура воздуха – 18-28°С. Оптимальная относительная влажность – 30-60%. Допустимая относительная влажность – не более 65%;

в) независимо от периода года:

1) помещения 1-й категории. Допустимое содержание углекислого газа – 800 $см^3/м^3$ и менее;

2) помещения 2-й категории. Допустимое содержание углекислого газа – 800-1000 $см^3/м^3$;

3) помещения 3-й категории. Допустимое содержание углекислого газа – 1000-1400 $см^3/м^3$;

4) помещения 4-й категории. Допустимое содержание углекислого газа – 1400 $см^3/м^3$ и более;

– аппаратная часть комплекса должна включать в себя следующие связанные между собой компоненты:

а) коммутационный стенд – предназначен для соединения датчиков с открытой аппаратной платформой;

б) открытая аппаратная платформа – используется для создания электронного компонента управления;

– программная часть комплекса должна включать в себя следующие связанные между собой компоненты:

а) сервер – предназначен для управления аппаратной частью комплекса: инициализация датчиков, запуск измерений, сбор результатов измерений;

б) клиент – используется для визуализации и анализа измерений, полученных от сервера, фиксации и структурировании информации в текстовых файлах;

– для обеспечения связи между компонентами комплекса необходимо использовать протоколы последовательной связи: I2C, UART, SPI и др.;

– сервер должен обеспечивать надежный обмен данными между другими компонентами комплекса, используя механизмы протоколов связи;

– клиент должен предоставлять пользователю возможность изменения критериев анализа параметров в зависимости от категории помещения и других условий [1];

– программная часть комплекса должна работать под ОС Windows версии 10 и выше.

При разработке программно-аппаратного комплекса следует учитывать исключительные ситуации и создавать механизмы для их обработки. Алгоритм работы комплекса отражает UML-диаграмма деятельности, которая представлена в приложении А.1.

1.2 Анализ аналогов

Рынок систем измерения и контроля параметров микроклимата характеризуется разнообразием производителей и систем. На рынке представлены как зарубежные, так и отечественные производители. Всё множество систем можно разделить на два типа: профессиональное оборудование и бытовые датчики. Чтобы проанализировать предметную область данной работы, следует рассмотреть некоторых производителей систем измерения и контроля микроклимата.

Из отечественных производителей профессиональных систем мониторинга микроклимата можно выделить Научно-Техническое предприятие «ТКА». Его зарубежным аналогом является компания Testo Instruments Co. Ltd. В быту наибольшую популярность приобрели приборы измерения микроклимата фирмы Xiaomi.

Научно-Техническое предприятие «ТКА» занимается разработкой и производством профессиональных систем мониторинга микроклимата, которые

прошли Государственные испытания, включены в государственный Реестр средств измерений и рекомендованы для оснащения организаций, осуществляющих контрольные и надзорные функции [2].

Компания Testo Instruments Co. Ltd. специализируется на производстве измерительных приборов и регистраторов. Разработанная компанией экосистема позволяет объединять приборы и регистраторы для более точного измерения и контроля микроклимата [3].

Фирма Xiaomi разрабатывает и производит широкий спектр бытовых приборов, включая системы мониторинга микроклимата. Такие системы просты в использовании, а собственная экосистема «Умный дом» является ключевым критерием для покупателей при выборе бытовых приборов измерения [4].

Для сравнительного анализа следует сформулировать следующие характеристики систем мониторинга: тип систем, измеряемые параметры, поддерживаемые интерфейсы, наличие ПО, особенности и недостатки. Тип системы определяет архитектуру и принцип работы системы. Измеряемые параметры системы включают в себя величины, отслеживаемые системой. Список поддерживаемых интерфейсов показывает возможные сценарии интегрирования системы. Наличие ПО обеспечивает систему возможностью настройки, отображения и хранения данных.

Сравнение систем измерения и контроля микроклимата представлено в таблице 1.

Из таблицы видно, что общий недостаток рассмотренных систем измерения и контроля микроклимата – строго определённый набор датчиков для измерения и отсутствие возможности расширения, например, интеграции других датчиков, или ограничения излишнего функционала. Использование открытой аппаратной платформы в аппаратной части разрабатываемого комплекса позволяет реализовать поддержку широкого набора датчиков и расширить функционал всего комплекса.

Таблица 1 – Сравнительные характеристики комплексов измерения и контроля микроклимата различных производителей

Характеристика	Производители систем измерения и контроля микроклимата		
	Научно-техническое предприятие «ТКА»	Testo Instruments Co. Ltd.	Xiaomi
Тип систем	Профессиональные приборы для определения показателей микроклимата	Комбинированные измерители, измерители-регистраторы	Бытовая система датчиков измерения микроклимата
Измеряемые параметры	Температура, влажность воздуха, скорость движения воздуха, освещённость, атмосферное давления	Температура, влажность воздуха, скорость движения воздуха, освещённость, атмосферное давления	Температура, влажность
Интерфейсы	Wi-Fi, USB	Bluetooth, USB, WLAN	Bluetooth
ПО	ПО для чтения сохранённых данных, ПО для мониторинга показаний	ПО для чтения и архивации данных	Мобильное ПО для мониторинга показаний
Особенности	Многие системы внесены в Госреестр средств измерений	Собственная экосистема для объединения нескольких измерителей	Собственная экосистема «Умный дом»
Недостатки	Набор датчиков для измерения определен производителем, нет возможности масштабирования функционала	Набор датчиков для измерения определен производителем	Набор датчиков для измерения определен производителем, нет возможности масштабирования функционала

1.3 Выбор технологий разработки

1.3.1 Выбор датчиков

Назначение датчиков в разрабатываемом комплексе – измерение таких параметров микроклимата, как температура воздуха, относительная влажности воздуха и содержание углекислого газа. При выборе датчиков необходимо учитывать следующие критерии:

– датчик должен измерять один или несколько параметров, относящихся к параметрам микроклимата;

– датчик должен соответствовать диапазонам измерений вышеописанных требований;

– выбранные датчики должны поддерживать одинаковый интерфейс и цифровой вывод.

С учетом требований можно сформулировать следующие характеристики датчиков: диапазон измерений и интерфейс связи. Дополнительно можно учитывать их примерную стоимость и источник ценовых данных.

Сравнение датчиков температуры и влажности представлено в таблице 2.

Таблица 2 – Сравнительные характеристики датчиков влажности и температуры

Модель датчика	Характеристики			
	Диапазон	Интерфейс	Примерная стоимость	Источник ценовых данных
AHT21	-40..+120°C 0..100% RH	I2C	~300 руб.	Маркетплейс «Ozon»
AHT10	-40..+80°C 0..100% RH	I2C	~500 руб.	Маркетплейс «Ozon»
DHT22	-40..+80°C 0..100% RH	Single Wire	~400 руб.	Маркетплейс «Ozon»
HDC1080	-40..+125°C 0..100% RH	I2C	~500 руб.	Маркетплейс «Ozon»

Датчик модели AHT21 при невысокой стоимости относительно других аналогов обладает достаточными характеристиками для измерения температуры и относительной влажности воздуха. Учитывая эти преимущества, был выбран датчик модели AHT21.

Сравнение датчиков концентрации углекислого газа представлено в таблице 3.

Таблица 3 – Сравнительные характеристики датчиков концентрации углекислого газа

Модель датчика	Характеристики			
	Диапазон, $см^3/м^3$	Интерфейс	Примерная стоимость	Источник ценовых данных
MH-Z19B	400..5000	UART/PWM	~600 руб.	Маркетплейс «Ozon»
SCD40	400..2000	I2C	~900 руб.	Маркетплейс «Ozon»
SCD41	400..5000	I2C	~1200 руб.	Маркетплейс «Ozon»
SenseAir S8	400..2500	UART/I2C	~1500 руб.	Маркетплейс «Ozon»

Датчик модели MH-Z19B соответствует требуемым диапазонам измерений и обладает низкой стоимостью среди других датчиков, однако интерфейс датчика не удовлетворяет условию единства интерфейсов набора датчиков. Датчик модели SCD40 отличается средней стоимостью аналогов, а его интерфейс совпадает с интерфейсом датчика AHT21, поэтому в качестве датчика концентрации углекислого газа был выбран датчик модели SCD40.

1.3.2 Выбор открытой аппаратной платформы

Назначение открытой аппаратной платформы в разрабатываемом комплексе – создание электронного компонента управления, который должен обеспечивать связь между программной аппаратной частями комплекса.

Главное преимущество такой платформы – возможность изменения логики управления, что позволяет модернизировать и конфигурировать аппаратную часть комплекса, добавлять механизмы отладки и интегрировать комплекс в более крупные системы измерения.

Наиболее распространенными открытыми аппаратными платформами являются микроконтроллеры и ПЛИС. При выборе решения необходимо учитывать следующие критерии:

– возможность параллельной работы для одновременного управления программной и аппаратной частями комплекса;

– строгое соблюдение временных характеристик для синхронизации интерфейсов связи;

– возможность реализации логики поддержки аппаратных интерфейсов связи.

С учетом критериев можно сформулировать следующие сравнительные характеристики для аппаратных платформ: параллелизм, детерминизм временных параметров и гибкость интерфейсов. Дополнительно можно учитывать их архитектуру.

Сравнение открытых аппаратных платформ представлено в таблице 4.

Сравнение аппаратных платформ показывает, что ПЛИС обладает преимуществами по сравнению с микроконтроллером и удовлетворяет вышеописанными критериями. Поддержка параллелизма позволит реализовать независимое управление программной и аппаратной частями комплекса. Детерминированность обеспечит строгую временную стабильность работы интерфейсов связи. Гибкость интерфейсов исключит избыточное использование ресурсов, поскольку будет реализован только необходимый функционал. Учитывая все преимущества и недостатки представленных платформ, была выбрана ПЛИС. Кроме того, в распоряжении имелась отладочная плата AX309 с ПЛИС XC6SLX16-2FTG256C, которая и была выбрана в качестве электронного компонента управления для этого комплекса.

Таблица 4 – Сравнительные характеристики открытых аппаратных платформ

Характеристики	Аппаратные платформы	
	Микроконтроллер	ПЛИС
Архитектура	Архитектура платформы и внутренние связи между компонентами определены на этапе производства	Архитектура платформы подразумевает проектирование внутренних связей между компонентами логическими блоками CLB
Параллелизм	Программа выполняется последовательно. Псевдопараллелизм может быть реализован с использованием механизма прерываний	Задача выполняется параллельно благодаря параллельной структуре логических блоков
Детерминизм	Зависит от прерываний, поскольку обработка прерываний и переключение контекста приводит к задержкам в работе	Временные параметры фиксированы и определены частотой тактового синхросигнала. Все события выполняются на каждый цикл тактового сигнала
Гибкость интерфейсов	Аппаратно ограниченный набор интерфейсов, заданный на этапе производства	Возможность реализации необходимых интерфейсов проектированием внутренних связей

1.3.3 Выбор инструментов разработки аппаратной платформы

В инструменты разработки аппаратной платформы входят среда разработки и язык программирования логической схемы.

Одним из лидеров рынка программируемой логики является компания Xilinx, которая кроме производства открытых аппаратных платформ, конкретнее ПЛИС, занимается созданием сред разработки. В настоящее время Xilinx приобретена компанией AMD, но разработка новых платформ продолжается. Поскольку при разработке используется аппаратная платформа фирмы Xilinx, следует рассмотреть именно её среды разработки – Xilinx ISE Design Suite и Xilinx Vivado Design Suite. Главным критерием при выборе среды разработки – поддержка используемой при разработке комплекса серии ПЛИС.

Xilinx ISE Design Suite – первая среда разработки компании Xilinx, поддержка которой прекращена в 2013 году. Xilinx ISE включает в себя следующие инструменты:

– синтезатор XST – используется для преобразования исходного описания схемы в список логических связей;

– симулятор iSim – используется для моделирования и отладки работы схемы;

– набор средств реализации проекта – включает в себя инструменты объединения логических соединений, проецирования логической схемы на физические ресурсы конкретного ПЛИС, размещения и соединения логических блоков;

– генератор файла конфигурации – используется создания файла конфигурации, который может быть загружен в ПЛИС.

Из преимуществ данной среды разработки можно выделить бесплатную лицензию, выдаваемую компанией AMD, и поддержку семейств ПЛИС, заканчивая Spartan-6. Основным недостатком Xilinx ISE является отсутствие официальной поддержки.

Xilinx Vivado Design Suite – пришедший на смену Xilinx ISE среда разработки компании Xilinx, использующий полностью переработанный процесс проектирования. Xilinx Vivado включает в себя следующие инструменты:

– синтезатор HSL – используется для преобразования исходного описания схемы в список логических связей;

– симулятор Vivado Simulator – используется для моделирования и отладки работы схемы;

– отладчик ILA – используется для отладки проекта после реализации;

– набор средств реализации проекта – включает в себя инструменты объединения логических соединений, проецирования логической схемы на физические ресурсы конкретного ПЛИС, размещения и соединения логических блоков;

– генератор файла конфигурации – используется создания файла конфигурации, который может быть загружен в ПЛИС.

Из преимуществ Xilinx Vivado можно выделить более современный и улучшенный алгоритм проектирования и синтеза. Главным недостатком данной среды разработки является несовместимость с семействами Spartan-6.

Xilinx ISE соответствует требуемому критерию поддержки семейств ПЛИС, поэтому в качестве среды разработки был выбран Xilinx ISE Design Suite.

Выбор языка программирования, а корректнее языка описания аппаратуры, прямо зависит от среды разработки, поскольку определенные среды разработки поддерживают только ограниченный набор языков программирования. Выбранная Xilinx ISE Design Suite поддерживает следующие два стандарта:

– VHDL – отличается строгой типизацией и сложным синтаксисом. Каждый созданный модуль четко разделяются на интерфейс, описывающий выводы модуля, и архитектуру, описывающую функционал модуля;

– Verilog – обладает менее строгой типизацией и ограниченным набором базовых типов данных.

Язык описания аппаратуры VHDL за счет более строгой типизации снижает риск ошибок на этапе проектирования конечного автомата для управления датчиками, что является преимуществом по сравнению с Verilog, поэтому в качестве языка программирования был выбран VHDL.

1.3.4 Установка и настройка Xilinx ISE Design Suite

Установщик среды разработки можно получить бесплатно на официальном сайте компании Xilinx. В архиве представлено множество версий, однако для стабильной работы на последних версиях операционной системы Windows разработчик рекомендует устанавливать последнюю версию Xilinx ISE Design Suite 14.7. На официальном сайте представлена различная документация, в том числе про процесс установки среды разработки. При запуске установочного файла появляется приветственное окно, которое представлено на рисунке 1.

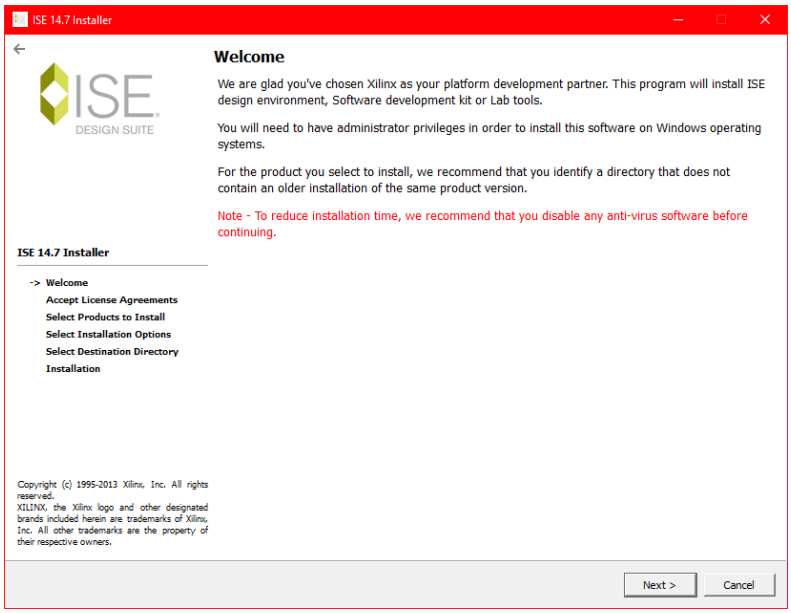


Рисунок 1 – Приветственное окно установщика

Следующее окно с первой частью лицензионного соглашения, которое необходимо принять для продолжение установки, продемонстрировано на рисунке 2.

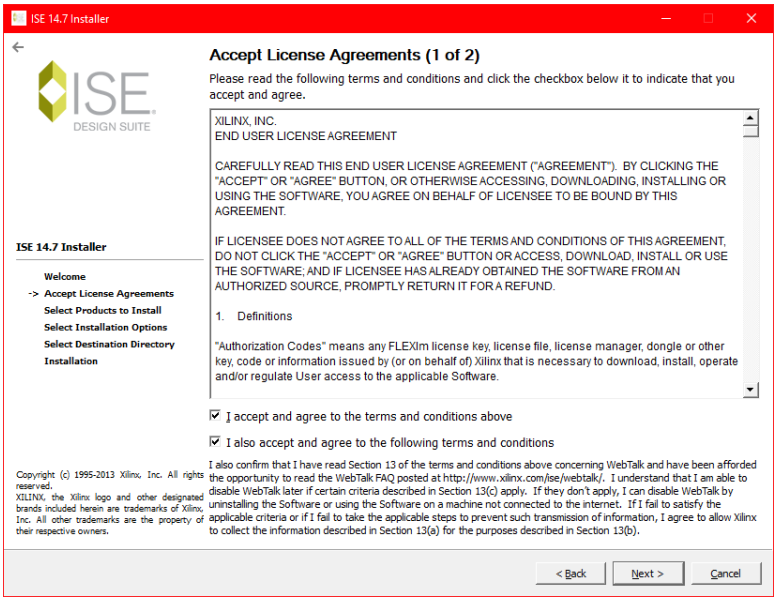


Рисунок 2 – Окно с первой частью лицензионного соглашения

После ознакомления с первой частью лицензионного соглашения открывается окно со второй частью лицензионного соглашения, представленного на рисунке 3.

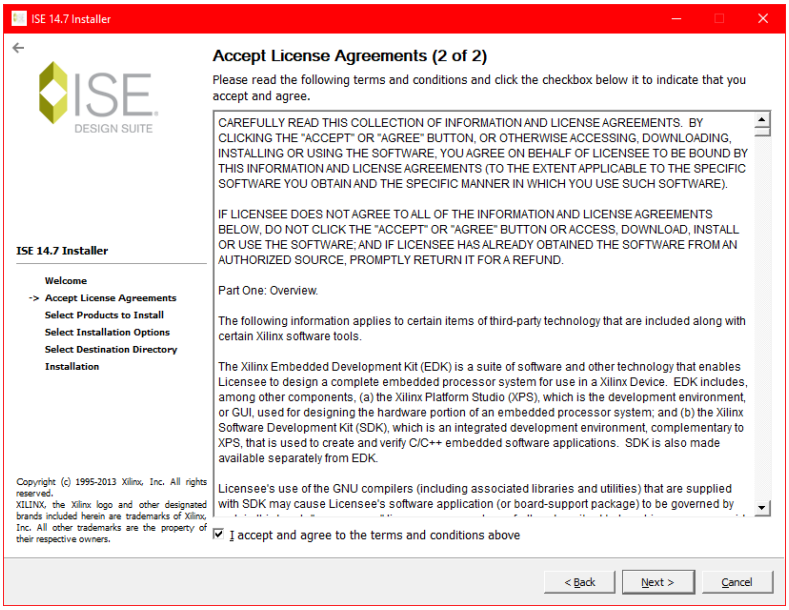


Рисунок 3 – Окно со второй частью лицензионного соглашения

Следующее окно с выбором продукта Xilinx ISE, в котором был выбран продукт ISE WebPack продемонстрировано на рисунке 4.

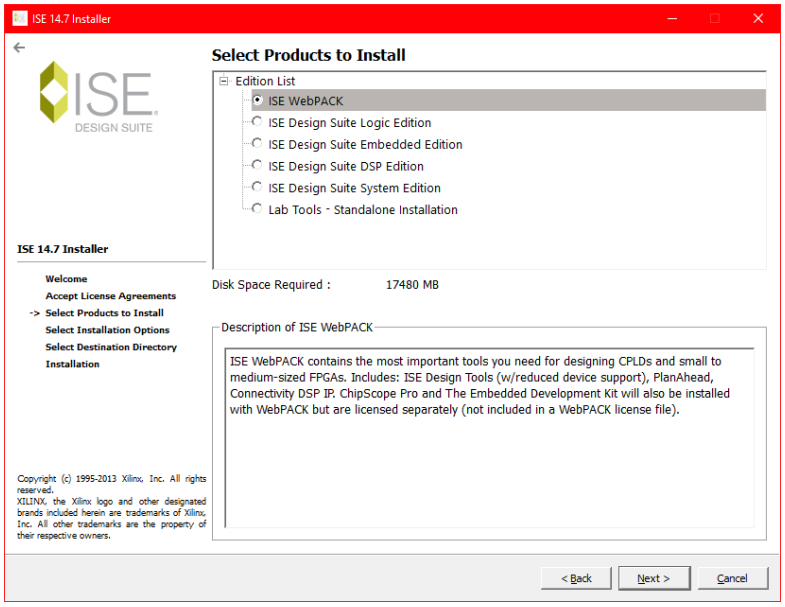


Рисунок 4 – Окно выбора продукта Xilinx ISE

После выбора продукта установщик предлагает настроить процесс установки, выбрав опциональную установку драйверов и использование нескольких ядер процессора для ускорения установки. Окно настройки установки представлено на рисунке 5

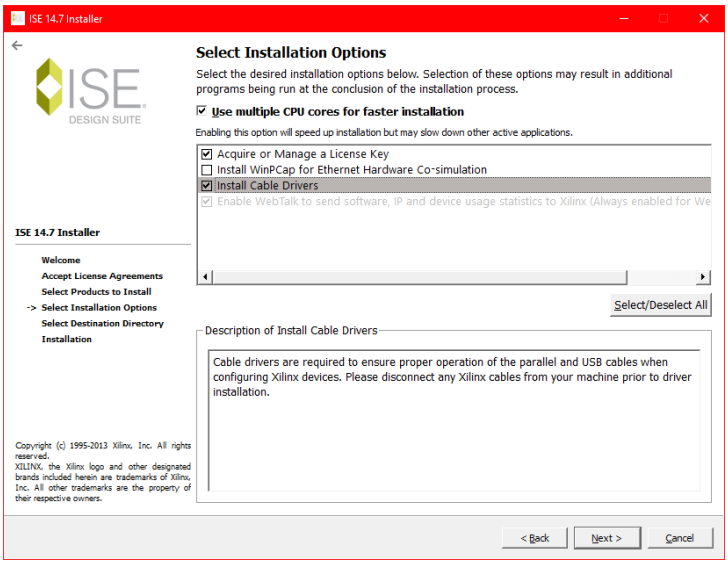


Рисунок 5 – Окно настройки установки

Предпоследним шагом установки среды разработки является выбор пути, в котором будет расположена программа. Окно выбора пути установки продемонстрировано на рисунке 6.

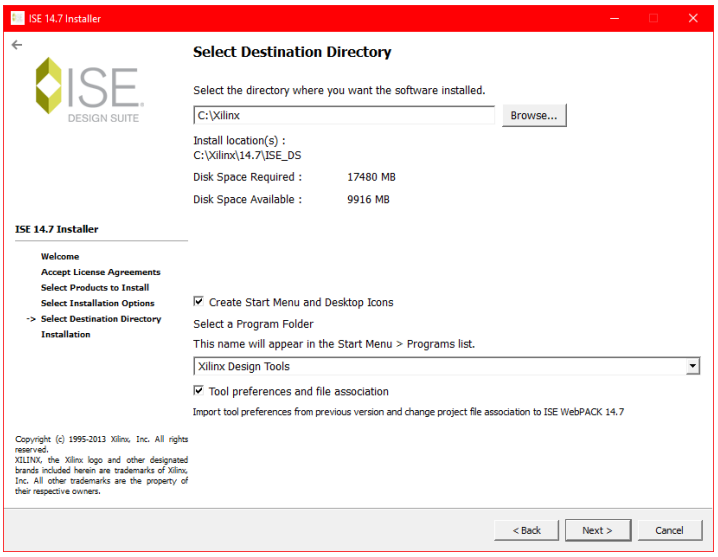


Рисунок 6 – Окно выбора пути установки

Завершающим этапом установки является копирование файлов среды разработки по указанному пути. Процесс копирования файлов представлен на рисунке 7

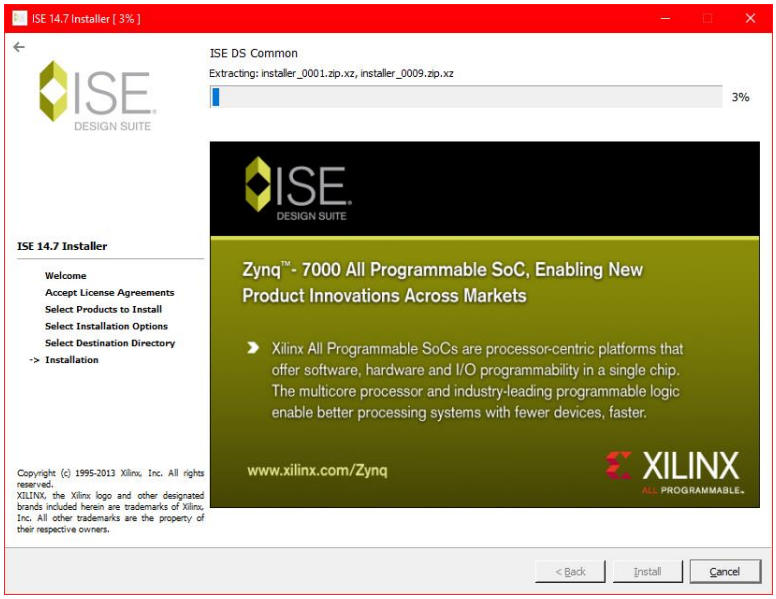


Рисунок 7 – Окно копирования файлов среды разработки

После успешного завершения установки можно запускать программу для создания проектов Xilinx ISE Project Navigator. Общий вид программы продемонстрирован на рисунке 8.

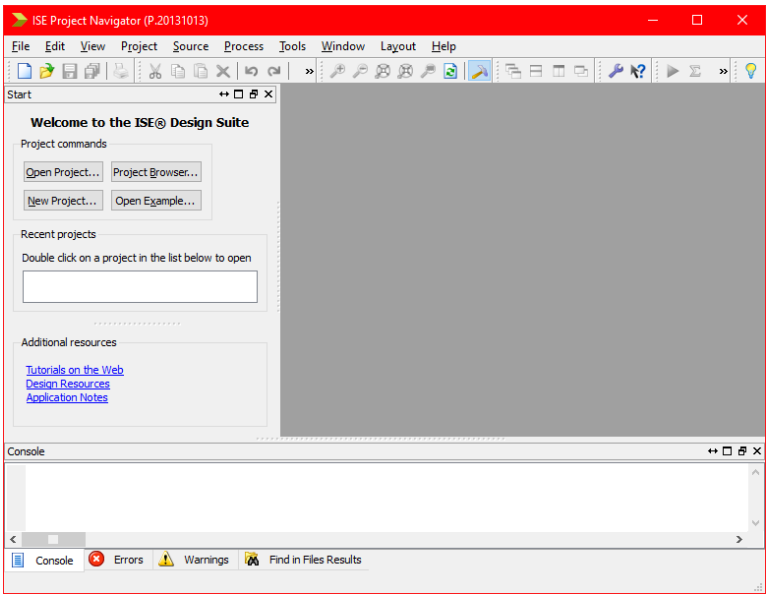


Рисунок 8 – Общий вид программы Xilinx ISE Project Navigator

Настройка среды разработки осуществляется в том числе при создании нового проекта. В первом окне указывается название проекта, расположение и тип проекта. Окно инициализации проекта представлено на рисунке 9.

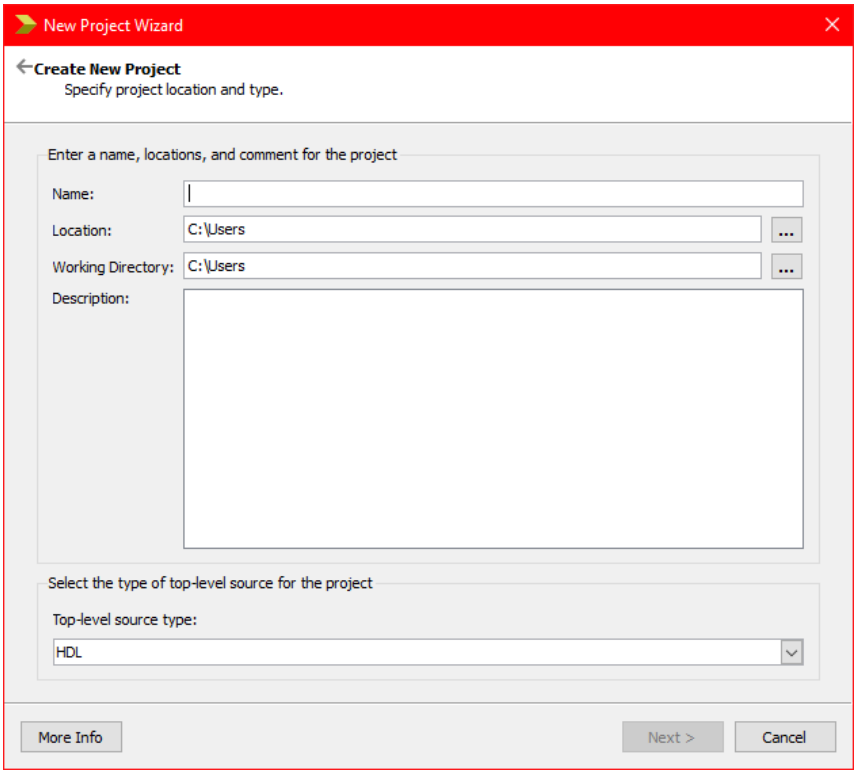


Рисунок 9 – Окно инициализации проекта

Следующее окно предназначено для выбора семейства ПЛИС, на котором будет проектироваться логическая схема. Окно позволяет выбрать инструмент синтеза, инструмент симуляции, предпочитаемый язык описания аппаратуры, который будет использоваться при создании логических модулей. При необходимости можно изменить свойства сборки проекта, стандарт анализа исходных файлов. Окно настройки свойств проекта продемонстрировано на рисунке 10.

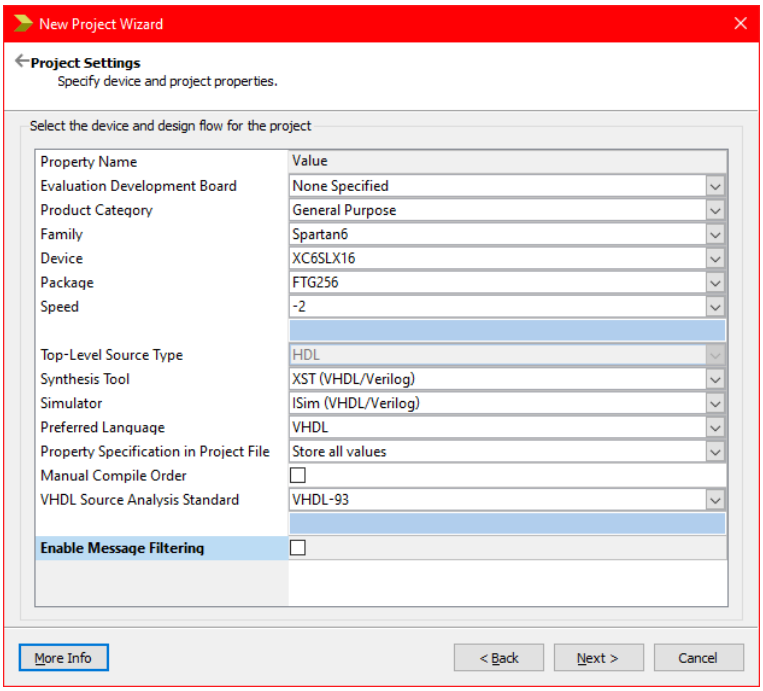


Рисунок 10 – Окно настройки свойств проекта

Перед записью готового конфигурационного файла на ПЛИС необходимо настроить среду разработки. Для этого следует открыть инструмент iMPACT, вид которого продемонстрирован на рисунке 11.

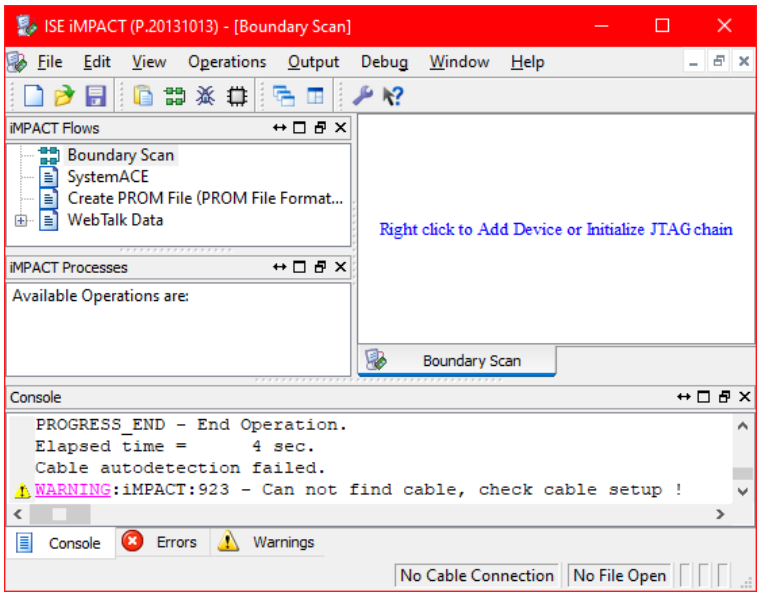


Рисунок 11 – Общий вид окна инструмента iMPACT

Для правильного соединения с ПЛИС требуется выбрать тип соединения Platform Cable USB/II, открыв окно настройки соединения, вид которого показан на рисунке 12.

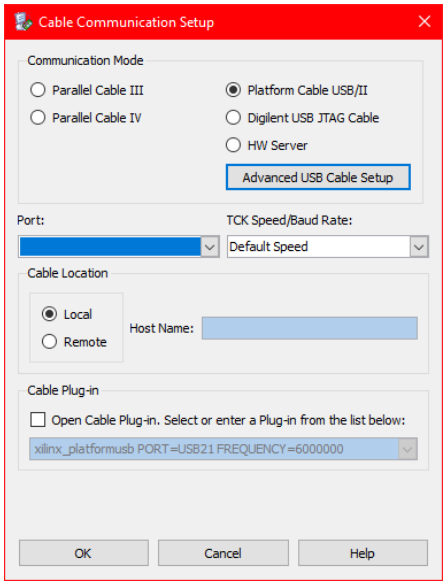


Рисунок 12 – Окно настройки соединения с ПЛИС

После успешной настройки и выполнения соединения окно инструмента iMPACT примет вид, представленный на рисунке 13.

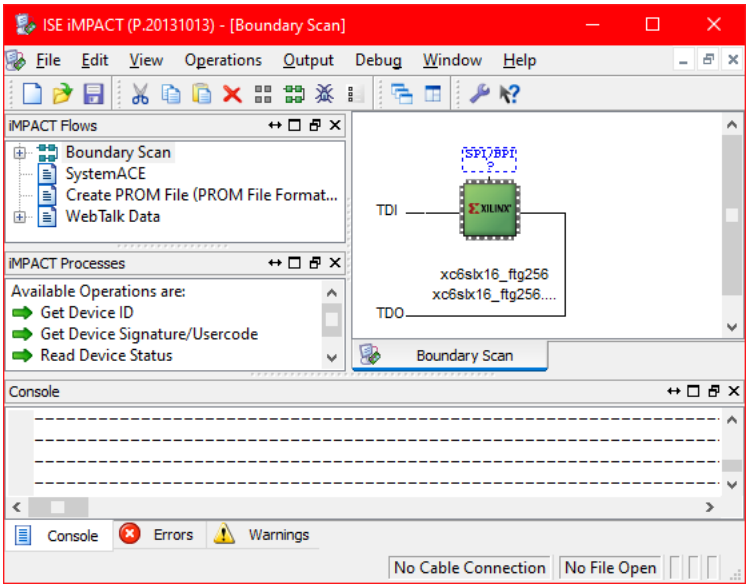


Рисунок 13 – Вид окна инструмента iMPACT при успешном подключении к ПЛИС

Правильно настроенная среда разработки Xilinx ISE Design Suite позволяет разрабатывать логические схемы и загружать их на открытую аппаратную платформу без ошибок.

1.3.5 Выбор инструментов разработки программной части

В инструменты разработки программной части комплекса входят среда разработки, языки программирования приложений клиента и сервера.

Серверное приложение программной части комплекса выполняет низкоуровневое взаимодействие с аппаратной частью, работу и обработку потока данных от клиентов. Для реализации этих функций был выбран язык C++, поскольку он обладает следующими преимуществами:

– высокая производительность и эффективность за счет полного контроля над системными ресурсами;

– низкоуровневый доступ к аппаратным интерфейсам с помощью библиотеки операционной системы Windows;

– конечное приложение является исполняемым файлом с минимальными внешними зависимостями, упрощая его развертывание;

– стандартные библиотеки предоставляют возможность реализации многопоточной программы.

Клиентское приложение программной части комплекса предоставляет пользователю интуитивно понятный интерфейс и визуализирует результаты измерения. Для реализации этих функций был выбран язык C# с использованием библиотеки Windows Forms, поскольку они обладают следующими преимуществами:

– стабильная платформа, позволяющая создавать классические приложения для операционной системы Windows;

– достаточный выбор элементов управления, возможность создания собственных элементов в виде классов;

– платформа содержит встроенные классы для сетевого взаимодействия.

Для реализации программной части комплекса была выбрана среда разработки Microsoft Visual Studio, главным преимуществом которой является поддержка технологий языков программирования. Кроме того, выбранная среда разработки обладает следующими преимуществами:

– интеграция с платформой Windows, поскольку целевой платформой среды является эта операционная система;

– встроенные инструменты разработки, отладки и поддержки кода;

– распространяется бесплатно для индивидуальных разработчиков и небольших некоммерческих команд.

1.3.6 Установка и настройка Microsoft Visual Studio

Установщик среды разработки Visual Studio можно загрузить бесплатно на официальном сайте компании Microsoft. На странице представлена последняя версия, при необходимости можно перейти в архив версий и загрузить более старые версии. При запуске установочного файла выводится окно, представленное на рисунке 14.

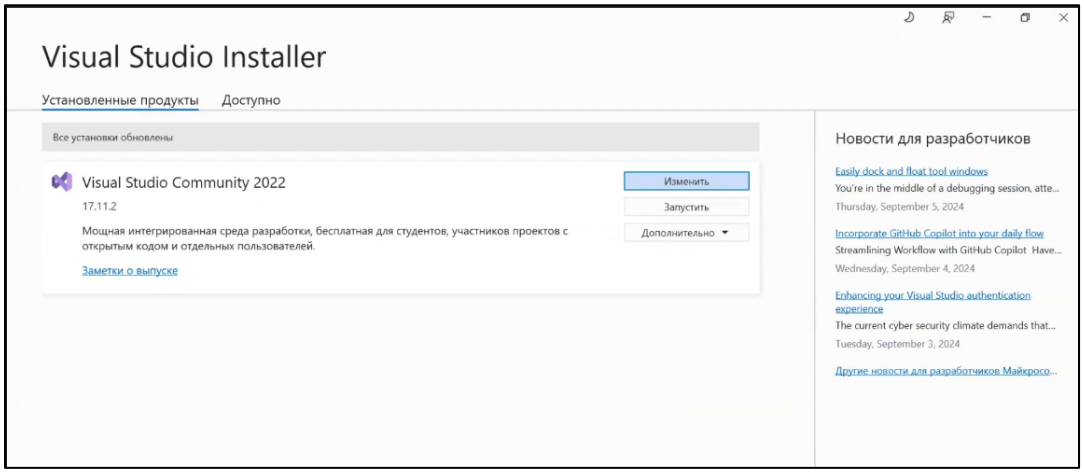


Рисунок 14 – Приветственное окно установщика

Используемая версия среды разработки Visual Studio Community 2022. Кроме установки программы, необходимо загрузить дополнительные компоненты, которые позволят создавать приложения на платформе Windows Forms с использованием языка программирования C# и классические приложения для Windows с использованием C++. Окно загрузки компонентов показано на рисунке 15.

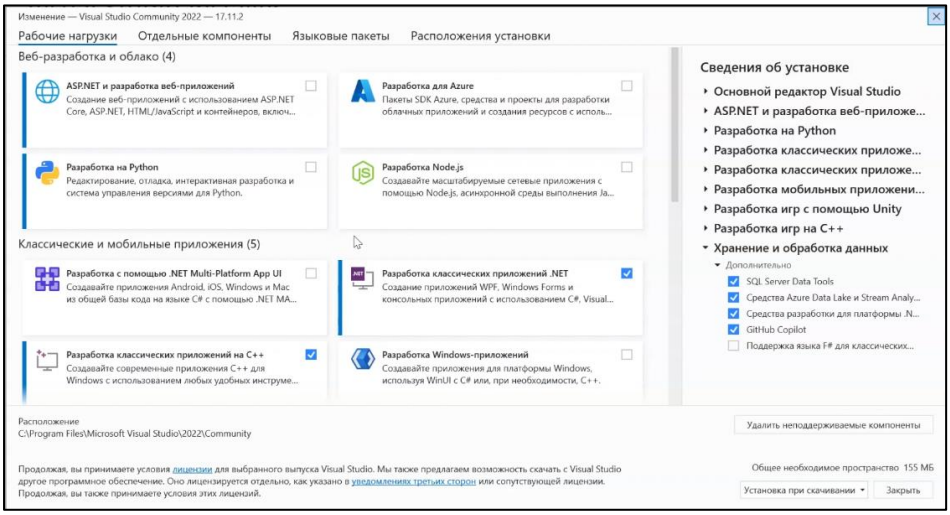


Рисунок 15 – Окно загрузки компонентов среды разработки

При загрузке компонентов окно установки примет вид, представленный на рисунке 16.

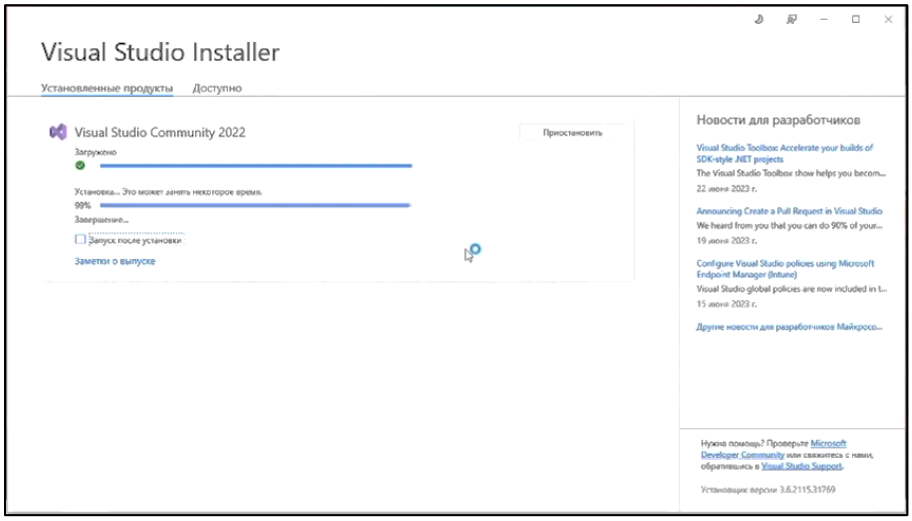


Рисунок 16 – Процесс загрузки и установки компонентов и среды разработки

После успешной установки предлагается запустить среду разработки. Общий вид программы продемонстрирован на рисунке 17.

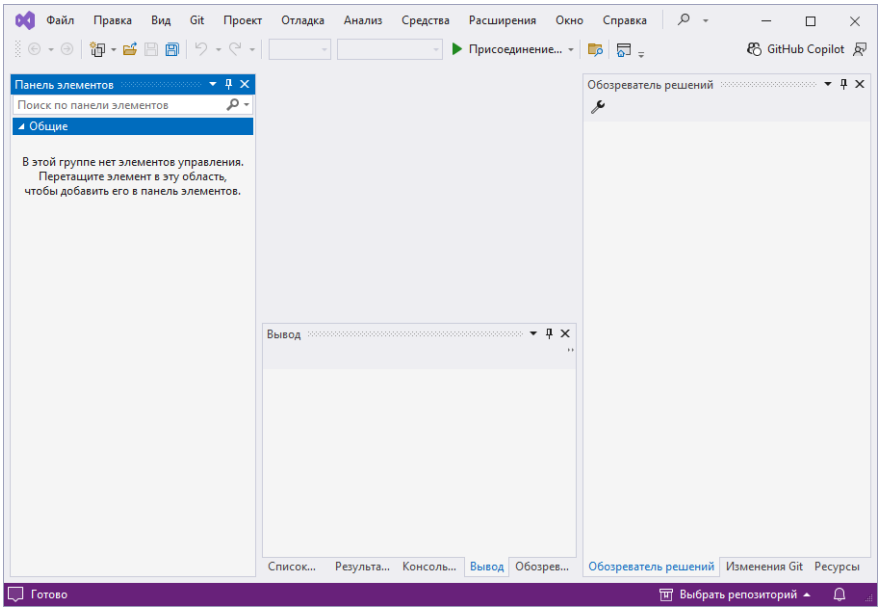


Рисунок 17 – Общий вид программы Visual Studio

Убедиться в успешной установке компонентов можно, вызвав окно создания проекта, представленного на рисунке 18.

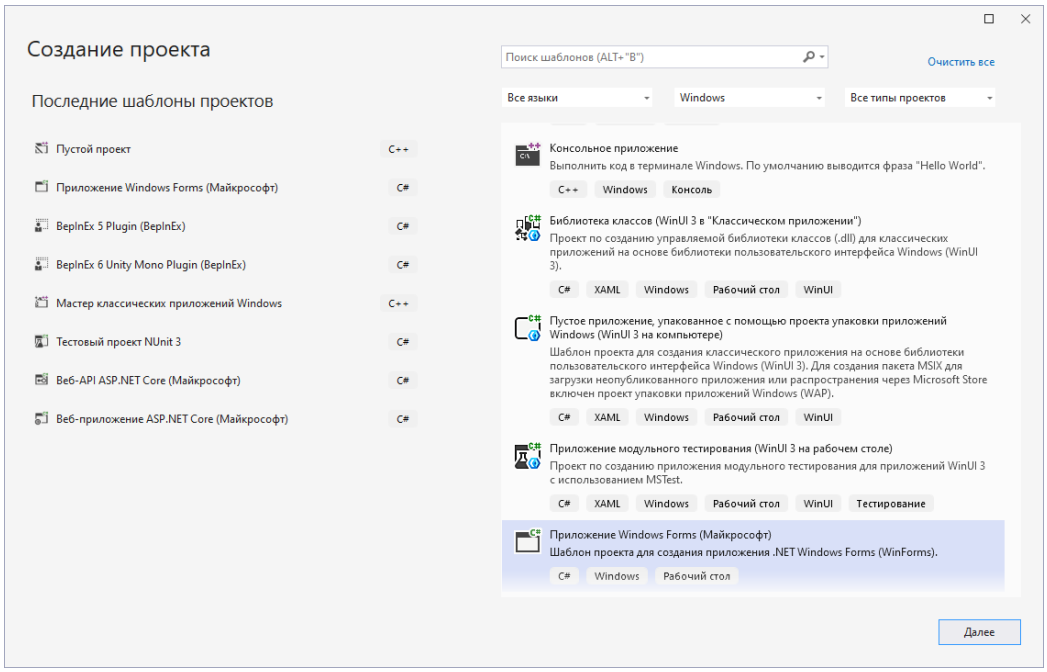


Рисунок 18 – Окно создания проекта

Правильно настроенная среда разработки Microsoft Visual Studio позволяет создавать клиентское приложения на платформе Windows Forms с использованием языка программирования C# и консольное серверное приложение с использованием языка C++.

2 Архитектура системы

Архитектура разрабатываемого программно-аппаратного комплекса включает аппаратное и программное обеспечение. Эти компоненты системы тесно взаимосвязаны и функционируют как единое целое, причем масштабируемость программно-аппаратного комплекса позволяет расширить возможности и функциональность каждого из компонентов. Структурная схема комплекса представлена на рисунке 19.

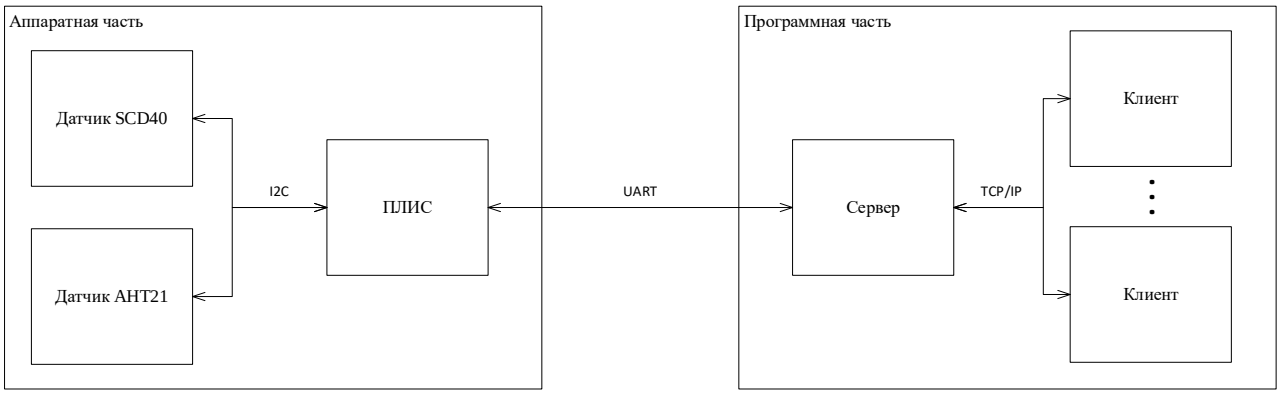


Рисунок 19 – Структурная схема программно-аппаратного комплекса

Аппаратная часть комплекса реализует функции управления сбором и первичной обработкой данных. Программная часть комплекса управляет аппаратной частью комплекса, принимает и преобразует полученные данные, отображает данные и реализует алгоритм сравнения параметров микроклимата с нормами ГОСТ. Связь между программной и аппаратной частями обеспечивается последовательным протоколом UART.

В данной главе описана и продемонстрирована архитектура всего программно-аппаратного комплекса и архитектура отдельных компонентов этой системы.

2.1 Архитектура аппаратной части

Архитектура аппаратной части комплекса объединяет открытую аппаратную платформу ПЛИС и датчики измерения параметров микроклимата AHT21 и SCD40. Структурная схема аппаратной части комплекса продемонстрирована на рисунке 20.

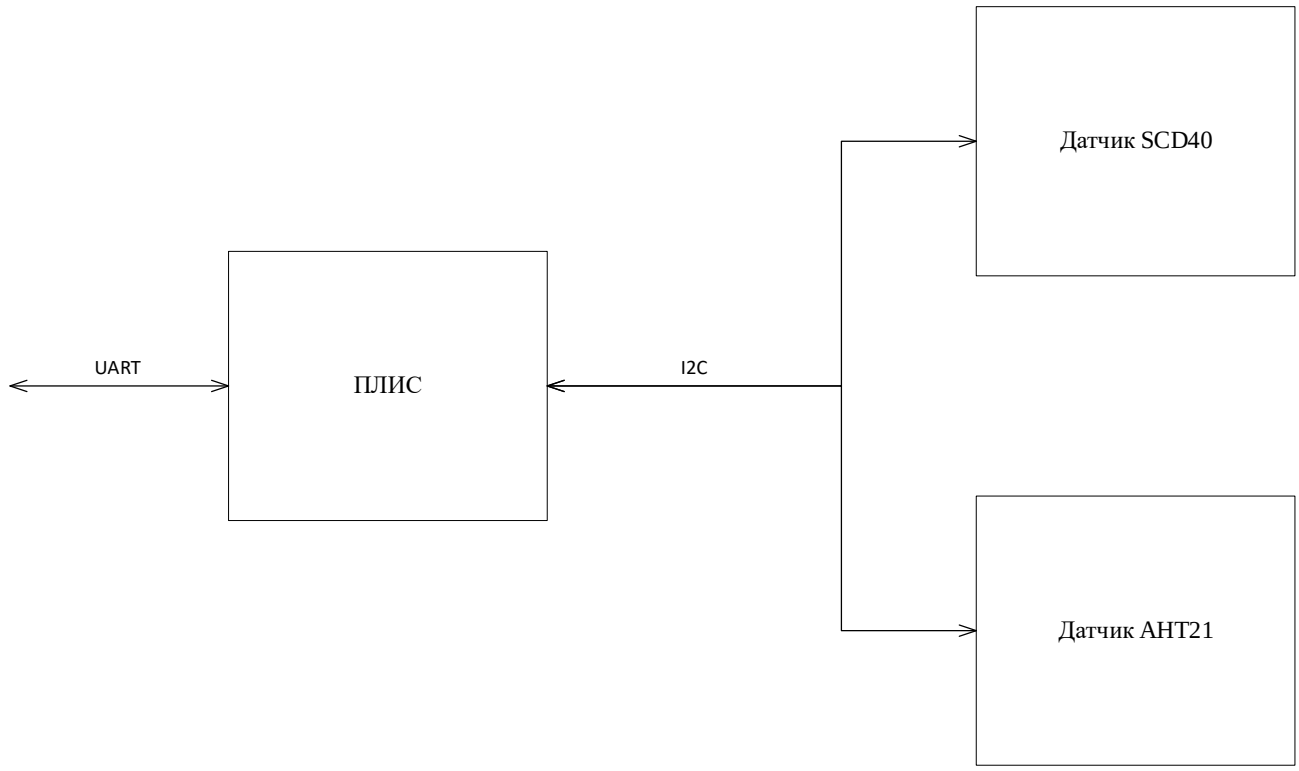


Рисунок 20 – Структурная схема аппаратной части комплекса

Датчик AHT21 измеряет температуру и относительную влажность воздуха, датчик SCD40 предназначен для измерения концентрации углекислого газа в воздухе. ПЛИС управляет сбором информации с датчиков, осуществляет первичную обработку и дальнейшую передачу данных на подключенный компьютер с использованием протокола UART. Для связи между ПЛИС и датчиками используется интерфейс I2C.

2.2 Архитектура программной части

Архитектура программной части комплекса представляет собой клиент-серверную архитектуру и состоит из сервера и клиента. Структурная схема программной части комплекса представлена на рисунке 21.

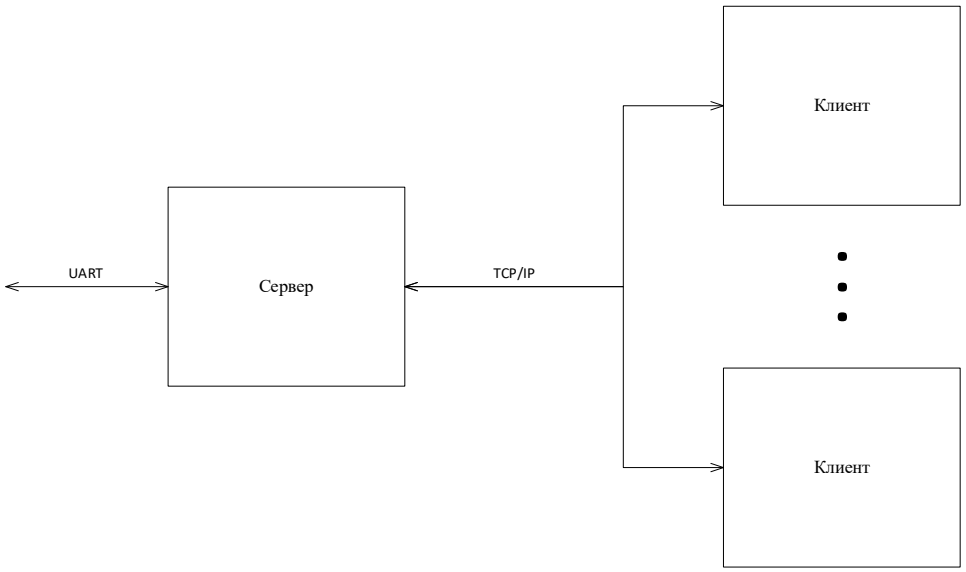


Рисунок 21 – Структурная схема программной части комплекса

Сервер представляет собой устройство, на котором развёрнута программа-сервер. Он управляет аппаратной частью комплекса: запускает сбор данных с определённого датчика, преобразует полученный массив данных в набор измеренных параметров и обрабатывает запросы от клиента. Клиентом является устройство с запущенной на нём программой-клиентом. Клиент запрашивает результаты измерения у сервера, проверяет соответствие параметров нормам ГОСТ и отображает полученную информацию пользователю через графический интерфейс. Связь между клиентом и сервером обеспечивается сетевой моделью TCP/IP.

3 Описание разработки модулей

В процессе разработки модулей сначала проектируется, а затем реализуется заданный для каждого компонента функционал с учётом описанной архитектуры программно-аппаратного комплекса.

Проектирование включает в себя создание структурных схем для отображения связей между модулями и поведенческих схем для моделирования поведения модулей. В качестве метода проектирования выбран модульный подход, поскольку он подходит для описания как аппаратной, так и программной частей.

Реализация представляет собой программирование функционала, причем для каждой части программно-аппаратного комплекса используется определённый метод программирования:

– При разработке модулей ПЛИС используется метод восходящего программирования. Сначала реализуются модули нижнего уровня, затем на основе этих модулей создаются модули высокого уровня. Модуль самого высокого уровня объединяет все низкоуровневые модули и является главным [5];

– При разработке серверных и клиентских модулей используется метод нисходящего программирования. На начальном этапе разрабатывается общая структура программы – объявляются классы, структуры, типы данных, функции. Затем разрабатываются отдельные части программы – определяются методы классов, поля классов, функции. На каждом этапе разработки решается более простая задача [5].

UML-диаграмма развёртывания программно-аппаратного комплекса показана на рисунке 22.

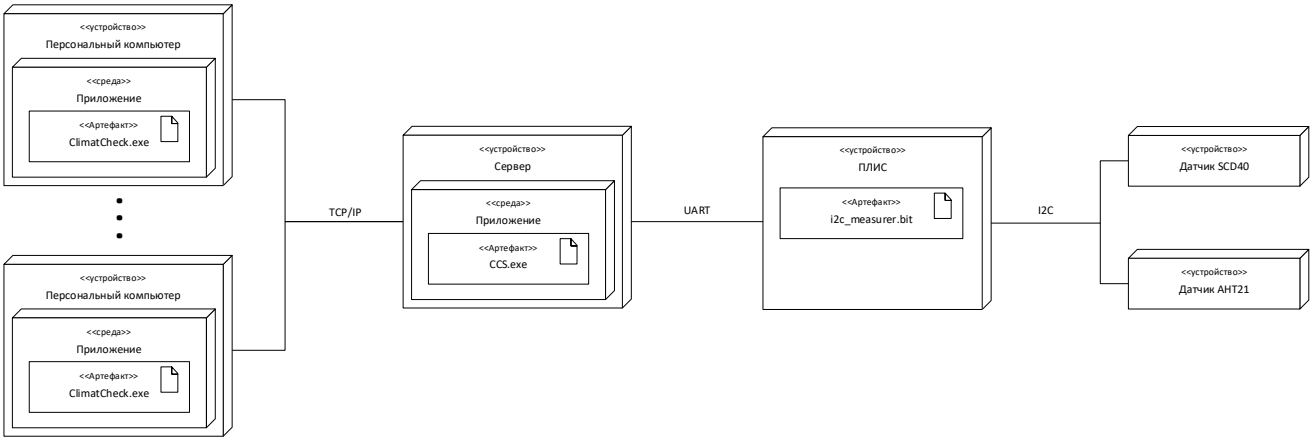


Рисунок 22 – UML-диаграмма развёртывания программно-аппаратного комплекса

В данной главе описан процесс разработки модулей программно-аппаратного комплекса, включая проектирование структуры и поведения модулей с помощью UML-диаграмм.

3.1 Разработка аппаратных модулей

Аппаратная часть комплекса включает в себя открытую аппаратную платформу ПЛИС и модули датчиков SCD40 и AHT21. Схема соединений, описывающая связи между аппаратными модулями, представлена на рисунке 23.

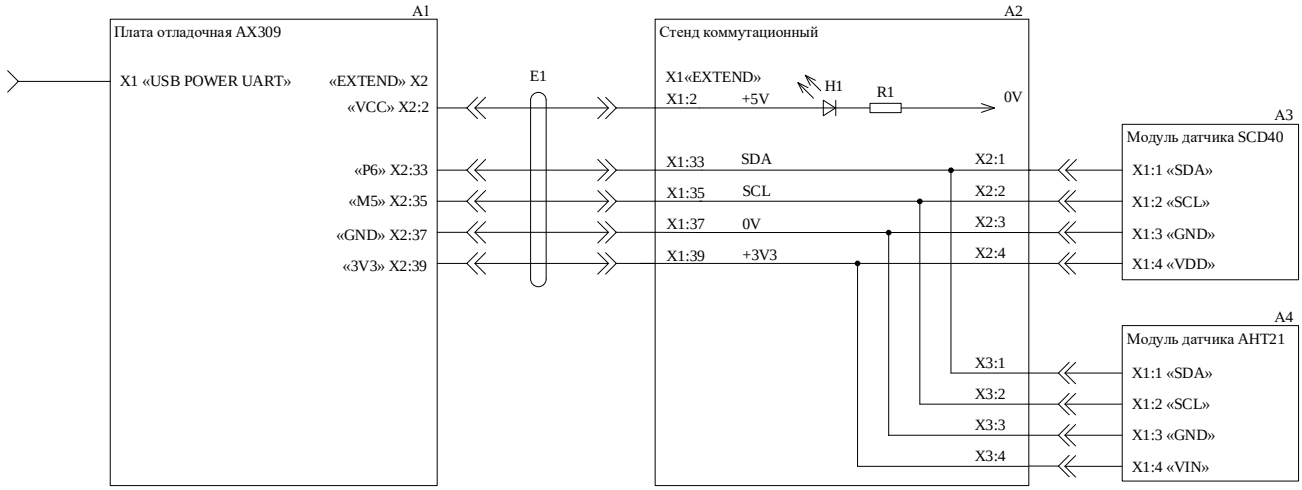


Рисунок 23 – Схема соединений аппаратной части комплекса

Коммутационный стенд реализует функцию соединения выводов модулей датчиков с выводами отладочной платы. На выводы питания датчиков AHT21 [6] и SCD40 [7] подаётся напряжение U_{33} = +3,3 В, что соответствует характеристикам, описанным в документации к данным датчикам. Для индикации питания используется красный светодиод 3AR4SD. В схему индикации питания также входит резистор R1, который ограничивает ток через светодиод H1. Сопротивление для резистора рассчитывается с учётом возможного влияния светодиода на измерения датчиками параметров микроклимата, в частности – температуры. Для расчёта сопротивления резистора представлена формула 1.

$$R = (U - U_{п}) / I_{пр}, \quad (1)$$

где U – напряжение источника питания;

$U_{п}$ – напряжение прямого падения;

$I_{пр}$ – протекающий ток.

Напряжение прямого падения $U_{п}$ определяется техническим описанием светодиода [8]. При протекающем через светодиод токе $I_{пр}$ = 20 мА достигается максимальная сила света [8]. Чтобы минимизировать тепловое влияние на датчики и снизить энергопотребление, достаточно $I_{пр}$ = 2 мА для индикации питания. Для вычисления сопротивления по формуле 1 ниже определены величины:

$$U = 5 \text{ В}; U_{п} = 2 \text{ В}; I_{пр} = 2 \text{ мА} = 0{,}002 \text{ А}.$$

После подстановки величин в формулу 1 ниже рассчитано сопротивление резистора:

$$R = (5 - 2) / 0{,}002 = 1500 \text{ (Ом)}.$$

В соответствии с расчётом, сопротивление резистора R1 равно 1500 Ом. Для коммутационного стенда используется SMD-резистор 1,5 кОм 152 1206.

Взаимодействие ПЛИС с датчиками осуществляется через двунаправленные линии интерфейса I2C – SCL и SDA. Связь с программной частью комплекса обеспечивается однонаправленными линиями протокола UART – RX и TX.

В рамках протокола I2C ПЛИС является ведущим, а датчики – ведомыми. Для координирования связи по линиям интерфейса I2C реализован модуль ведущего i2c_master. В его функционал входят инициирование сеанса обмена, генерирование тактового сигнала, адресация ведомого, чтение и запись данных, и завершение сеанса. Выводы модуля i2c_master представлены на рисунке 24.

```
port (
    -- Синхросигнал
    clk       : in     std_logic;
    -- Инвертированный сброс
    reset_n   : in     std_logic;

    -- Линия последовательных данных I2C
    sda       : inout  std_logic;
    -- Линия синхронизации I2C
    scl       : inout  std_logic;

    -- Флаг запуска передачи
    i2c_start : in     std_logic;
    -- Флаг процесса передачи
    i2c_busy  : out    std_logic;
    -- Флаг окончания передачи
    i2c_done  : out    std_logic;
    -- Флаг ошибки подтверждения
    ack_error : buffer std_logic;
    -- Флаг удержания
    i2c_stretch : in std_logic;

    -- Адрес ведомого
    address   : in     std_logic_vector(6 downto 0);
    -- Флаг типа операции: чтение/запись
    rw        : in     std_logic;
    -- Данные для записи
    data_wr   : in     std_logic_vector(7 downto 0);
    -- Полученные данные
    data_rd : out    std_logic_vector(7 downto 0)
);
```

Рисунок 24 – Выводы модуля i2c_master

Инициирование сеанса обмена предназначено для того, чтобы ведомые могли зафиксировать начало передачи данных. Эта функция предполагает отправку стартового сигнала, условие которого образуется при сбросе сигнала на линии SDA при единичном состоянии линии SCL [9]. Реализация функции инициирования сеанса обмена продемонстрирована на рисунке 25

```
-- Управление внутренним сигналом sda
with master_state select
                -- Старт сигнал
    sda_ena_n <= data_clk_prev when INIT_FIRST_BIT,
                -- Стоп сигнал
                not data_clk_prev when STOP,
                -- Обычное поведение внутреннего сигнала
                sda_int when others;
```

Рисунок 25 – Реализация функции инициирования сеанса обмена в модуле i2c_master

Генерирование тактового сигнала происходит, когда ведущий передаёт данные ведомым. В процессе передачи данных ведущий сбрасывает и удерживает сигнал на линии SCL, после передачи её состояние возвращается в высокое. Разработанная функция генерирования тактового сигнала представлена на рисунке 26.

```
-- Управление линиями I2C
--  Удерживать scl, когда происходит передача и внутренний синхросигнал scl 0, иначе не удерживать
scl <= '0' when (scl_ena = '1' and scl_clk = '0') else 'Z';
```

Рисунок 26 – Реализация функции генерирования тактового сигнала в модуле i2c_master

Адресация ведомого подразумевает отправку ведущим адреса ведомого и типа операции – запись или чтение. Реализация функции адресации продемонстрирована на рисунке 27.

```
-- Модуль отправляет заголовок (адрес ведомого + тип операции)
when SEND_HEADER =>
    -- Передний фронт внутреннего синхросигнала данных
    if data_clk = '1' and data_clk_prev = '0' then
        -- Если счетчик разрядов данных достиг 0, то передача данных выполнена
        if bit_cnt = 0 then
            -- Освободить sda для ведомого, чтобы он вернул подтверждение
            sda_int <= '1';
            -- Восстановить счетчик разрядов данных
            bit_cnt <= 7;
            -- Ожидать подтверждение от ведомого
            master_state <= SLV_ACK1;
        -- Если счетчик разрядов не равен 0, продолжать передачу
        else
            -- Уменьшить счетчик разрядов
            bit_cnt <= bit_cnt - 1;
            -- Записать следующий разряд
            sda_int <= i2c_header(bit_cnt-1);
            -- Продолжить цикл отправки
            master_state <= SEND_HEADER;
        end if;
    end if;
```

Рисунок 27 – Реализация функции инициирования сеанса обмена ведущим в модуле i2c_master

Отправка данных ведущим осуществляется путём передачи данных по линии SDA, причём после каждого переданного байта ожидается подтверждение от ведомого. Разработанная функция отправки данных показана на рисунке 28.

```
-- Модуль записывает данные
when WR =>
    -- Передний фронт внутреннего синхросигнала данных
    if data_clk = '1' and data_clk_prev = '0' then
        -- Установить флаг работы модуля, если передача продолжается
        i2c_busy <= '1';
        -- Если счетчик разрядов данных достиг 0, то запись данных выполнена
        if bit_cnt = 0 then
            -- Освободить sda для ведомого, чтобы он вернул подтверждение
            sda_int <= '1';
            -- Восстановить счетчик разрядов данных
            bit_cnt <= 7;
            -- Ожидать подтверждение от ведомого
            master_state <= SLV_ACK2;
        -- Если счетчик разрядов не равен 0, продолжать запись
        else
            -- Уменьшить счетчик разрядов
            bit_cnt <= bit_cnt - 1;
            -- Записать следующий разряд
            sda_int <= data_tx(bit_cnt-1);
            -- Продолжать цикл записи
            master_state <= WR;
        end if;
    end if;
```

Рисунок 28 – Реализация функции отправки данных в модуле i2c_master

Чтение данных ведущим подразумевает чтение сигналов с линии SDA, причём после каждого прочитанного байта отправляется подтверждение от ведущего. Реализация функции отправки данных показана на рисунке 29.

```
-- Передний фронт внутреннего синхросигнала данных
if data_clk = '1' and data_clk_prev = '0' then
    -- Установить флаг работы модуля, если передача продолжается
    i2c_busy <= '1';
    -- Если счетчик разрядов данных достиг 0, то чтение данных выполнено
    if bit_cnt = 0 then
        --  Кол-во байт передаваемой информации описано в документациях датчика
        -- Если требуется продолжать чтение, ведущий отправляет сигнал подтверждения ведомому
        --  то есть просто удерживает sda
        if (i2c_start = '1' and i2c_header = address & rw) then
            -- Удерживать sda
            sda_int <= '0';
        -- Если чтение требуется закончить, ведущий отправляет сигнал "отсутствие подтверждения"
        --  то есть не занимает sda
        else
            -- Не занимать sda
            sda_int <= '1';
        end if;
        -- Восстановить счетчик разрядов данных
        bit_cnt <= 7;
        -- Передать прочитанный байт информации на вывод модуля
        data_rd <= data_rx;
        -- Перейти к отправке подтверждения ведомому
        master_state <= MSTR_ACK;
    -- Если счетчик разрядов не равен 0, продолжать чтение
    else
        -- Уменьшить счетчик разрядов
        bit_cnt <= bit_cnt - 1;
        -- Продолжать цикл чтения
        master_state <= RD;
    end if;
-- Задний фронт внутреннего синхросигнала данных
elsif data_clk = '0' and data_clk_prev = '1' then
    -- Прочитать разряд с sda
    data_rx(bit_cnt) <= sda;
end if;
```

Рисунок 29 – Реализация функции чтения данных в модуле i2c_master

Завершение сеанса обмена предназначено для того, чтобы ведомые могли зафиксировать остановку передачи данных. Эта функция предполагает отправку стоп-сигнала, условие которого образуется при установке сигнала на линии SDA

при единичном состоянии линии SCL [9]. Разработанная функция завершения сеанса обмена продемонстрирована на рисунке 30.

```
-- Управление внутренним сигналом sda
with master_state select
                -- Старт сигнал
    sda_ena_n <= data_clk_prev when INIT_FIRST_BIT,
                -- Стоп сигнал
                not data_clk_prev when STOP,
                -- Обычное поведение внутреннего сигнала
                sda_int when others;
```

Рисунок 30 – Реализация функции завершения сеанса в модуле i2c_master

Для обращения к датчикам необходимо использовать закреплённые за каждыми датчиками адреса. Более того, каждый из их принимает и возвращает данные разной размерности, поэтому для работы с датчиками реализованы модули-драйверы SCD40Driver и AHT21Driver. Главным функционалом таких модулей является дешифрация команды и управление модулем i2c_master для работы с определённым датчиком.

SCD40Driver предназначен для работы с датчиком SCD40. Выводы модуля показаны на рисунке 31.

```
port (
    clk   : in STD_LOGIC;
    reset : in STD_LOGIC;

    i2c_done         : in  STD_LOGIC;
    i2c_busy         : in  STD_LOGIC;
    i2c_start        : out STD_LOGIC := '0';
    i2c_error        : in  STD_LOGIC;
    i2c_headSCD40    : out STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    i2c_rd           : in  STD_LOGIC_VECTOR(7 downto 0);
    i2c_wr           : out STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    i2c_stretchSCD40 : out STD_LOGIC := '0';

    startSCD40   : in  STD_LOGIC;
    doneSCD40    : out STD_LOGIC := '0';
    commandSCD40 : in STD_LOGIC_VECTOR(7 downto 0);
    resultSCD40  : out STD_LOGIC_VECTOR(9*8-1 downto 0) := (others => '0')
);
```

Рисунок 31 – Выводы модуля SCD40Driver

Команда, полученная модулем SCD40Driver, может быть дешифрована как:

– команда запуска периодических измерений;

– команда запроса результатов измерения;

– команда остановки периодических измерений;

– некорректная команда.

Реализация функции дешифрации и управления модулем i2c_master представлена на рисунке 32.

```
if startSCD40 = '1' then
    result_s <= (others => '0');
    command_s <= commandSCD40;
    counter <= 9*8-1;
    counter_timer <= 0;

    -- Команда запуска периодических измерений
    if commandSCD40(6 downto 5) = "01" then
        i2c_head_s <= address(6 downto 0) & '0';
        i2c_wr_s <= start_periodic_measurement(0);
        i2c_start <= '1';
        state <= COMMAND_STARTM0;
        result_s <= x"A1111111111111111";
    -- Команда запроса результатов измерения
    elsif commandSCD40(6 downto 5) = "10" then
        i2c_head_s <= address(6 downto 0) & '0';
        i2c_wr_s <= read_measurement(0);
        i2c_start <= '1';
        state <= COMMAND_READM0;
    -- Команда остановки периодических измерений
    elsif commandSCD40(6 downto 5) = "11" then
        i2c_head_s <= address(6 downto 0) & '0';
        i2c_wr_s <= stop_periodic_measurement(0);
        i2c_start <= '1';
        state <= COMMAND_STOPM0;
        result_s <= x"A3333333333333333";
    else
        result_s <= x"A0000000000000000";
        state <= STOP;
    end if;
end if;
```

Рисунок 32 – Реализация функции дешифрации и управления модулем i2c_master в модуле SCD40Driver

Модуль-драйвер AHT21Driver используется ПЛИС для взаимодействия с датчиком AHT21. Выводы модуля продемонстрированы на рисунке 33.

```
PORT (
    clk   : in STD_LOGIC;
    reset : in STD_LOGIC;

    i2c_done         : in  STD_LOGIC;
    i2c_busy         : in  STD_LOGIC;
    i2c_start        : out STD_LOGIC := '0';
    i2c_error        : in  STD_LOGIC;
    i2c_headAHT21    : out STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    i2c_rd           : in  STD_LOGIC_VECTOR(7 downto 0);
    i2c_wr           : out STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    i2c_stretchAHT21 : out STD_LOGIC := '0';

    startAHT21   : in  STD_LOGIC;
    doneAHT21    : out STD_LOGIC := '0';
    commandAHT21 : in STD_LOGIC_VECTOR(7 downto 0);
    resultAHT21  : out STD_LOGIC_VECTOR(7*8-1 downto 0) := (others => '0')
);
```

Рисунок 33 – Выводы модуля AHT21Driver

Полученная модулем AHT21 команда дешифрируется как:

– команда запроса результатов измерения;

– некорректная команда.

Разработанная функция дешифрации и управления модулем i2c_master представлена на рисунке 34.

```
if startAHT21 = '1' then
    result_s <= (others => '0');
    command_s <= commandAHT21;
    counter <= 7*8-1;
    counter_timer <= 0;

    -- Команда запроса результатов измерения
    if commandAHT21(6 downto 5) = "11" then
        i2c_head_s <= address(6 downto 0) & '0';
        i2c_wr_s <= trigger_measurement(0);
        i2c_start <= '1';
        state <= COMMAND_TRIGGERM0;
    else
        result_s <= x"B0000000000000";
        state <= STOP;
    end if;
end if;
```

Рисунок 34 – Реализация функции дешифрации и управления модулем i2c_master в модуле AHT21Driver

Для обеспечения взаимодействия между ПЛИС и компьютером через последовательный порт с использованием протокола UART реализованы модули rx и tx.

Модуль rx выполняет роль приёмника команд от компьютера. Модуль обращается по линии RX, считывает поток битов и записывает их в буфер для дальнейшей передачи драйверам. Выводы модуля rx продемонстрированы на рисунке 35.

```
port (
    --clock
    clk   : in STD_LOGIC;
    --sync reset
    reset : in STD_LOGIC;

    --receiver port
    rx_serial : in STD_LOGIC;
    --receiver buffer
    rx_data : out STD_LOGIC_VECTOR(7 downto 0);
    --receiver error flag
    rx_error  : out STD_LOGIC := '0';
    --receiver done flag
    rx_done   : out STD_LOGIC := '0'
);
```

Рисунок 35 – Выводы модуля rx

Когда линии данных не задействованы, они определены единичным состоянием. Сигналом, по которому начинается считывание данных, является

нулевое состояние линии RX. После байта информации может дополнительно передаваться бит четности [10]. Реализация функции считывания потока битов и записи данных в буфер показана на рисунке 36.

```
--when receive data
when READ_DATA =>
    --pick bit in the middle of baud
    if baud_counter = BAUD_DIVIDER - 1 then
        baud_counter <= 0;
        rx_data_s(bit_index) <= rx_serial;
        --last bit received
        if bit_index = 7 then
            bit_index <= 0;
            --if parity control - check parity bit
            if PARITY_FLAG then
                rx_state <= READ_PARITY;
            --if not parity control - stop reading
            else
                rx_state <= READ_STOP;
            end if;
        --increment bit index
        else
            bit_index <= bit_index + 1;
        end if;
    --increment baud counter
    else
        baud_counter <= baud_counter + 1;
    end if;
```

Рисунок 36 – Реализация функции считывания потока битов и записи данных в буфер в модуле rx

Модуль tx выполняет роль передатчика результатов выполнения команды. Модуль, используя линию TX, побитно передаёт данные компьютеру. Выводы модуля tx продемонстрированы на рисунке 37.

```
port (
    clk   : in STD_LOGIC;
    reset : in STD_LOGIC;

    tx_serial : out STD_LOGIC := '1';
    tx_data : in STD_LOGIC_VECTOR(7 downto 0);
    tx_start : in STD_LOGIC;
    tx_busy   : out STD_LOGIC := '0'
);
```

Рисунок 37 – Выводы модуля tx

Когда модулю поступает запрос на отправку данных, он переводит линию TX в нулевое состояние, сигнализируя о начале передачи данных, и начинает побитовую отправку данных. Так как размерность данных может различаться от типа команды, на компьютере заранее определено количество байт для получения, что исключает потерю информации. Если установлен контроль четности, модуль

дополнительно отправляет бит четности. Разработанная функция передачи данных компьютеру представлена на рисунке 38.

```
when WRITE_DATA =>
    if baud_counter = BAUD_DIVIDER - 2 then
        bit_index <= bit_index + 1;
        baud_counter <= baud_counter + 1;
    elsif baud_counter = BAUD_DIVIDER - 1 then
        baud_counter <= 0;
        if bit_index = 8 then
            bit_index <= 0;
            if PARITY_FLAG then
                if PARITY_EVEN then
                    tx_serial <= xor_reduce(tx_data_s);
                else
                    tx_serial <= not xor_reduce(tx_data_s);
                end if;
                tx_state <= WRITE_PARITY;
            else
                tx_serial <= '1';
                tx_state <= WRITE_STOP;
            end if;
        else
            tx_serial <= tx_data_s(bit_index);
        end if;
    else
        baud_counter <= baud_counter + 1;
    end if;
```

Рисунок 38 – Реализация функции передачи данных компьютеру в модуле tx

Разработанные модули обеспечивают выполнения всех функций аппаратной части, предусмотренных её архитектурой.

3.2 Разработка серверных модулей

Одним из компонентов программной части комплекса является сервер. Разработка серверных модулей предполагает реализацию следующих функций:

– работа с последовательным портом;

– работа с TCP-сокетом;

– поддержка форматов данных датчиков;

– реализация работы в режиме службы на ОС Windows;

– работа с событиями ОС Windows;

– работа с файловой системой.

UML-диаграмма классов, отражающая структуру сервера и связи между серверными модулями, показана на рисунке 39.

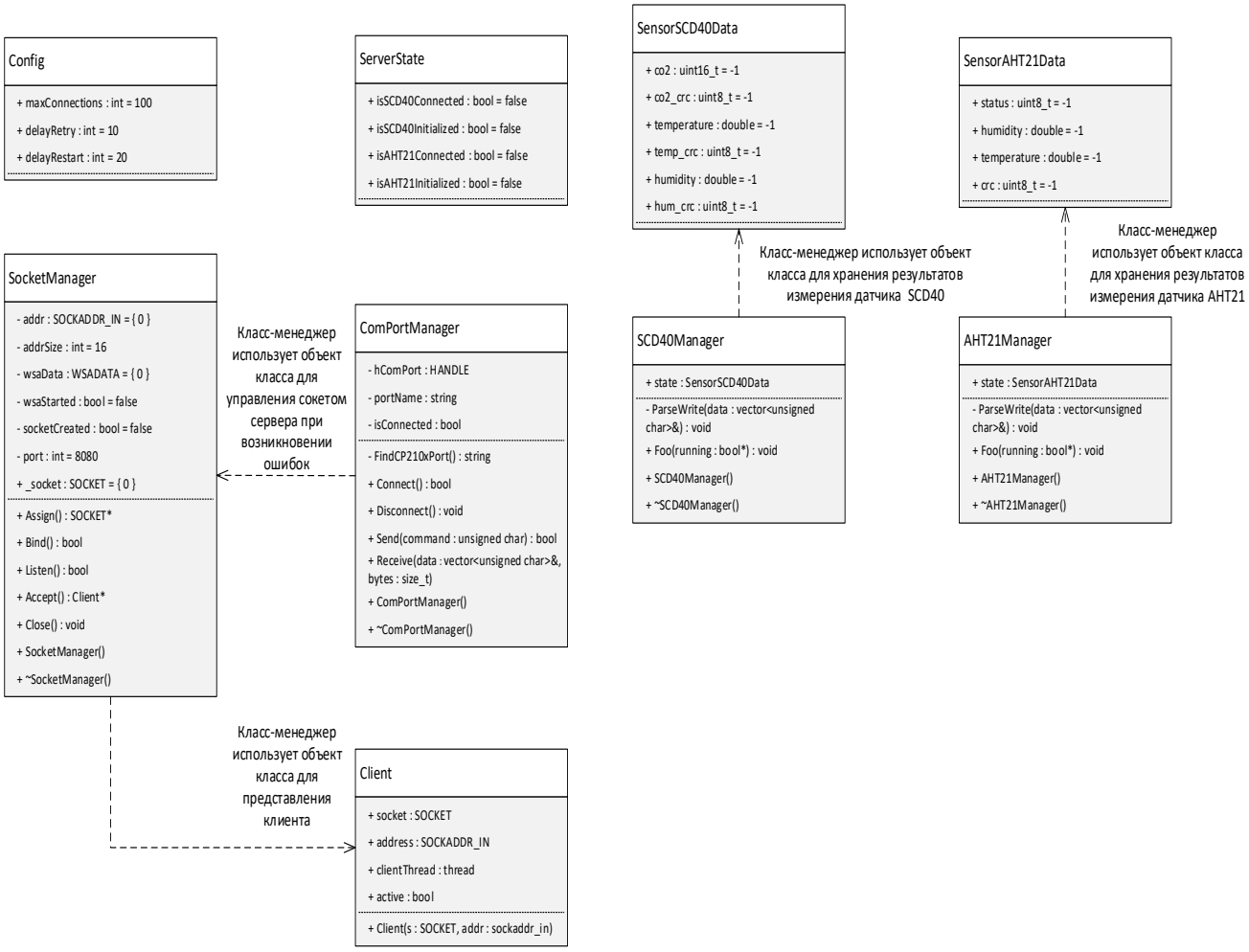


Рисунок 39 – UML-диаграмма классов сервера

Работа с последовательным портом необходима для взаимодействия с аппаратной частью: управление с помощью команд и получение результатов выполнения в виде массива байтов. Этот функционал реализован в классе ComPortManager.

Перед подключением последовательного порта необходимо удостовериться, что аппаратная часть комплекса подключена к компьютеру, на котором запущен сервер. Подключение определяется наличием такого устройства, как Silicon Labs CP210x, в системном реестре. Доступ к реестру предоставляет библиотека SetupAPI в ОС Windows. Реализация функции поиска последовательного порта представлена на рисунке 40.

```
// цикл просмотра всех устройств
for (i = 0; SetupDiEnumDeviceInfo(hDevInfo, i, &devInfoData); i++) {
    DWORD dataType;
    char buffer[256];
    DWORD bufferSize = sizeof(buffer);

    // если библиотека setupapi вернула устройство
    if (SetupDiGetDeviceRegistryPropertyA(hDevInfo, &devInfoData,
        SPDRP_DEVICEDESC, &dataType, (PBYTE)buffer, bufferSize, &bufferSize)) {
        std::string deviceDesc(buffer);

        // если название устройства совпадает с необходимым
        if (deviceDesc.find("CP210") != std::string::npos ||
            deviceDesc.find("Silicon Labs") != std::string::npos) {
            // открыть раздел реестра с конфигурацией этого устройства
            HKEY hKey = SetupDiOpenDevRegKey(hDevInfo, &devInfoData,
                DICS_FLAG_GLOBAL, 0, DIREG_DEV, KEY_READ);

            // если такой раздел есть, то устройство найдено и подключено
            if (hKey != INVALID_HANDLE_VALUE) {
                char portName[256];
                DWORD portNameSize = sizeof(portName);

                if (RegQueryValueExA(hKey, "PortName", NULL, NULL,
                    (LPBYTE)portName, &portNameSize) == ERROR_SUCCESS) {

                    RegCloseKey(hKey);
                    SetupDiDestroyDeviceInfoList(hDevInfo);
                    return std::string(portName);
                }
                RegCloseKey(hKey);
            }
        }
    }
}
```

Рисунок 40 – Реализация функции поиска последовательного порта в классе ComPortManager

Если аппаратная часть подключена, то выполняется попытка получить доступ к последовательному порту для дальнейшего чтения и записи. При успешной попытке в поле hComPort записывается дескриптор устройства. Для обеспечения корректной работы с аппаратной частью комплекса устанавливаются единые параметры последовательного порта:

– скорость передачи – 115200 бод;

– размер байта – 8 бит;

– размер стоп-бита – 1 бит;

– контроль четности – выключен.

Разработанная функция подключения последовательного порта продемонстрирована на рисунке 41.

```cpp
// путь к устройству
std::string fullPortName = "\\\\.\\" + portName;
// попытка получить доступ к последовательному порту
hComPort = CreateFileA(fullPortName.c_str(),
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);

// если не удалось получить доступ
if (hComPort == INVALID_HANDLE_VALUE) {
    Log("Ошибка открытия порта Silicon Labs CP210x", 2, 3);
    return false;
}

// попытка получить доступ к параметрам порта
DCB dcbSerialParams = { 0 };
dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
if (!GetCommState(hComPort, &dcbSerialParams)) {
    Log("Ошибка получения состояния порта Silicon Labs CP210x", 2, 3);
    CloseHandle(hComPort);
    hComPort = INVALID_HANDLE_VALUE;
    return false;
}

// установка параметров порта
dcbSerialParams.BaudRate = CBR_115200;
dcbSerialParams.ByteSize = 8;
dcbSerialParams.StopBits = ONESTOPBIT;
dcbSerialParams.Parity = NOPARITY;
```

Рисунок 41 – Реализация функции подключения последовательного порта в классе ComPortManager

Отправка команды для аппаратной части комплекса осуществляется с использованием дескриптора последовательного порта. Если попытка отправки завершается неудачей, то устройство последовательного порта отключается от сервера и происходит разрыв соединения с клиентами. Реализация функции отправки команды показана на рисунке 42.

```cpp
DWORD bytesWritten;
// отправка команды
bool sended = WriteFile(hComPort, &command, 1, &bytesWritten, NULL);
// если не удалось отправить данные - устройство порта было отключено
if (!sended) {
    isConnected = false;
    {
        std::lock_guard<std::mutex> lock(servstateMutex);

        Log("Потеряно соединение с портом Silicon Labs CP210x", 2, 3);

        // так как соединение с аппаратной частью потеряно, разрываем соединение с клиентами
        socketManager->Close();
    }
    return false;
}

// если отправлено некорректное количество байт - ошибка работы устройства порта
if (bytesWritten != 1) {
    isConnected = false;
    {
        std::lock_guard<std::mutex> lock(servstateMutex);

        Log("Потеряно соединение с портом Silicon Labs CP210x", 2, 3);

        // так как ошибка может быть критической, разрываем соединение с клиентами
        socketManager->Close();
    }
    return false;
}

return true;
```

Рисунок 42 – Реализация функции отправки команды в классе ComPortManager

Чтение результатов выполнения команды выполняется при обращении к дескриптору последовательного порта. Если попытка чтения завершается неудачей, то устройство последовательного порта отключается от сервера и происходит разрыв соединения с клиентами. Разработанная функция чтения результатов выполнения команды представлена на рисунке 43.

```
DWORD bytesRead = 0;
DWORD totalBytesRead = 0;

while (totalBytesRead < bytes) {
    // Если не удалось прочитать байт информации - устройство порта было отключено
    if (!ReadFile(hComPort, &data[totalBytesRead], bytes - totalBytesRead, &bytesRead, NULL)) {
        isConnected = false;
        {
            std::lock_guard<std::mutex> lock(servstateMutex);

            Log("Потеряно соединение с портом Silicon Labs CP210x", 2, 3);

            // так как соединение с аппаратной частью потеряно, разрываем соединение с клиентами
            socketManager->Close();
        }
        return false;
    }
    // Таймаут ожидания чтения - ошибка работы устройства порта
    if (bytesRead == 0) {
        data.resize(totalBytesRead);
        isConnected = false;
        {
            std::lock_guard<std::mutex> lock(servstateMutex);

            Log("Потеряно соединение с портом Silicon Labs CP210x", 2, 3);

            // так как ошибка может быть критической, разрываем соединение с клиентами
            socketManager->Close();
        }
        return false;
    }

    totalBytesRead += bytesRead;
}
```

Рисунок 43 – Реализация функции чтения результатов выполнения команды в классе ComPortManager

Работа с TCP-сокетом подразумевает взаимодействие с клиентами: принятие подключения и обработка запросов.

Функция принятия подключения реализована в классе SocketManager. Если клиент подключился к серверу, его дескриптор сокета и адрес заносятся в объект структуры Client. Если сервер разрывает соединение или произошла ошибка подключения, созданный объект удаляется. Реализация функции принятия подключения продемонстрирована на рисунке 44.

```
SOCKADDR_IN* clientAddr = new SOCKADDR_IN();
SOCKET clientSocket = accept(_socket, (struct sockaddr*)clientAddr, &addrSize);
Client* client = new Client(clientSocket, *clientAddr);

if (!client->socket) {
    delete clientAddr;
    delete client;

    return NULL;
}

return client;
```

Рисунок 44 – Реализация функции принятия подключения в классе SocketManager

Обработка запросов клиентов реализована функциями ReceiveRequests и ClientFoo. Первая функция принимает новые соединения и в зависимости от запроса возвращает определённый ответ:

– если клиент запрашивает текущее состояние датчиков, то сервер выполняет запрос;

– если клиент готовится периодически запрашивать результаты измерений датчиков, сервер создаёт новый поток, в котором выполняется функция ClientFoo.

Разработанная функция обработки запросов клиентов ReceiveRequest показана на рисунке 45.

```
Client* currentClient = socketManager->Accept();
// если сокет сервера был закрыт, прекратить принимать новые подключения
if (currentClient == NULL) { ... }
if (NULL == recv(currentClient->socket, authBuffer, 2, 0)) { ... }
if (authBuffer[0] == NULL && authBuffer[1] == NULL) { ... }
// запрос состояния датчиков
if (authBuffer[1] == '0') {
    send(currentClient->socket, authBuffer, 2, 0);
    delete& (currentClient->address);
    delete currentClient;
}
// периодические запросы измерения
else if (authBuffer[1] == '1') {
    send(currentClient->socket, authBuffer, 2, 0);
    send(currentClient->socket, (char*)&serverState, sizeof(ServerState), 0);
    {
        std::lock_guard<std::mutex> lock(clientsMutex);

        currentClient->clientThread = std::thread(ClientFoo, currentClient->socket, currentClient->address, _gServerRunning);
        clients.push_back(currentClient);
    }
}
```

Рисунок 45 – Реализация функции обработки запросов клиентов ReceiveRequest

Поток, выполняющий функцию ClientFoo, ожидает от клиента периодических запросов результатов измерений датчиков. На каждый запрос сервер делает копию измеренных параметров и последовательно отправляет состояние датчиков, результаты измерения датчиков SCD40 и AHT21. При возникновении ошибки соединение с клиентом закрывается. Реализация функции обработки запросов клиентов ClientFoo представлена на рисунке 46.

```
bytesReceived = recv(clientSocket, (char*)&request, sizeof(request), 0);
if (bytesReceived == SOCKET_ERROR || bytesReceived == 0)
    break;

// запрос на получение результатов измерений
if (request == 0x01) {
    ServerState serverStateCopy;
    SensorAHT21Data AHT21DataCopy;
    SensorSCD40Data SCD40DataCopy;

    // копирование объектов структур
    {
        std::lock_guard<std::mutex> lock(servstateMutex);
        serverStateCopy = serverState;
    }
    {
        std::lock_guard<std::mutex> lock(sensor1Mutex);
        SCD40DataCopy = sensor1State;
    }
    {
        std::lock_guard<std::mutex> lock(sensor2Mutex);
        AHT21DataCopy = sensor2State;
    }

    // последовательная отправка данных
    int bytesSent = send(clientSocket, (char*)&serverState, sizeof(ServerState), 0);
    if (bytesSent == SOCKET_ERROR)
        break;

    bytesSent = send(clientSocket, (char*)&SCD40DataCopy, sizeof(SensorSCD40Data), 0);
    if (bytesSent == SOCKET_ERROR)
        break;

    bytesSent = send(clientSocket, (char*)&AHT21DataCopy, sizeof(SensorAHT21Data), 0);
    if (bytesSent == SOCKET_ERROR)
        break;
```

Рисунок 46 – Реализация функции обработки запросов клиентов ClientFoo

Поддержка форматов данных датчиков применяется для корректного формирования команд и преобразования результатов измерения. Каждый датчик измеряет определённые параметры микроклимата и, соответственно, предоставляет свой формат данных.

Получение, преобразование и хранение данных, полученных с датчика SCD40, реализовано в классе SCD40Manager. Работа с датчиком выполняется в методе Foo(). На первом этапе выполняется запуск периодических измерений посылкой команды старта 0x20. Разработанная функция запуска измерений продемонстрирована на рисунке 47.

```
    for (int i = 0; i < 10; i++) {
        Sleep(100);
        if (!*running)
            return;
    }

    {
        std::lock_guard<std::mutex> lock(i2cMutex);

        if (!comPortManager.Send(cmdStart))
            return;

        if (!comPortManager.Receive(response, 9))
            continue;
    }

    if (response[0] != 0xA1 || response[1] != 0x11) {
        continue;
    }
```

Рисунок 47 – Реализация функции запуска датчика SCD40 в классе SCD40Manager

После успешного запуска периодических измерений в цикле каждые 5 секунд запрашивается результат измерения посылкой команды 0x40. Реализация функции запроса измерений показана на рисунке 48.

```
for (int i = 0; i < 50; i++) {  // 50 * 100 = 5000 мс
    Sleep(100);
    if (!*running)
        return;
}

{
    std::lock_guard<std::mutex> lock(i2cMutex);

    if (!comPortManager.Send(cmdGet))
        return;
    //continue;

    if (!comPortManager.Receive(response, 9)) {
        serverState.isSCD40Connected = false;
        serverState.isSCD40Initialized = false;
        break;
    }
}

if (response[0] == 0xFF && response[1] == 0xFF) {
    serverState.isSCD40Connected = false;
    serverState.isSCD40Initialized = false;
    break;
}
serverState.isSCD40Connected = true;
serverState.isSCD40Initialized = true;

{
    std::lock_guard<std::mutex> lock(sensor1Mutex);

    ParseWrite(response);
}
```

Рисунок 48 – Реализация функции запроса измерений датчика SCD40 в классе SCD40Manager

Преобразование и сохранение данных выполняется после успешного получения результата измерения с помощью метода ParseWrite(). Разработанная функция преобразования и сохранения данных представлена на рисунке 49.

```
sensor1State.co2 = (data[0] << 8) | data[1];
sensor1State.co2_crc = data[2];

// Температура: байты 3-4 (word, big-endian)
uint16_t temp_raw = (data[3] << 8) | data[4];
sensor1State.temperature = -45.0 + 175.0 * temp_raw / 65535.0;
sensor1State.temp_crc = data[5];

// Влажность: байты 6-7 (word, big-endian)
uint16_t hum_raw = (data[6] << 8) | data[7];
sensor1State.humidity = 100.0 * hum_raw / 65535.0;
sensor1State.hum_crc = data[8];
```

Рисунок 49 – Реализация функции преобразования и сохранения данных датчика SCD40 в классе SCD40Manager

Получение, преобразование и хранение данных, полученных с датчика AHT21, реализовано в классе AHT21Manager. Работа с датчиком выполняется в методе Foo(). В цикле каждые 100 миллисекунд запрашивается результат измерения посылкой команды 0xA0. Реализация функции запроса измерений продемонстрирована на рисунке 50.

```
Sleep(100);
if (!*running)
    return;

{
    std::lock_guard<std::mutex> lock(i2cMutex);

    if (!comPortManager.Send(cmdGet)) {
        *running = false;
        return;
    }

    if (!comPortManager.Receive(response, 7)) {
        serverState.isAHT21Connected = false;
        serverState.isAHT21Initialized = false;
        continue;
    }

    if (response[0] == 0xFF || response[1] == 0xFF) {
        serverState.isAHT21Connected = false;
        serverState.isAHT21Initialized = false;
        continue;
    }

    serverState.isAHT21Connected = true;
    serverState.isAHT21Initialized = true;
}

{
    std::lock_guard<std::mutex> lock(sensor2Mutex);

    ParseWrite(response);

    //printSensor2Data(state);
}
```

Рисунок 50 – Реализация функции запроса измерений датчика SCD40 в классе SCD40Manager

Реализация работы в режиме службы на операционной системе Windows предназначена для автоматического запуска сервера после загрузки операционной системы. Инициализация службы включает в себя создание таблицы службы и запуск обработчика событий. Разработанная функция инициализации службы показана на рисунке 51.

```
SERVICE_TABLE_ENTRYA serviceTable[] = {
    {NAME, (LPSERVICE_MAIN_FUNCTIONA)ServiceMain},
    {NULL, NULL}
};

if (!StartServiceCtrlDispatcherA(serviceTable)) {
    int index = GetLastError();
    Log("Не удалось запустить обработчик службы!", index, 3);
    return index;
}
return 0;
```

Рисунок 51 – Реализация функции инициализации службы

Запуск службы выполняется в функции StartService(). Сначала регистрируется функция для обработки запросов управления службой, затем создаётся событие остановки службы. В случае успеха служба устанавливается в рабочее состояние. Реализация функции запуска службы представлена на рисунке 52.

```
gStatusHandle = RegisterServiceCtrlHandlerA(_NAME, ServiceCtrlHandler);
if (!gStatusHandle) {
    Log("Ошибка запуска службы!", GetLastError(), 3);
    return false;
}

gServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
gServiceStatus.dwCurrentState = SERVICE_START_PENDING;

SetServiceStatus(gStatusHandle, &gServiceStatus);
Log("Служба запускается...", 1, 0);

gServiceStopEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
if (!gServiceStopEvent) {
    gServiceStatus.dwCurrentState = SERVICE_STOPPED;
    gServiceStatus.dwWin32ExitCode = GetLastError();
    SetServiceStatus(gStatusHandle, &gServiceStatus);
    Log("Ошибка запуска службы!", GetLastError(), 3);
    return false;
}

gServiceStatus.dwCurrentState = SERVICE_RUNNING;
gServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
gServiceStatus.dwWin32ExitCode = 0;
gServiceStatus.dwCheckPoint = 0;
SetServiceStatus(gStatusHandle, &gServiceStatus);
*_gServiceRunning = true;
Log("Служба запущена успешно", 1, 1);

return true;
```

Рисунок 52 – Реализация функции запуска службы

Остановка службы может быть выполнена, когда пользователь самостоятельно завершает службу. В этом случае вызывается событие, при котором выполняется функция StopServiceByEvent(). Разработанная функция остановки службы продемонстрирована на рисунке 53.

```
if (gServiceStatus.dwCurrentState != SERVICE_RUNNING) {
    Log("Получен сигнал остановки службы, но служба уже остановлена!", 1, 2);
    *_gServiceRunning = false;
    return;
}

*_gServiceRunning = false;
//Log("Служба остановлена успешно", 1, 1);
gServiceStatus.dwControlsAccepted = 0;
gServiceStatus.dwCurrentState = SERVICE_STOP_PENDING;
gServiceStatus.dwWin32ExitCode = 0;
gServiceStatus.dwCheckPoint = 4;

SetServiceStatus(gStatusHandle, &gServiceStatus);
SetEvent(gServiceStopEvent);
return;
```

Рисунок 53 – Реализация функции остановки службы

Работа с событиями операционной системы Windows выполняется для логирования ошибок и сведений в процессе работы сервера. Функцией взаимодействия с утилитой управления событиями EventCreate является Log(), реализация которой показана на рисунке 54.

```
void Log(const char* message, int id, int type) {
    char buffer[1024] = { 0 };
    sprintf(buffer, "EventCreate /T %s /ID %d /L APPLICATION /SO %s /D \"%s\"",
        !type ? "INFORMATION" : type == 1 ? "SUCCESS" : type == 2 ? "WARNING" : "ERROR",
        id, _NAME, message);

    system(buffer);
}
```

Рисунок 54 – Реализация функции работы с событиями операционной системы Windows

Работа с файловой системой предназначена для создания файла конфигурации и пакетных файлов, необходимых для запуска и корректной работы программы в виде службы Windows. Функцией работы с файлами является Initialisation(), реализация которой представлена на рисунке 55.

Рисунок 55 – Реализация функции работы с файловой системой

Разработанные модули обеспечивают выполнения всех функций сервера, предусмотренных архитектурой программной части.

3.3 Разработка клиентских модулей

Вторым компонентом программной части комплекса является клиент. Процесс разработки клиентских модулей включает реализацию следующих функций:

– предоставление пользователю графического интерфейса;

– работа с TCP-сокетом;

– преобразование потока двоичных данных в формат структуры;

– хранение и привязка параметров микроклимата к элементам графического интерфейса;

– сопоставление параметров микроклимата с требованиями ГОСТ [1];

– сохранение результатов процесса измерения в виде текстового файла;

– настройка IP-адреса сервера и периодичности обновления.

UML-диаграмма классов, отражающая структуру клиента и связи между клиентскими модулями, показана на рисунке 56.

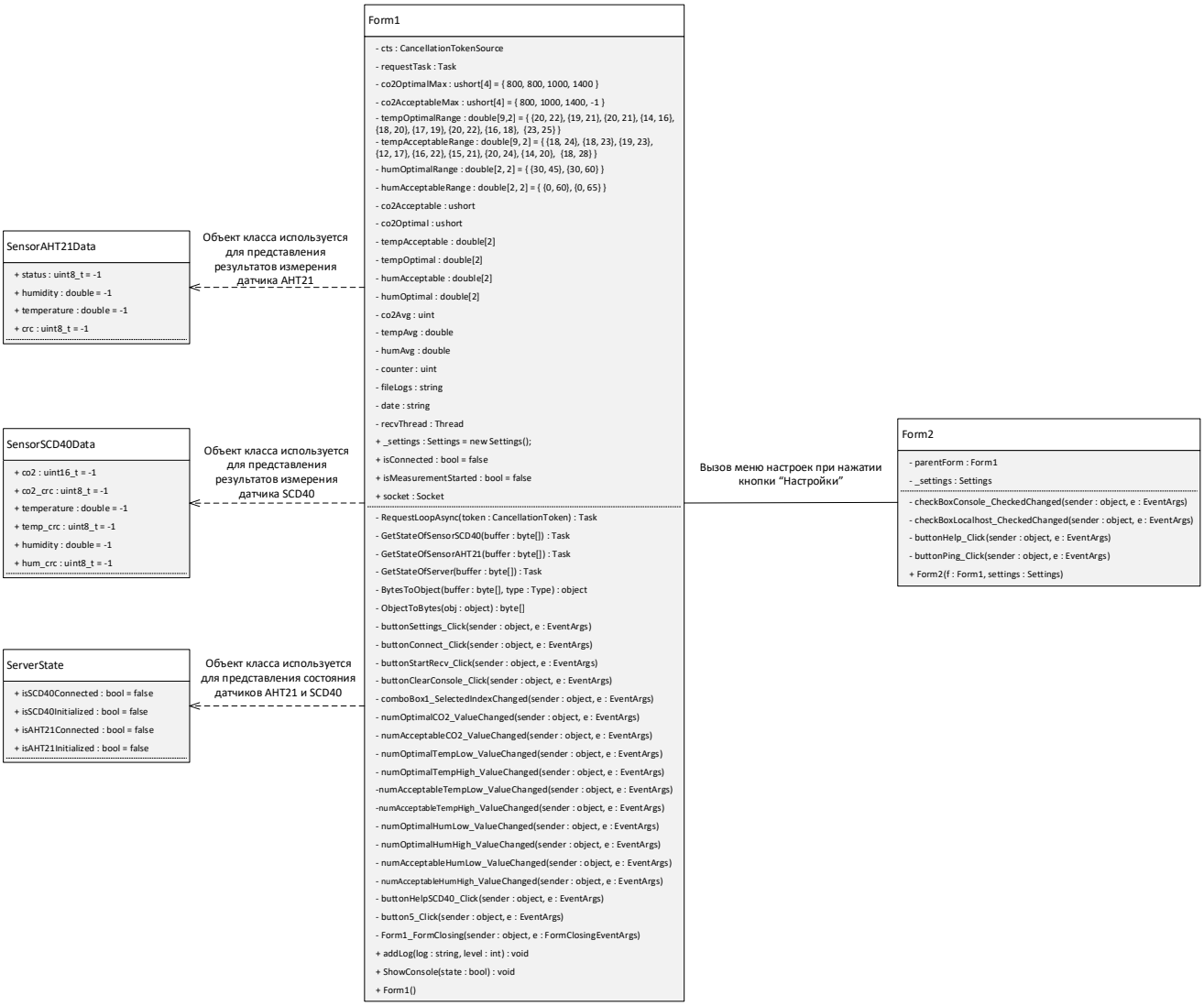


Рисунок 56 – UML-диаграмма классов клиента

Предоставление пользователю графического интерфейса позволяет взаимодействовать с программой с помощью мыши и клавиатуры, отображать параметры микроклимата и сигнализировать о несоответствии заданному диапазону в удобной для пользователя форме. Для разработки графического интерфейса используется платформа программирования графических приложений Windows Forms, обеспечивающая удобное размещение элементов графического интерфейса и определение выполняемых функций при определённом событии.

Главное окно Form1 предоставляет основной функционал клиента: управление работой программы, отображение параметров микроклимата, настройка условия измерений, отображение состояний датчиков, вызов окна настроек и открытие справочной информации о датчиках. Разработанный интерфейс главного окна продемонстрирован на рисунке 57.

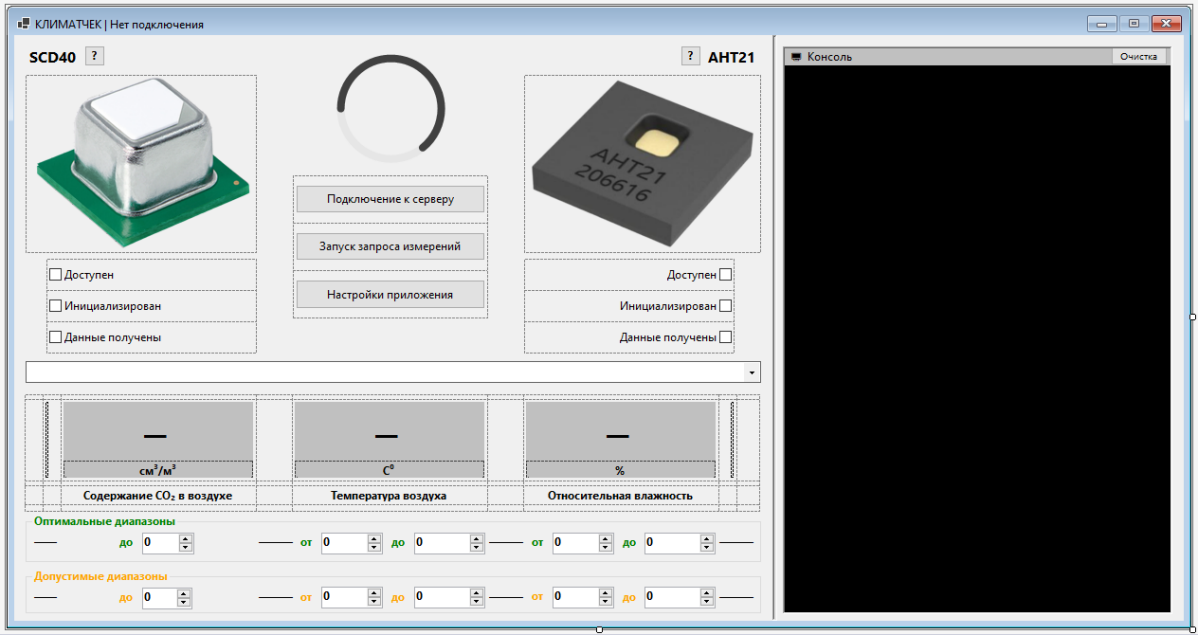


Рисунок 57 – Разработанный интерфейс главного окна

Окно настроек Form2 предназначено для настройки подключения: IP-адреса сервера и периодичности обновления. Окно может быть открыто в любой момент, однако в зависимости от текущей операции некоторые пункты окна могут быть заблокированы для редактирования. Реализованный интерфейс окна настроек представлен на рисунке 58.

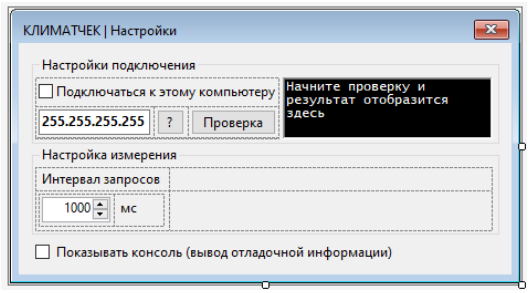


Рисунок 58 – Разработанный интерфейс окна настроек

Работа с TCP-сокетом подразумевает взаимодействие с сервером: подключение к серверу, получение состояний датчиков и периодические запросы результатов измерений.

Подключение к серверу выполняется при нажатии на кнопку «Подключение к серверу». При этом вызывается метод buttonConnect_Click() класса Form1. При успешном соединении отправляется проверочный байт, представляющий собой случайное число. Если ответ от сервера совпадает с отправленным байтом, то указанный сервер корректен. Разработанная функция подключения к серверу показана на рисунке 59.

```
Random random = new Random();
byte ack = (byte)random.Next();
byte[] ackBuffer = { ack, (byte)'1' };
socket = new Socket(SocketType.Stream, ProtocolType.Tcp);

socket.Connect(_settings.UseLocalhost ? "127.0.0.1" : _settings.ManualAddress, 8080);

addLog("Успешное подключение, отправка проверочного байта (" + ack.ToString() + ")", 3);
_ = socket.Send(ackBuffer);
_ = socket.Receive(ackBuffer);

if (ackBuffer[0] == ack)
    addLog("Ответ от сервера (" + ackBuffer[0].ToString() + ") совпадает с проверочным байтом (" +
else
{
    addLog("Ответ от сервера (" + ackBuffer[0].ToString() + ") не совпадает с проверочным байтом ("
```

Рисунок 59 – Реализация функции подключения к серверу

Одноразовое получение состояний датчиков выполняется после успешной попытки подключения к серверу в методе buttonConnect_Click() класса Form1. При этом выполняется проверка полученной информации, и если ответ сервера корректен, состояние датчиков выставляется успешно. Реализация функции получения состояний датчиков продемонстрирована на рисунке 60.

```
await Task.Run(() => GetStateOfServer(buffer), token);

state = (ServerState?)BytesToObject(buffer, typeof(ServerState));
if (state == null)
    break;
checkBoxSCD40Connected.Checked = state.Value.isSCD40Connected;
checkBoxAHT21Connected.Checked = state.Value.isAHT21Connected;
checkBoxSCD40Initialized.Checked = state.Value.isSCD40Initialized;
checkBoxAHT21Initialized.Checked = state.Value.isAHT21Initialized;
```

Рисунок 60 – Реализация функции получения состояний датчиков

Периодические запросы результатов измерений запускаются в отдельном потоке при нажатии на кнопку «Запуск запроса измерений». Функцией, отвечающей за запросы параметров, является метод RequestLoopAsync() класса Form1. Последовательно запрашивается состояния датчиков, результаты измерения датчиков SCD40 и AHT21. Разработанная функция периодических запросов результатов измерений представлена на рисунке 61.

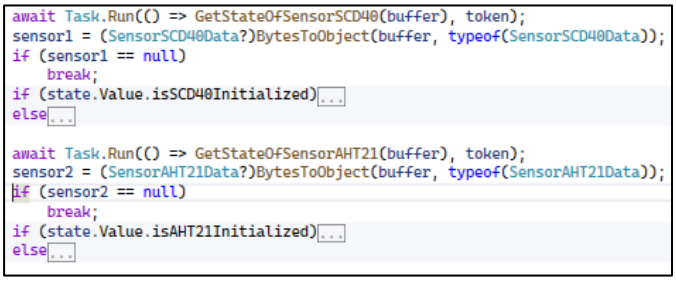


Рисунок 61 – Реализация функции периодических запросов результатов измерений

Преобразование потока двоичных данных в формат структуры необходимо, поскольку данные передаются между процессами, написанными на различных языках программирования. Сервер написан на языке C++, который является неуправляемым кодом, а клиент на C# – управляемом коде. Выполнение преобразований осуществляется с использованием библиотеки Marshal. Реализация функции преобразования потока двоичных данных в формат структуры показана на рисунке 62.

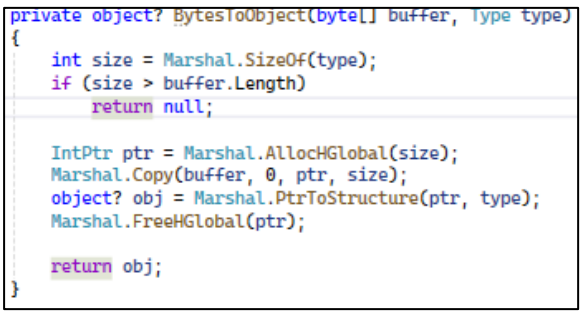


Рисунок 62 – Реализация функции преобразования потока двоичных данных в формат структуры

Хранение и привязка параметров микроклимата к элементам графического интерфейса выполняется в процессе периодических измерений и после остановки периодических измерений.

В процессе периодических измерений элементы графического интерфейса привязываются к параметрам микроклимата, полученным от сервера. Если датчик недоступен или произошла ошибка получения данных, то соответствующие параметры не будут отображаться. Разработанная функция хранения и привязки параметров микроклимата к элементам графического интерфейса в процессе периодических измерений продемонстрирована на рисунке 63.

```
        if (state.Value.isSCD40Initialized)
        {
            co2 = sensor1.Value.co2;
            co2Avg += co2;

            infoCO2.Text = co2.ToString();
            int index = comboBox1.SelectedIndex;
            if (index == 3 || index == 12)
                tableCO2.BackColor = co2 <= co2Optimal ? Color.Green : Color.Red;
            else if (index == 0 || index == 9 || index == 10 || index == 18 || index == 19)
                tableCO2.BackColor = Color.Silver;
            else
                if (co2 <= co2Optimal)
                tableCO2.BackColor = Color.Green;
            else tableCO2.BackColor = co2 <= co2Acceptable ? Color.Orange : Color.Red;

            measurementThis += "CO2 = " + co2.ToString() + " см3/м3 ";

            checkBoxSCD40Recv.Checked = true;
            counterSensor1++;
        }
        else
        {
            measurementThis += "CO2 = - см3/м3 ";

            infoCO2.Text = "-";
            tableCO2.BackColor = Color.Silver;
        }
if (state.Value.isAHT21Initialized)
{
    temp = sensor2.Value.temperature;
    hum = sensor2.Value.humidity;
    tempAvg += temp;
    humAvg += hum;

    infoTemp.Text = string.Format("{0:F1}", temp);
    int index = comboBox1.SelectedIndex;
    if (index == 0)
        tableTemp.BackColor = Color.Silver;
    else
    {
        if (temp >= tempOptimal[0] && temp <= tempOptimal[1])
            tableTemp.BackColor = Color.Green;
        else tableTemp.BackColor = temp >= tempAcceptable[0] && temp <= tempAcceptable[1] ? Color.Orange : Color.Red;
    }

    measurementThis += "T = " + string.Format("{0:F1}", temp) + " °C ";

    infoHum.Text = string.Format("{0:F1}", hum);
    index = comboBox1.SelectedIndex;
    if (index == 0 || index == 10)
        tableHum.BackColor = Color.Silver;
    else
    {
        if (hum >= humOptimal[0] && hum <= humOptimal[1])
            tableHum.BackColor = Color.Green;
        else tableHum.BackColor = hum >= humAcceptable[0] && hum <= humAcceptable[1] ? Color.Orange : Color.Red;
    }

    measurementThis += "RH = " + string.Format("{0:F1}", hum) + " % \n";

    checkBoxAHT21Recv.Checked = true;
    counterSensor2++;
}
else
{
    measurementThis += "T = - °C ";
    measurementThis += "RH = - % \n";

    infoTemp.Text = "-";
    tableTemp.BackColor = Color.Silver;
    infoHum.Text = "-";
    tableHum.BackColor = Color.Silver;
}
```

Рисунок 63 – Реализация функции хранения и привязки параметров микроклимата к элементам графического интерфейса в процессе периодических измерений

После остановки периодических измерений к элементам графического интерфейса привязываются средние значения измерений, полученных от сервера. Если во время запросов датчик был недоступен, то соответствующие параметры не будут отображены. Реализация функции хранения и привязки параметров микроклимата к элементам графического интерфейса после остановки периодических измерений представлена на рисунке 64.

```
            if (co2Avg <= co2Optimal)
            {
                tableCO2.BackColor = Color.Green;
                header1 += " – оптимально (CO2 <= " + co2Optimal.ToString() + ")\n";
            }
            else if (co2Avg <= co2Acceptable)
            {
                tableCO2.BackColor = Color.Orange;
                header1 += " – допустимо (CO2 <= " + co2Acceptable.ToString() + ")\n";
            }
            else
            {
                tableCO2.BackColor = Color.Red;
                header1 += " – ВНЕ ДИАПАЗОНА! (CO2 > " + co2Acceptable.ToString() + ")\n";
            }
infoTemp.Text = string.Format("{0:F1}", tempAvg);
header1 += " T = " + infoTemp.Text + " °C ";
if (tempAvg >= tempOptimal[0] && tempAvg <= tempOptimal[1])
{
    tableTemp.BackColor = Color.Green;
    header1 += " – оптимально (" + tempOptimal[0] + " <= T <= " + tempOptimal[1] + ")\n";
}
else if (tempAvg >= tempAcceptable[0] && tempAvg <= tempAcceptable[1])
{
    tableTemp.BackColor = Color.Orange;
    header1 += " – допустимо (" + tempAcceptable[0] + " <= T <= " + tempAcceptable[1] + ")\n";
}
else
{
    tableTemp.BackColor = Color.Red;
    header1 += " – ВНЕ ДИАПАЗОНА (T > " + tempAcceptable[1] + " или T < " + tempAcceptable[0] + ")\n";
}
/* if (tempAvg < tempAcceptable[0] || tempAvg > tempAcceptable[1]) ...
```

Рисунок 64 – Реализация функции хранения и привязки параметров микроклимата к элементам графического интерфейса после остановки периодических измерений

Сопоставление параметров микроклимата с требованиями ГОСТ выполняется в процессе периодических измерений и после остановки периодических измерений.

В процессе периодических измерений пользователь может изменять условие, поэтому проверка на соответствие нормам ГОСТ проводится каждый раз после получения данных от сервера. Учитываются не только диапазоны параметров, но и нормативность конкретных параметров в определённых условиях. Разработанная функция сопоставления параметров микроклимата с требованиями ГОСТ в процессе периодических измерений показана на рисунке 65.

```
            if (co2 <= co2Optimal)
                tableCO2.BackColor = Color.Green;
            else tableCO2.BackColor = co2 <= co2Acceptable ? Color.Orange : Color.Red;
if (hum >= humOptimal[0] && hum <= humOptimal[1])
    tableHum.BackColor = Color.Green;
else tableHum.BackColor = hum >= humAcceptable[0] && hum <= humAcceptable[1] ? Color.Orange : Color.Red;

if (temp >= tempOptimal[0] && temp <= tempOptimal[1])
    tableTemp.BackColor = Color.Green;
else tableTemp.BackColor = temp >= tempAcceptable[0] && temp <= tempAcceptable[1] ? Color.Orange : Color.Red;
```

Рисунок 65 – Реализация функции сопоставления параметров микроклимата с требованиями ГОСТ в процессе периодических измерений

После остановки периодических измерений используется условие, установленное перед остановкой. В этом случае проверяются средние значения параметров на соответствие нормам ГОСТ. Как и в прошлом случае, учитывается также наличие нормирования конкретных параметров в установленных условиях. Результаты сопоставления используются не только для индикации элементов графического интерфейса, но и для формирования файла с результатами измерений. Реализация функции сопоставления параметров микроклимата с требованиями ГОСТ после остановки периодических измерений продемонстрирована на рисунке 66.

```
  if (co2Avg <= co2Optimal)
  {
      tableCO2.BackColor = Color.Green;
      header1 += " – оптимально (CO2 <= " + co2Optimal.ToString() + ")\n";
  }
  else if (co2Avg <= co2Acceptable)
  {
      tableCO2.BackColor = Color.Orange;
      header1 += " – допустимо (CO2 <= " + co2Acceptable.ToString() + ")\n";
  }
  else
  {
      tableCO2.BackColor = Color.Red;
      header1 += " – ВНЕ ДИАПАЗОНА! (CO2 > " + co2Acceptable.ToString() + ")\n";
  }
if (tempAvg >= tempOptimal[0] && tempAvg <= tempOptimal[1])
{
    tableTemp.BackColor = Color.Green;
    header1 += " – оптимально (" + tempOptimal[0] + " <= T <= " + tempOptimal[1] + ")\n";
}
else if (tempAvg >= tempAcceptable[0] && tempAvg <= tempAcceptable[1])
{
    tableTemp.BackColor = Color.Orange;
    header1 += " – допустимо (" + tempAcceptable[0] + " <= T <= " + tempAcceptable[1] + ")\n";
}
else
{
    tableTemp.BackColor = Color.Red;
    header1 += " – ВНЕ ДИАПАЗОНА (T > " + tempAcceptable[1] + " или T < " + tempAcceptable[0] + ")\n";
}
```

Рисунок 66 – Реализация функции сопоставления параметров микроклимата с требованиями ГОСТ после остановки периодических измерений

Сохранение результатов процесса измерения в виде текстового файла предоставляет пользователям возможность зафиксировать и в будущем обратиться к результатам измерения. Название текстового файла включает в себя дату и время начала запросов измерений. Формат файла .txt открывается встроенным в ОС Windows редактором «Блокнот». Примерная структура файла приведена на рисунке 67.

```
/ Журнал измерений: 05.05.2026. Время начала измерений: 21:48:55. Кол-во измерений: 49
| Интервал запроса измерений: 100 мсек.
Средние значения:  (выставлено условие -     Помещения с временным пребыванием людей (вестибюли,
 CO2 = 1054 см3/м3  - не нормируется
 T = 26,1 °C  - ВНЕ ДИАПАЗОНА (20 < T < 14)
 RH = 43,6 %  - не нормируется
/
[1] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[2] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[3] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[4] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[5] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[6] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[7] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[8] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,4 %
[9] CO2 = 1054 см3/м3 T = 26,1 °C RH = 43,5 %
```

Рисунок 67 – Примерная структура файла с результатами процесса измерения

Сохранение результатов происходит после остановки измерений. Записывается следующая информация:

– дата и время начала измерений;

– количество запросов измерений и интервал;

– средние значения и проверка соответствия нормам ГОСТ [1];

– история измерений.

Разработанная функция сохранения результатов в виде текстового файла представлена на рисунке 68.

```
string path = "./logs/" + date + "_" + hour + "h" + minute + "m" + second + "s.txt";

string header = "/ Журнал измерений: " + date + ". Время начала измерений: " + hour + ":" + minute + ":" + second + ". Кол-во измерений: " + counter + "\n" +
    "| Интервал запроса измерений: " + _settings.Interval + " мсек.\n" +
    header1 + "/\n";

_ = Directory.CreateDirectory("./logs/");
FileStream fstream = new FileStream(path, FileMode.OpenOrCreate);
byte[] text = Encoding.Default.GetBytes(header + fileLogs);
fstream.Write(text);
fstream.Close();
```

Рисунок 68 – Реализация функции сохранения результатов в виде текстового файла

Настройка IP-адреса сервера и периодичности обновления реализована в окне настроек Form2. Настройка IP-адреса блокируется во время подключения к серверу, тем самым исключается возможность возникновения ошибок в работе. Так как предусмотрена возможность отправки проверочного байта, пользователь может проверить корректность введённого адреса. Реализация функции проверки введённого адреса показана на рисунке 69.

```
watch.Start();
serverSocket.Connect(checkBoxLocalhost.Checked ? "127.0.0.1" : textBoxAddress.Text, 8080);
parentForm.addLog("Успешное подключение, отправка проверочного байта (0x" + ack.ToString() + ")", 3);

serverSocket.Send(buffer);
serverSocket.Receive(buffer);
watch.Stop();

if (buffer[0] == ack)
{
    consoleStatus.ForeColor = Color.Green;
    consoleStatus.Text = "Получен ответ от сервера! ~" + watch.ElapsedMilliseconds.ToString() + " мс";
    parentForm.addLog("Ответ от сервера (0x" + buffer[0].ToString() + ") совпадает с проверочным байтом (0x" + ack.ToString() + "), проверка завершена", 3);
}
else
{
    consoleStatus.ForeColor = Color.Red;
    consoleStatus.Text = "Получен неверный ответ от сервера! ~" + watch.ElapsedMilliseconds.ToString() + " мс";
    parentForm.addLog("Ответ от сервера (0x" + buffer[0].ToString() + ") не совпадает с проверочным байтом (0x" + ack.ToString() + "), проверка завершена", 2);
}
```

Рисунок 69 – Реализация функции проверки введённого IP-адреса сервера

Настройка периодичности обновления заблокирована в процессе запросов измерений. При закрытии окна настроек выставленные параметры сохраняются в локальный файл настроек с помощью класса Settings. Разработанная функция сохранения IP-адреса и периодичности обновления продемонстрирована на рисунке 70.

```
_settings.UseLocalhost = checkBoxLocalhost.Checked;
_settings.Interval = (int)numericUpDownDelay.Value;
_settings.DebugMode = checkBoxConsole.Checked;
_settings.ManualAddress = textBoxAddress.Text;

_settings.Save();
```

Рисунок 70 – Реализация функции настройки и сохранения IP-адреса сервера и периодичности обновления

Разработанные модули обеспечивают выполнения всех функций клиента, предусмотренных архитектурой программной части.

4 Демонстрация работы проекта

Демонстрация работы программно-аппаратного комплекса включает в себя описание работы и тестирование системы. В описании работы приводится примерный сценарий использования программы, разделённый на этапы, каждый из которых подробно описывается. Перед тестированием системы определяются ситуации и возможное поведение системы. Само тестирование проводится для проверки поведения комплекса при определённой ситуации.

В данной главе описана работа программно-аппаратного комплекса и проведено тестирование системы в различных сценариях.

4.1 Описание работы

Поскольку сервер разработан, как служба, он не может быть запущен обычным способом. Вместо этого используется системная утилита управления службами sc, которая позволяет управлять службами. Для упрощения операций установки, удаления и перезапуска службы предоставляются пакетные файлы формата .bat. Примерный список файлов, поставляемых вместе с исполняемым файлом сервера, представлен на рисунке 71.

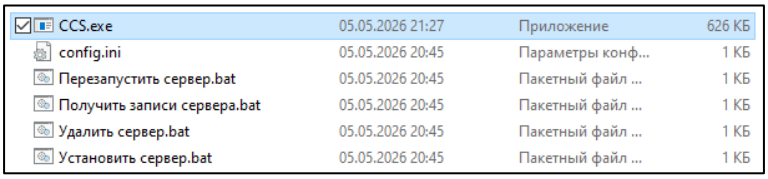


Рисунок 71 – Примерный список файлов сервера

Взаимодействие с диспетчером служб подразумевает создание или изменение записей в реестре, поэтому все пакетные файлы должны запускаться от имени администратора. При запуске пакетного файла открывается командная строка и отображается результат выполнения файла. Если отсутствует один или несколько пакетных файлов, при запуске сервера недостающие файлы будут восстановлены.

Установка сервера включает в себя создание новой службы, добавление её в реестр и запуск. Результат успешно выполненного пакетного файла установки сервера показан на рисунке 72.

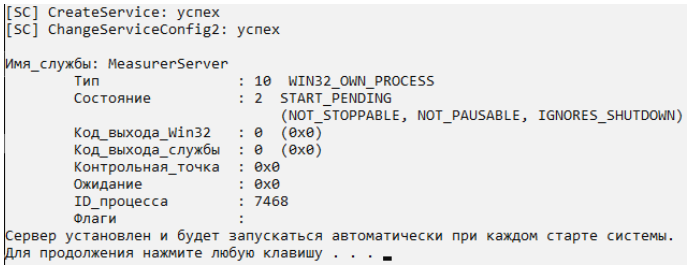


Рисунок 72 – Результат успешного выполнения пакетного файла установки сервера

Сервер при возникновении исключительных ситуаций формирует события, которые могут быть просмотрены системной программой «Просмотр событий» или системной утилитой wevtutil. Для удобства пользователю предоставляется пакетный файл, который запрашивает текущее состояние службы и последние десять событий. Дополнительно формируется файл states.log, дублирующий информацию с командной строки. Результат успешно выполненного пакетного файла отображения событий продемонстрирован на рисунке 73.

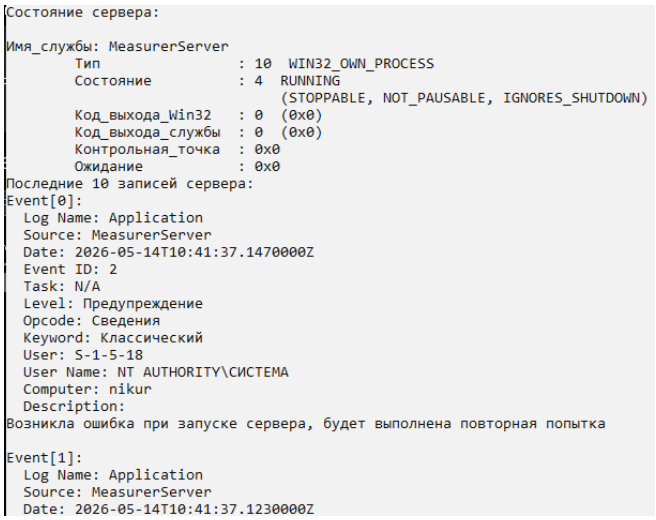


Рисунок 73 – Результат успешного выполнения пакетного файла отображения событий

Настройка сервера включает ограничение максимального числа подключений, время таймаута между попытками запуска сервера и время тайм-аута между попытками перезапуска сервера. Структура файла настроек представлена на рисунке 74.

```
[Settings]
MaxConnections=100
SecDelayRetry=10
SecDelayRestart=20
```

Рисунок 74 – Структура файла настроек сервера

Для применения настроек необходимо перезапустить службу. Это осуществляется с помощью соответствующего пакетного файла. Результат успешно выполненного пакетного файла перезапуска сервера продемонстрирован на рисунке 75.

```
Имя_службы: MeasurerServer
        Тип                : 10  WIN32_OWN_PROCESS
        Состояние          : 3  STOP_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        Код_выхода_Win32   : 0  (0x0)
        Код_выхода_службы  : 0  (0x0)
        Контрольная_точка  : 0x4
        Ожидание           : 0x0

Время ожидания 1 сек., нажмите любую клавишу для продолжения ...
[SC] DeleteService: успех
[SC] CreateService: успех
[SC] ChangeServiceConfig2: успех

Имя_службы: MeasurerServer
        Тип                : 10  WIN32_OWN_PROCESS
        Состояние          : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        Код_выхода_Win32   : 0  (0x0)
        Код_выхода_службы  : 0  (0x0)
        Контрольная_точка  : 0x0
        Ожидание           : 0x0
        ID_процесса        : 7620
        Флаги              :
Сервер перезапущен успешно.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 75 – Результат успешного выполнения пакетного файла перезапуска сервера

При необходимости пользователь может послать сигнал остановки и удалить сервер из списка служб с помощью соответствующего пакетного файла. Результат успешно выполненного пакетного файла удаления сервера показан на рисунке 76.

```
Имя_службы: MeasurerServer
        Тип                : 10  WIN32_OWN_PROCESS
        Состояние          : 3  STOP_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        Код_выхода_Win32   : 0  (0x0)
        Код_выхода_службы  : 0  (0x0)
        Контрольная_точка  : 0x4
        Ожидание           : 0x0

Время ожидания 0 сек., нажмите любую клавишу для продолжения ...
[SC] DeleteService: успех
Служба удалена.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 76 – Результат успешного выполнения пакетного файла удаления сервера

Успешный запуск сервера зависит от следующих условий:

– отладочная плата с ПЛИС подключена к устройству с сервером;

– сокет для сетевого взаимодействия с клиентами создан и настроен.

Примерный список событий, сигнализирующий об успешном запуске сервера, представлен на рисунке 77.

```
Event[0]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-16T20:02:25.2400000Z
  Event ID: 2
  Task: N/A
  Level: Сведения
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Сервер запущен успешно!

Event[1]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-16T20:02:24.8120000Z
  Event ID: 1
  Task: N/A
  Level: Сведения
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Служба запущена успешно

Event[2]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-16T20:02:24.7740000Z
  Event ID: 1
  Task: N/A
  Level: Сведения
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Служба запускается...
```

Рисунок 77 – Примерный список событий успешного запуска сервера

Запуск клиента может выполняться как локально на устройстве с запущенным сервером, так и на удалённо на другом устройстве. Главное окно, отображающееся при запуске приложения клиента, продемонстрировано на рисунке 78.

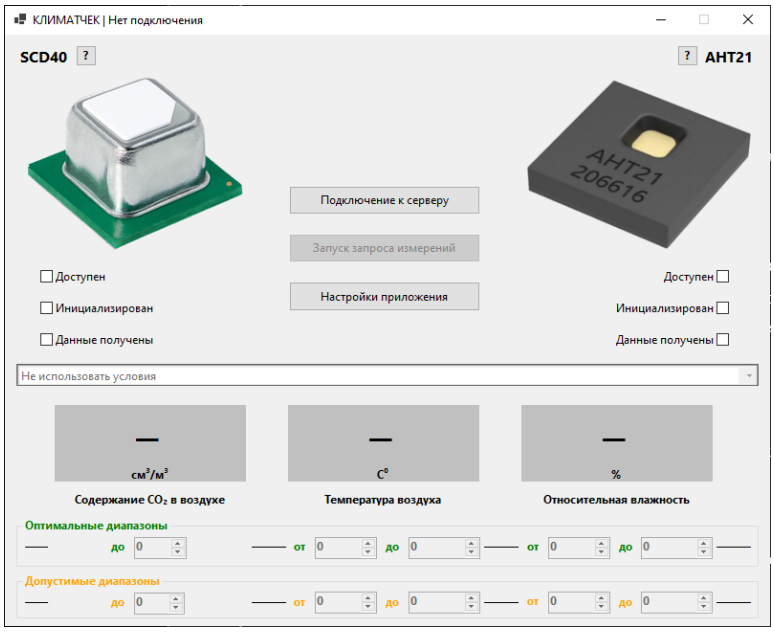


Рисунок 78 – Интерфейс главного окна клиента

Настройка приложения включает в себя настройки подключения и измерения. При настройке подключения пользователю предоставляется возможность проверки корректности введённого адреса сервера. Пример успешной проверки адреса сервера показан на рисунке 79.

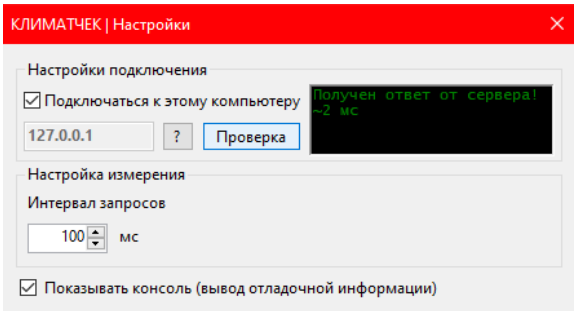


Рисунок 79 – Успешная проверка адреса сервера в настройках приложения

При нажатии на кнопку «Подключение к серверу» выполняется попытка подключения по указанному в настройках адресу сервера. Активация кнопки «Запуск запроса измерений» и изменение флагов состояний датчиков свидетельствует об успешном подключении к серверу. Примерный вид главного окна приложения представлен на рисунке 80.

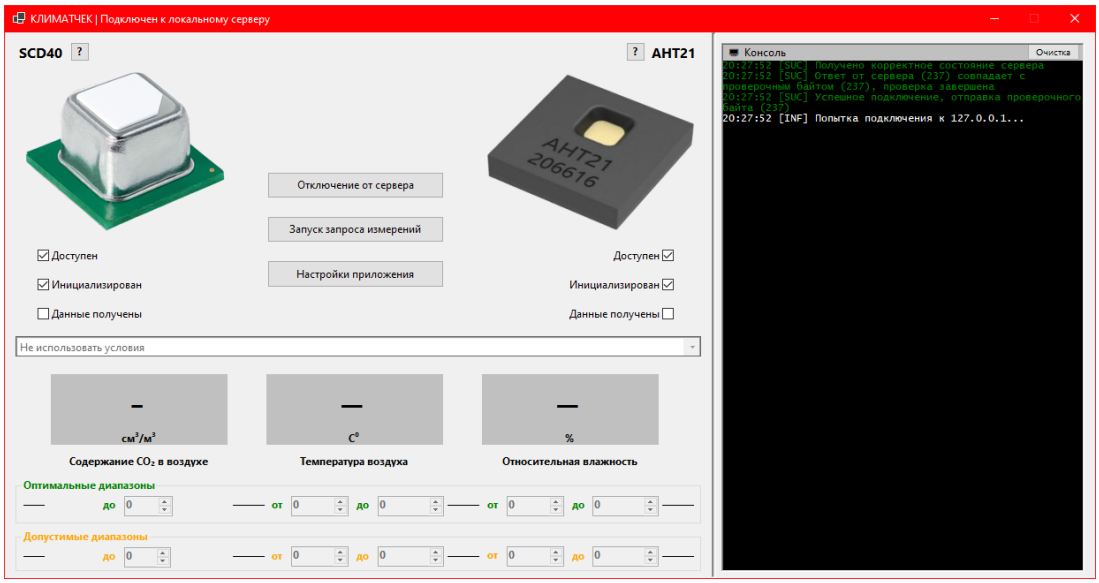


Рисунок 80 – Примерный вид главного окна при успешном подключении к серверу

При нажатии на кнопку «Запуск запроса измерений» начинается периодический сбор информации с датчиков с интервалом, указанным в настройках. При успешном запуске процесса изменяется содержимое элементов интерфейса, отвечающих за параметры микроклимата и состояния датчиков. Пользователь может выбирать готовое условие или устанавливать произвольные диапазоны, с которыми сопоставляются параметры микроклимата. Примерный вид главного окна при периодичных запросах измерений продемонстрирован на рисунке 81.

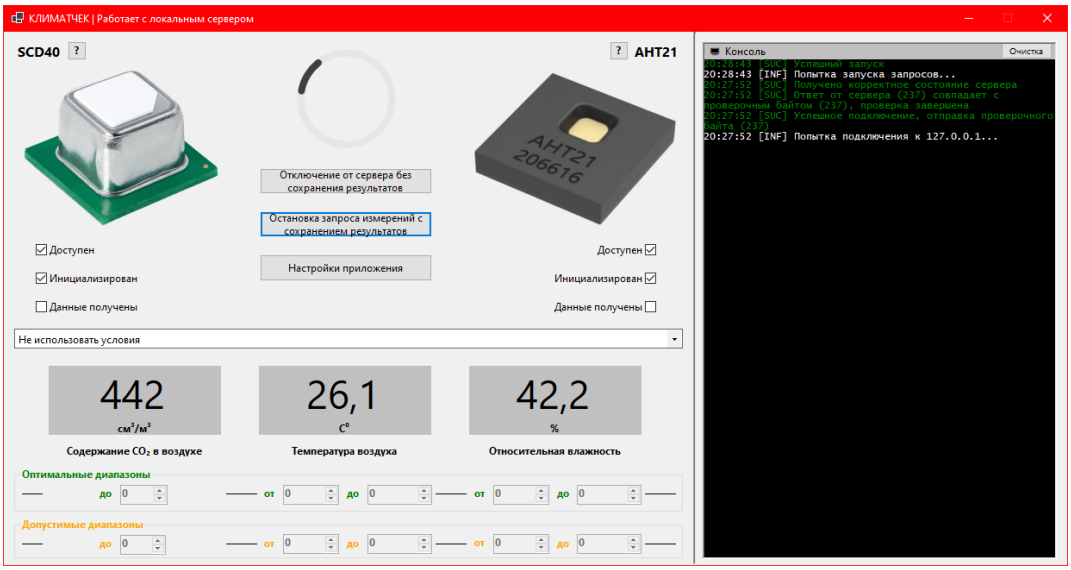


Рисунок 81 – Примерный вид главного окна при периодических измерениях

При остановке периодических измерений результаты измерений сохраняются в текстовый файл формата .txt. Однако если в процессе запросов произойдёт внезапное и преднамеренное отключение от сервера, результаты не будут сохранены в файл во избежание некорректной записей данных. Примерная структура файла с результатами измерений показана на рисунке 82.

```
/ Журнал измерений: 16.05.2026. Время начала измерений: 20:28:43. Кол-во измерений: 3978
| Интервал запроса измерений: 100 мсек.
Средние значения:  (условие не выставлено):
 CO2 = 525 см3/м3
 T = 29,0 °C
 RH = 38,6 %
/
[1] CO2 = 556 см3/м3 T = 33,9 °C RH = 33,9 %
[2] CO2 = 556 см3/м3 T = 33,9 °C RH = 33,9 %
[3] CO2 = 556 см3/м3 T = 33,9 °C RH = 33,9 %
[4] CO2 = 556 см3/м3 T = 33,9 °C RH = 34,0 %
[5] CO2 = 556 см3/м3 T = 33,9 °C RH = 34,0 %
[6] CO2 = 556 см3/м3 T = 34,0 °C RH = 34,0 %
[7] CO2 = 556 см3/м3 T = 34,0 °C RH = 34,0 %
[8] CO2 = 556 см3/м3 T = 34,0 °C RH = 34,0 %
[9] CO2 = 556 см3/м3 T = 34,0 °C RH = 33,9 %
[10] CO2 = 556 см3/м3 T = 34,0 °C RH = 33,9 %
```

Рисунок 82 – Примерная структура файла с результатами измерений

Для получения дополнительной информации предусмотрена консоль, которая может быть активирована в настройках клиента. Ошибки, предупреждения и успехи сигнализируются соответствующими цветами, дополнительно выводится время события. Примерный вид главного окна с открытой консолью представлен на рисунке 83.

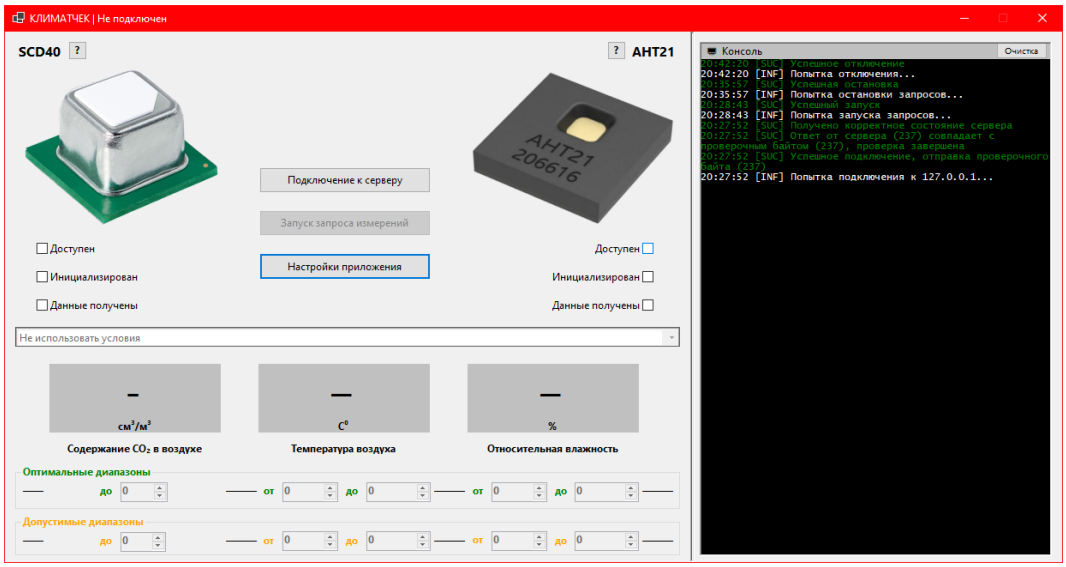


Рисунок 83 – Примерный вид главного окна с открытой консолью

Вызов документации по датчику может быть необходим пользователю в случае, когда требуется узнать информацию о характеристиках датчика. Для этого в программе предусмотрены кнопки справки, расположенные рядом с названиями датчиков. При нажатии на одну из кнопок открывается документация формата .pdf.

4.2 Тестирование системы

Главная цель тестирования программно-аппаратного комплекса – показать его корректную работоспособность и устойчивость к нештатным ситуациям. Корректная работоспособность предполагает, что все функции комплекса выполняются в точности, как задумано на этапе проектирования. Так как аппаратная часть наиболее подвержена внешним воздействиям, могут возникать такие нештатные ситуации, как отключение одного из датчиков и потеря соединения с ПЛИС. Это в свою очередь может привести к завершению работы сервера, что тоже относится к непредвиденным ситуациям. Обеспечение устойчивости к нештатным ситуациям включает в себя реализацию дополнительных проверок, которые исключат возникновение неопределённого поведения всего комплекса.

Для реализации цели тестирования сформулированы следующие сценарии:

– установка сервера в виде службы при уже установленной службе;

– удаление неустановленной службы сервера;

– перезапуск неустановленной службы сервера;

– отключение ПЛИС во время работы комплекса;

– отключение службы сервера во время работы комплекса;

– сохранение результатов измерения при штатной работе комплекса.

Тестирование комплекса производилось на следующей конфигурации:

– ОС Windows 10;

– версия Microsoft Visual Studio 2019;

– подключение отладочной платы с ПЛИС через интерфейс USB-UART;

– состав датчиков: AHT21 и SCD40.

Установка сервера в виде службы при уже установленной службе недопустима, поскольку в системном реестре уже существует соответствующая запись, связанная со службой, и в системе службы не могут иметь одинаковое название. Предполагается, что установка не выполнится и в командной строке отобразится соответствующая ошибка. Для выполнения сценария был запущен пакетный файл, отвечающий за установку сервера в виде службы. Выполнение пакетного файла продемонстрировано на рисунке 84.

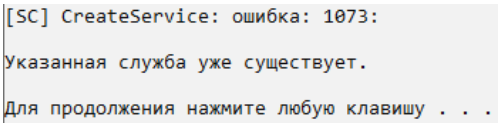


Рисунок 84 – Выполнение пакетного файла установки сервера в виде службы

При попытке установить сервер была возвращена ошибка, что указанная служба уже существует. Установленная служба осталась в реестре, что показано на рисунке 85.

```
C:\Users\user>sc query MeasurerServer

Имя_службы: MeasurerServer
        Тип                : 10  WIN32_OWN_PROCESS
        Состояние          : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        Код_выхода_Win32   : 0  (0x0)
        Код_выхода_службы  : 0  (0x0)
        Контрольная_точка  : 0x0
        Ожидание           : 0x0
```

Рисунок 85 – Наличие записи о установленной службе, проверенное с помощью системной утилиты sc

Удаление неустановленной службы сервера не может быть выполнено, поскольку запись о службе отсутствует в системном реестре. Ожидается, что удаление не выполнится и в командной строке отобразится соответствующая ошибка. Для выполнения сценария был запущен пакетный файл, отвечающий за удаление сервера в виде службы. Выполнение пакетного файла представлено на рисунке 86.

```
[SC] OpenService: ошибка: 1060:

Указанная служба не установлена.

Для продолжения нажмите любую клавишу . . .
```

Рисунок 86 – Выполнение пакетного файла удаления сервера в виде службы

При попытке удалить сервер была возвращена ошибка, что указанная служба не установлена. Служба сервера отсутствует в реестре, что показано на рисунке 87.

```
C:\Users\user>sc query MeasurerServer
[SC] EnumQueryServicesStatus:OpenService: ошибка: 1060:

Указанная служба не установлена.
```

Рисунок 87 – Отсутствие записи о службе, проверенное с помощью системной утилиты sc

Перезапуск неустановленной службы сервера невозможен из-за отсутствия записи о службе в системном реестре. Предполагается, что перезапуск не выполнится и в командной строке отобразится соответствующая ошибка. Для выполнения сценария был запущен пакетный файл, отвечающий за перезапуск сервера. Выполнение пакетного файла продемонстрировано на рисунке 88.

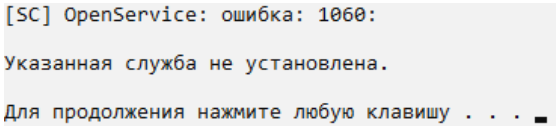


Рисунок 88 – Выполнение пакетного файла перезапуска сервера

При попытке перезапустить сервер была возвращена ошибка, что указанная служба не установлена. Служба сервера отсутствует в реестре, что представлено на рисунке 89.

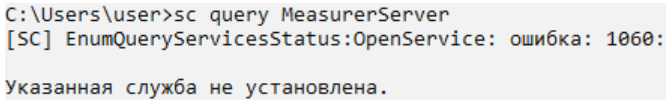


Рисунок 89 – Отсутствие записи о службе, проверенное с помощью системной утилиты sc

Отключение отладочной платы с ПЛИС во время работы комплекса критично для всего комплекса, поскольку именно ПЛИС контролирует работу датчиков и связывает программную часть с датчиками. Причинами этого сценария могут быть отключение питания отладочной платы, нарушение соединения между компьютером и отладочной платой и другие причины. Ожидается, что сервер сформирует событие с ошибкой соединения, закроет соединения с клиентами и периодически будет запрашивать соединение с ПЛИС через последовательный порт. Клиент, в свою очередь, получит сообщение о разрыве соединения с сервером. Для выполнения сценария было отключено питание отладочной платы с ПЛИС при работе сервера и периодичных запросах клиента. Список событий, сформированных сервером, показан на рисунке 90.

```
Event[0]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-15T19:37:40.2470000Z
  Event ID: 3
  Task: N/A
  Level: Предупреждение
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Сервер завершил работу, но служба не остановлена, будет выполнен повторный запуск

Event[1]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-15T19:37:40.1000000Z
  Event ID: 2
  Task: N/A
  Level: Ошибка
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Потеряно соединение с портом Silicon Labs CP210x

Event[2]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-15T19:37:35.3030000Z
  Event ID: 2
  Task: N/A
  Level: Сведения
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Сервер запущен успешно!
```

Рисунок 90 – Список событий при отключении ПЛИС во время работы комплекса

Сервер сформировал ошибку, что соединение с последовательным портом было потеряно, далее сервер проинформировал о перезапуске сервера. Последующие события связаны с попытками соединения с ПЛИС. Состояние клиента с сообщением о потере соединения с сервером продемонстрировано на рисунке 91.

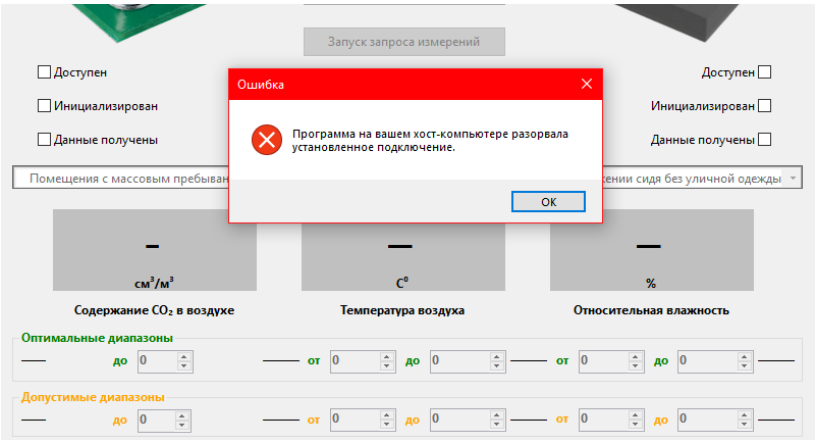


Рисунок 91 – Состояние клиента с сообщением о потере соединения с сервером

Отключение службы сервера во время работы комплекса приводит к его остановке, поскольку сервер отправляет команды измерения, принимает исходные данные с ПЛИС и обрабатывает запросы клиентов на получение значений параметров микроклимата. Предполагается, что сервер сформирует соответствующее событие и завершит свою работу, а на стороне клиента отобразится сообщение об ошибке соединения с сервером. Для выполнения сценария был запущен пакетный файл, отвечающий за удаление сервера в виде службы. Выполнение пакетного файла представлено на рисунке 92.

```
Имя_службы: MeasurerServer
        Тип                : 10  WIN32_OWN_PROCESS
        Состояние          : 3  STOP_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        Код_выхода_Win32   : 0  (0x0)
        Код_выхода_службы  : 0  (0x0)
        Контрольная_точка  : 0x4
        Ожидание           : 0x0

Время ожидания 0 сек., нажмите любую клавишу для продолжения ...
[SC] DeleteService: успех
Служба удалена.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 92 – Выполнение пакетного файла удаления сервера в виде службы

Вывод командной строки показывает, что службе был отправлен сигнал остановки и, спустя время ожидания, служба была успешно удалена. Список событий, сформированных сервером, показан на рисунке 93.

```
Последние 10 записей сервера:
Event[0]:
  Log Name: Application
  Source: MeasurerServer
  Date: 2026-05-15T19:52:24.0090000Z
  Event ID: 1
  Task: N/A
  Level: Сведения
  Opcode: Сведения
  Keyword: Классический
  User: S-1-5-18
  User Name: NT AUTHORITY\СИСТЕМА
  Computer: WIN-0E74UJ399II
  Description:
Служба остановлена успешно
```

Рисунок 93 – Список событий при отключении сервера во время работы комплекса

Сервер сформировал событие о успешной остановке службы. Состояние клиента с сообщением о потере соединения с сервером продемонстрировано на рисунке 94.

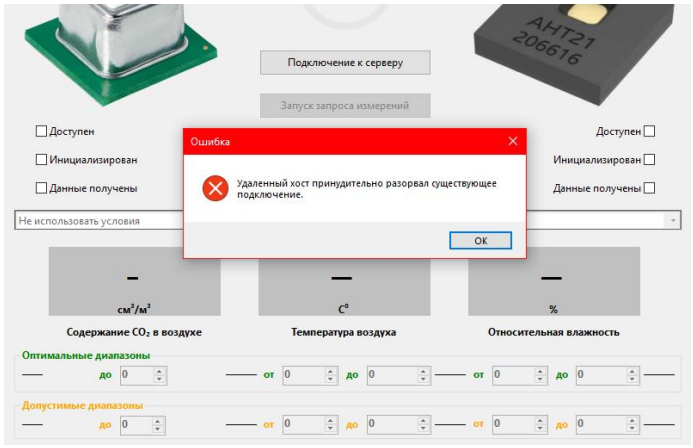


Рисунок 94 – Состояние клиента с сообщением о потере соединения с сервером

Сохранение результатов измерений при штатной работе комплекса предполагает, что будет сохранена история измерений, а также средние значения параметров микроклимата. Для выполнения сценария на клиенте были запущены и через некоторое время остановлены периодические запросы результатов измерений. Ожидается, что в папке logs будет создан файл с журналом измерений. Выполнение и остановка периодических запросов представлены на рисунке 95.

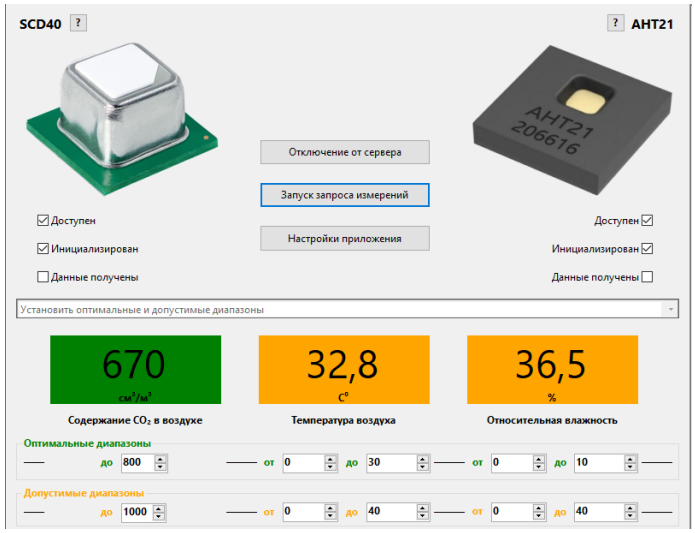


Рисунок 95 – Выполнение и остановка периодических запросов на клиенте

После остановки периодических запросов в папке logs был создан текстовый файл, в названии которого указывалось время начала измерения. Структура файла показана на рисунке 96.

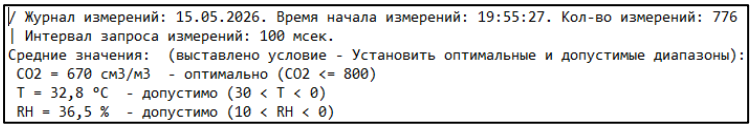


Рисунок 96 – Структура журнала измерения

Результаты тестирования представлены в таблице 5.

Таблица 5 – Таблица результатов тестирования

Сценарий	Ожидаемый результат	Реальный результат	Итог сценария
Установка сервера в виде службы при уже установленной службе	Событие об ошибке установки, установка не выполняется	Ошибка «Указанная служба уже существует», установка не выполнилась	Пройден
Удаление неустановленной службы сервера	Событие об ошибке удаления, удаление не выполняется	Ошибка «Указанная служба не установлена», удаление не выполнилось	Пройден
Перезапуск неустановленной службы сервера	Событие об ошибке перезапуска, перезапуск не выполняется	Ошибка «Указанная служба не установлена», перезапуск не выполнился	Пройден
Отключение ПЛИС во время работы комплекса	Событие об ошибке соединения, закрытие соединения с клиентами, периодический опрос соединения	События «Потеряно соединение с портом», перезапуск сервера и ожидание подключения ПЛИС	Пройден
Отключение службы сервера во время работы комплекса	Событие о успешном завершении работы, закрытие соединения с клиентами	Событие «Служба остановлена успешно», сервер остановлен	Пройден
Сохранение результатов измерения при штатной работе комплекса	Корректное сохранение журнала измерений в текстовый файл	Журнал измерений был сформирован и сохранён в файл	Пройден

Тестирование показало, что при разработке программно-аппаратного комплекса были учтены возможные нештатные ситуации и разработаны меры для их обработки. Реальные результаты сценариев полностью совпали с ожидаемыми, что подтверждает корректность работы системы и её устойчивость к нештатным ситуациям проекта.

5 Безопасность жизнедеятельности

В данном разделе описаны правила безопасности при работе с программно-аппаратным комплексом. Вся информация основана на требованиях, которые описаны в следующих документах:

– пятый раздел «Охрана труда» трудового кодекса Российской Федерации;

– Санитарные Правила и Нормы 1.2.3685-21;

– раздел «Система Стандартов Безопасности Труда. Рабочее место при выполнении работ сидя. Общие эргономические требования» ГОСТ 12.2.032-78;

– приказ «Правила по охране труда при эксплуатации электроустановок».

5.1 Правила безопасности при работе с ПК

При работе с ПК существуют факторы, которые могут быть вредны и даже опасны для человека:

– повышенный уровень электромагнитного излучения от монитора, системного блока и дополнительных периферийных устройств [11];

– длительное статическое напряжение мышц спины и шеи, включая неправильную осанку, неправильное расположение монитора, неправильная организация рабочего места, недостаточная физическая активность [11];

– поражение электрическим током при нарушении изоляции по причине старения изоляции, повреждения изоляционной оболочки, неправильная эксплуатация оборудования и контакт с водой [12];

– недостаточная освещённость рабочего места [11].

Для исключения вышеперечисленных факторов предусмотрены следующие требования:

– требования к рабочему месту;

– режим труда и отдыха;

– электробезопасность;

СанПиН 1.2.3685-21 и ГОСТ 12.2.032-78 регламентируют следующие требования к рабочему месту:

– площадь рабочего места не менее 4,5 м^2 [11];

– расстояние от глаз до экрана – 50-70 см [11];

– высота стола – 680-800 мм [13];

– освещённость рабочего стола – 300-500 лк [11].

Раздел «Охрана труда» трудового кодекса Российской Федерации указывает следующие требования к режиму труда и отдыха:

– время работы за ПК – 45-60 мин;

– перерыв между работой за ПК – 10-15 мин [14].

Электробезопасность регламентирует приказ «Правила по охране труда при эксплуатации электроустановок», где описаны следующие правила:

– подключение ПК к розетке с заземлением [12];

– запрет на использование ПК с повреждённым кабелем питания [12].

5.2 Правила безопасности при работе с аппаратной частью

Работа с отладочной платой с ПЛИС и коммутационным стендом включает следующие риски и опасные факторы:

– остаточное напряжение на конденсаторах, при прикосновении к которым может произойти поражение током [12];

– короткое замыкание при соединении двух точек электрической цепи с низким сопротивлением [12];

– электрические перегрузки, приводящие к выходу из строя электронных компонентов [12].

Для предупреждения вышеописанных опасных факторов существуют следующие требования:

– требования к установке и монтажу плат;

– защита от статического электричества;

– действия в аварийных ситуациях.

Правила установки и монтажа плат регламентируются в приказе «Правила по охране труда при эксплуатации электроустановок», где описываются следующие требования:

– подключение отладочной платы и коммутационного стенда осуществляется только при выключенном питании [12];

– питание для датчиков и светодиода подаётся только от стабилизированного источника на отладочной плате с ПЛИС [12];

– обязательная проверка правильности соединений перед включением [12].

Информация о защите от статического электричества описана в приказе «Правила по охране труда при эксплуатации электроустановок»:

– при работе с отладочной платой и коммутационным стендом использовать антистатический браслет или заземлённый коврик [12];

– без необходимости не прикасаться к выводам плат [12].

Возможные действия в аварийных ситуациях описываются в приказе «Правила по охране труда при эксплуатации электроустановок»:

– при появлении дыма или запаха гари – немедленно обесточить комплекс [12];

– при поражении электрическим током во время работы комплекса – отключить питание и вызвать скорую помощь [12].

Заключение

В результате ВКР был создан и протестирован работоспособный программно-аппаратный комплекс для измерения и контроля микроклимата. Поставленные в техническом задании цель и задачи ВКР выполнены в полном объёме.

В процессе ВКР решены следующие поставленные задачи: проведён анализ существующих комплексов измерения и контроля микроклимата, выявлены их достоинства и недостатки; изучены технологии разработки и эксплуатации открытых аппаратных платформ.

В первой части дипломного проектирования была спроектирована архитектура комплекса, включающая аппаратную часть сбора данных и программную часть с серверным приложением и клиентом с графическим интерфейсом.

Для используемой аппаратной платформы ПЛИС XC6SLX16-2FTG256C выбрана и настроена среда разработки Xilinx ISE Design Suite 14.7, изучен и использован язык описания аппаратуры VHDL. Для связи ПЛИС с датчиками AHT21 и SCD40 спроектирован коммутационный стенд и использован протокол связи I2C.

Программная часть комплекса реализована в рамках клиент-серверного приложения. Сервер реализован на C++, для работы в трёх режимах: работа с последовательным портом, TCP-сервер и работа в режиме службы. Клиент реализован на C# и содержит функции анализа и визуализации параметров микроклимата по требованиям ГОСТ.

В дальнейшем комплекс может быть усовершенствован. Разработчик планирует добавить функции поддержки дополнительных датчиков, реализовать веб-интерфейс для контроля параметров микроклимата без установки клиентского ПО, реализовать динамические графики для подробной визуализации параметров микроклимата.

Список использованных источников

1. ГОСТ 30494-2011. Межгосударственный стандарт. Здания жилые и общественные. Параметры микроклимата в помещениях. – URL: https://normativ.kontur.ru/document?moduleId=9&documentId=470521&ysclid=mpdqfyybs7375309537 (дата обращения: 15.05.2026).

2. Научно-Техническое предприятие «ТКА». Научные разработки – URL: https://www.tkaspb.ru/about/razrabotki/ (дата обращения 17.05.2026).

3. Компания Testo Instruments Co. Ltd. Измерительные приборы. – URL: https://www.testo.ru/ru-RU/pribory/vse-izmeritelnye-pribory/ (дата обращения 17.05.2026).

4. Фирма Xiaomi. Приборы мониторинга температуры и относительной влажности. – URL: https://www.mi.com/global/product-list/environment-appliance/temperature-humidity-monitors/ (дата обращения 17.05.2026).

5. Глухова Л. А., Бахтизин В. В. Технологии разработки ПО. Учебное пособие. // БГУИР. – 2010. – С. 82–109.

6. Data Sheet AHT21. – URL: https://cdn.compacttool.ru/downloads/AHT21.pdf (дата обращения 17.05.2026).

7. CD DS SCD40 SCD41 Datasheet D1. – URL: https://sensirion.com/media/documents/E0F04247/631EF271/CD_DS_SCD40_SCD41_Datasheet_D1.pdf (дата обращения 17.05.2026).

8. 5048 Datasheet. – URL: https://static.chipdip.ru/lib/573/DOC016573311.pdf (дата обращения 18.05.2026).

9. Семёнов Б. Ю. Шина I2C в радиотехнических конструкциях. // СОЛОН-Пресс. – 2010. – С. 5–21.

10. Лакамера Д. Архитектура встраиваемых систем. // ДМК Пресс. – 2023. – С. 172–202.

11. СанПиН 1.2.3685-21. Гигиенические нормативы и требования к обеспечению безопасности (или) безвредности для человека факторов среды обитания. – URL: https://nagut6.gosuslugi.ru/netcat_files/174/2801/SP123685_21.pdf?ysclid=mpdqk317ox122042059 (дата обращения 19.05.2026).

12. Приказ Минтруда РФ от 15 декабря 2020 г. № 903н «Об утверждении правил по охране труда при эксплуатации электроустановок». – URL: https://normativ.kontur.ru/document?moduleId=1&documentId=501479&ysclid=mpdql7pfc377303705 (дата обращения 19.05.2026).

13. ГОСТ 12.2.032-78. Система Стандартов Безопасности Труда. Рабочее место при выполнении работ сидя. Общие эргономические требования. – URL: https://normativ.kontur.ru/document?moduleId=9&documentId=222135&ysclid=mpdqkpwl5s151323255 (дата обращения 19.05.2026).

14. Трудовой кодекс Российской федерации от 30.12.2001 №197-ФЗ (ред. от 29.12.2025): Охрана труда. – Собрание законодательства Российской Федерации. – 07.01.2002. – С. 91–128.

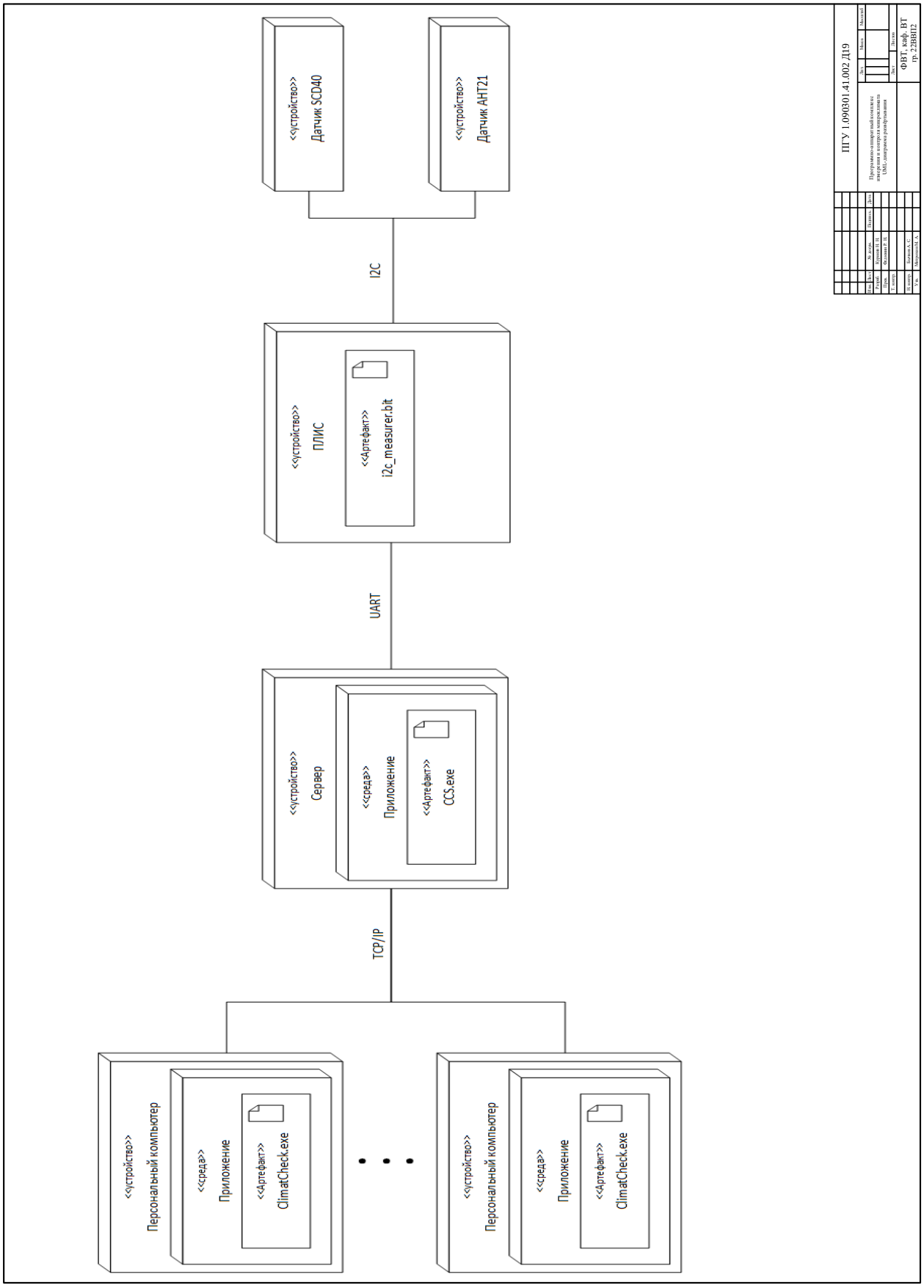
Приложение А

(обязательное)

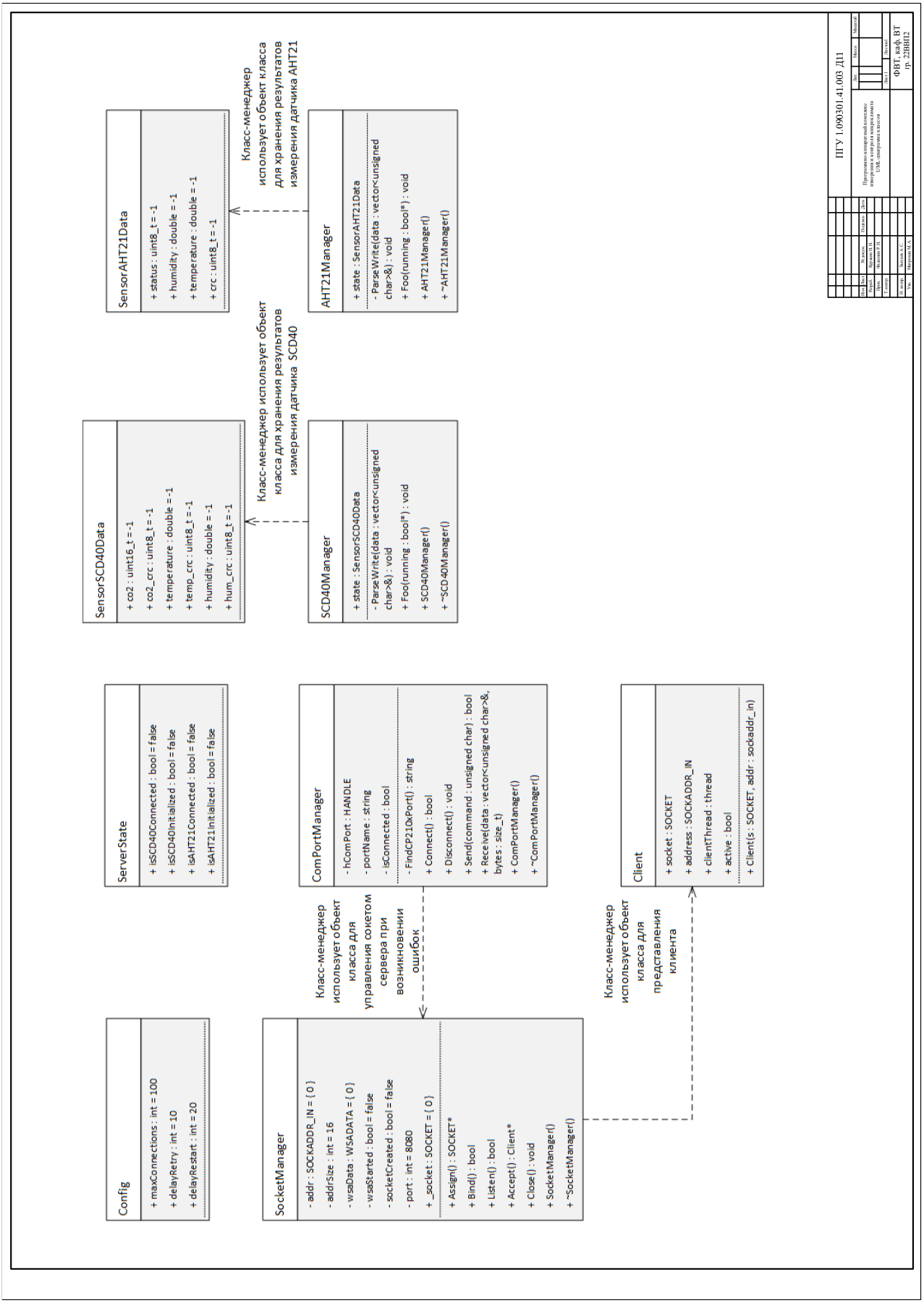
Программно-аппаратный комплекс измерения и контроля микроклимата

UML-диаграммы

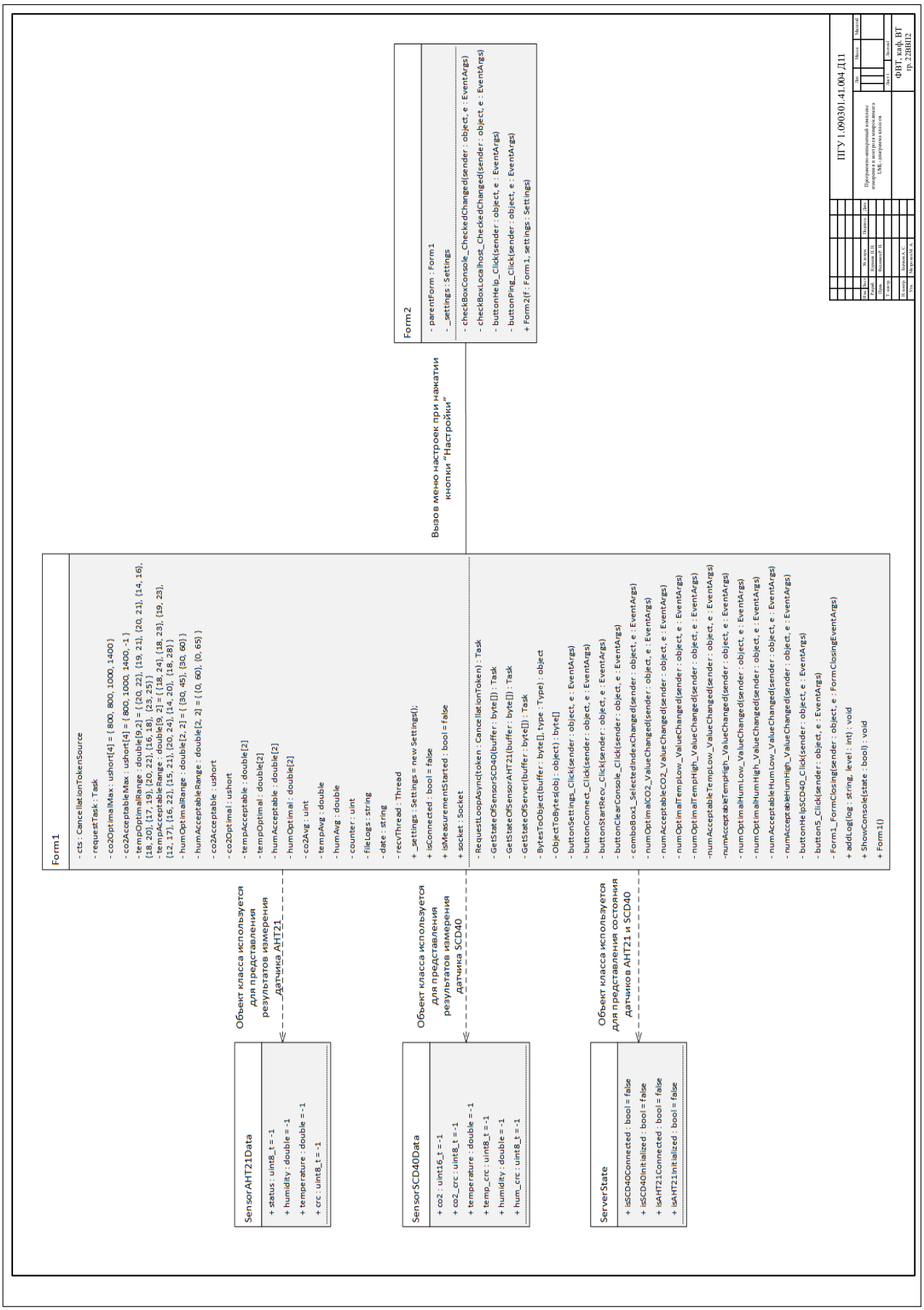
Приложение А.1 – Диаграмма развёртывания



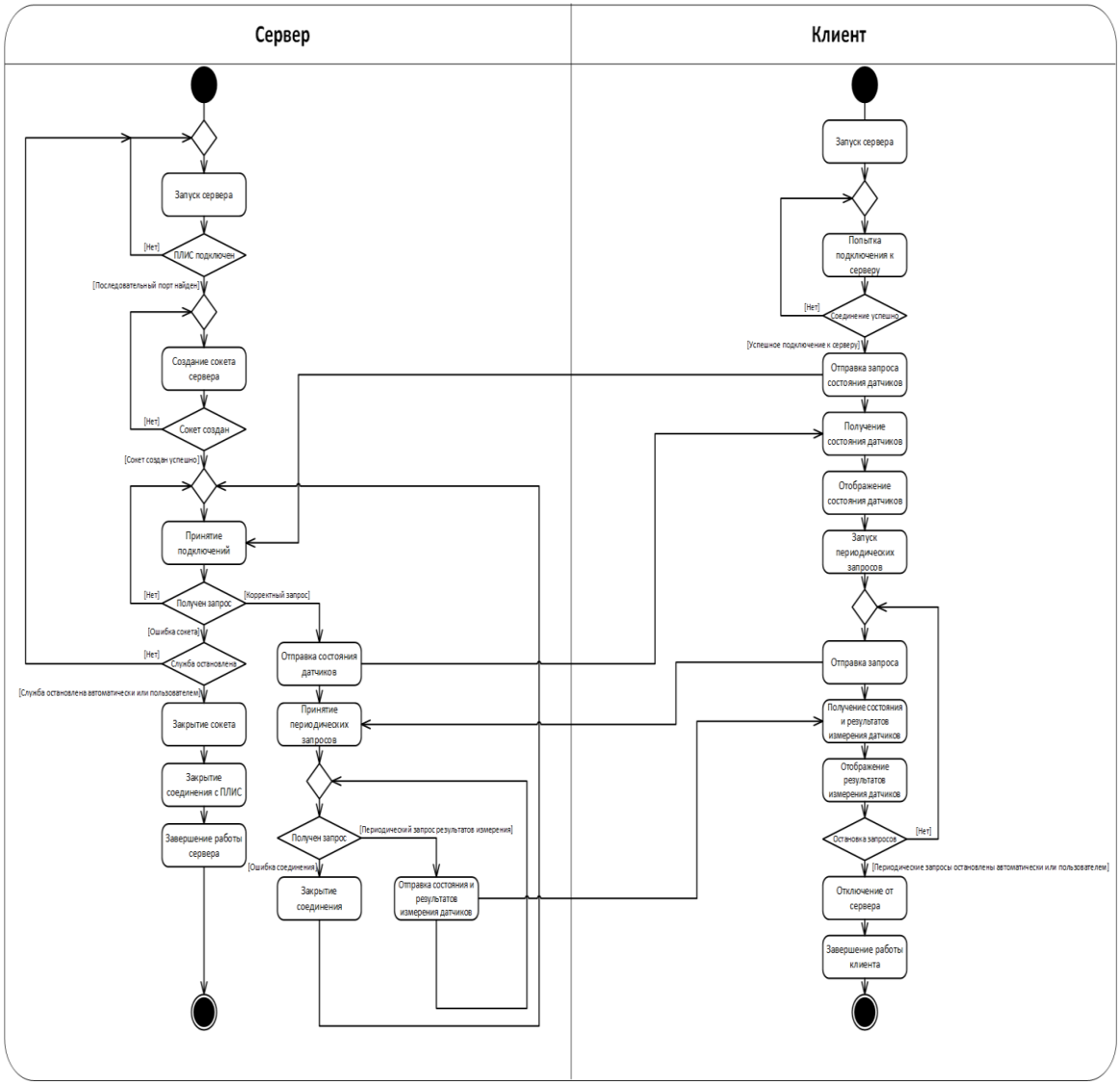
Приложение А.2 – Диаграмма классов сервера



Приложение А.3 – Диаграмма классов клиента



Приложение А.4 – Диаграмма деятельности



					ПГУ 1.090301.41.005 Д17	
Изм.	Лист	№ докум.	Подпись	Дата	Программно-аппаратный комплекс измерения и контроля микроклимата UML-диаграмма деятельности	Лит. / Масса / Масштаб
Разраб.		Куралин И. И.				Лист 1 / Листов1
Пров.		Федоцкая Р. И.				
Т. контр.						
Н. контр.		Быченя А. С.			ФВТ, каф. ВТ гр. 22ВВП2	
Утв.		Митрохин М. А.				

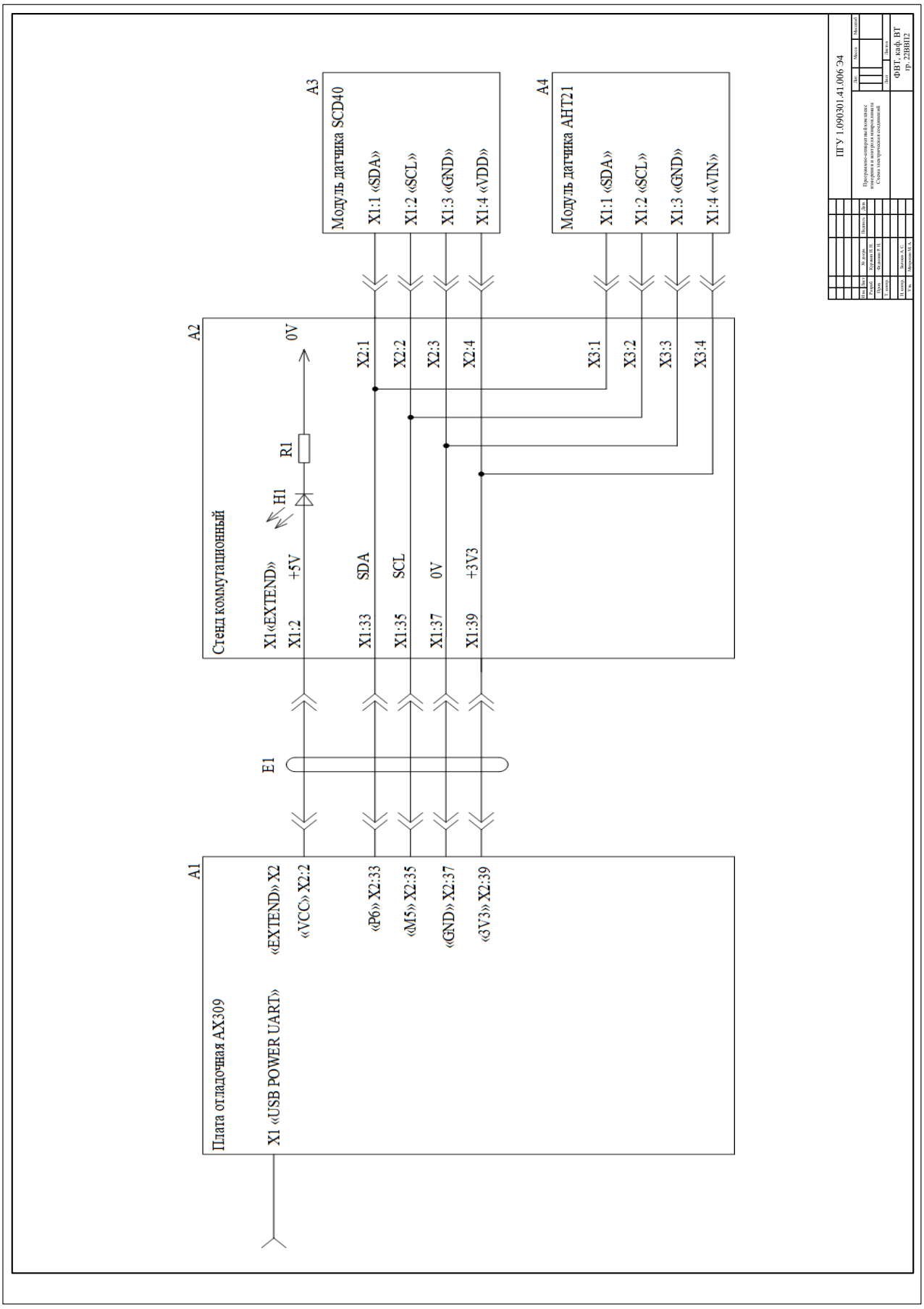
Приложение Б

(обязательное)

Программно-аппаратный комплекс измерения и контроля микроклимата

Схема электрическая

Приложение Б.1 – Схема электрическая соединений



Приложение В

(обязательное)

Программно-аппаратный комплекс измерения и контроля микроклимата

Листинг программ

Приложение В.1 – Листинг программы клиента

Файл Form1.cs

```
public partial class Form1 : Form
{
    public static Settings _settings = new();
    public bool isConnected = false;
    public bool isMeasurementStarted = false;
    public Socket socket;

    private CancellationTokenSource cts;
    private Task requestTask;

    private async Task RequestLoopAsync(CancellationToken token)
    {
        byte[] buffer = new byte[1024];
        ServerState? state;
        SensorSCD40Data? sensor1;
        SensorAHT21Data? sensor2;

        ushort co2 = 0;
        double temp = 0;
        double hum = 0;
        uint counterSensor1 = 0, counterSensor2 = 0;

        counter = 0;
        co2Avg = 0;
        tempAvg = 0;
        humAvg = 0;
        fileLogs = "";

        date = DateTime.Now.ToShortDateString();
        hour = DateTime.Now.Hour.ToString();
        minute = DateTime.Now.Minute.ToString();
        second = DateTime.Now.Second.ToString();

        while (!token.IsCancellationRequested)
        {
            try
            {
                await Task.Run(() => GetStateOfServer(buffer), token);
```

```
                            state       =       (ServerState?)BytesToObject(buffer,
typeof(ServerState));
                            if (state == null)
                                break;
                            checkBoxSCD40Connected.Checked                          =
state.Value.isSCD40Connected;
                            checkBoxAHT21Connected.Checked                          =
state.Value.isAHT21Connected;
                            checkBoxSCD40Initialized.Checked                        =
state.Value.isSCD40Initialized;
                            checkBoxAHT21Initialized.Checked                        =
state.Value.isAHT21Initialized;

                            string measurementThis = "[" + (counter + 1) + "] ";

                            await   Task.Run(()   =>   GetStateOfSensorSCD40(buffer),
token);
                            sensor1    =    (SensorSCD40Data?)BytesToObject(buffer,
typeof(SensorSCD40Data));
                            if (sensor1 == null)
                                break;
                            if (state.Value.isSCD40Initialized)
                            {
                                co2 = sensor1.Value.co2;
                                co2Avg += co2;

                                infoCO2.Text = co2.ToString();
                                int index = comboBox1.SelectedIndex;
                                if (index == 3 || index == 12)
                                    tableCO2.BackColor   =   co2   <=   co2Optimal   ?
Color.Green : Color.Red;
                                else if (index == 0 || index == 9 || index == 10 ||
index == 18 || index == 19)
                                    tableCO2.BackColor = Color.Silver;
                                else
                                    if (co2 <= co2Optimal)
                                        tableCO2.BackColor = Color.Green;
                                    else tableCO2.BackColor = co2 <= co2Acceptable ?
Color.Orange : Color.Red;
```

```
                measurementThis += "CO2 = " + co2.ToString() + "
см3/м3 ";

                checkBoxSCD40Recv.Checked = true;
                counterSensor1++;
            }
            else
            {
                measurementThis += "CO2 = - см3/м3 ";

                infoCO2.Text = "-";
                tableCO2.BackColor = Color.Silver;
            }

            await Task.Run(() => GetStateOfSensorAHT21(buffer),
token);
            sensor2 = (SensorAHT21Data?)BytesToObject(buffer,
typeof(SensorAHT21Data));
            if (sensor2 == null)
                break;
            if (state.Value.isAHT21Initialized)
            {
                temp = sensor2.Value.temperature;
                hum = sensor2.Value.humidity;
                tempAvg += temp;
                humAvg += hum;

                infoTemp.Text = string.Format("{0:F1}", temp);
                int index = comboBox1.SelectedIndex;
                if (index == 0)
                    tableTemp.BackColor = Color.Silver;
                else
                {
                    if (temp >= tempOptimal[0] && temp <=
tempOptimal[1])
                        tableTemp.BackColor = Color.Green;
                    else tableTemp.BackColor = temp >=
tempAcceptable[0] && temp <= tempAcceptable[1] ? Color.Orange : Color.Red;
                }
```

```
                                measurementThis += "T = " + string.Format("{0:F1}",
temp) + " °C ";

                                infoHum.Text = string.Format("{0:F1}", hum);
                                index = comboBox1.SelectedIndex;
                                if (index == 0 || index == 10)
                                    tableHum.BackColor = Color.Silver;
                                else
                                {
                                    if (hum >= humOptimal[0] && hum <= humOptimal[1])
                                        tableHum.BackColor = Color.Green;
                                    else tableHum.BackColor = hum >= humAcceptable[0]
&& hum <= humAcceptable[1] ? Color.Orange : Color.Red;
                                }

                                measurementThis += "RH = " + string.Format("{0:F1}",
hum) + " % \n";

                                checkBoxAHT21Recv.Checked = true;
                                counterSensor2++;
                            }
                            else
                            {
                                measurementThis += "T = - °C ";
                                measurementThis += "RH = - % \n";

                                infoTemp.Text = "-";
                                tableTemp.BackColor = Color.Silver;
                                infoHum.Text = "-";
                                tableHum.BackColor = Color.Silver;
                            }

                            counter++;
                            fileLogs += measurementThis;

                            await Task.Delay(_settings.Interval / 4, token);
                            checkBoxSCD40Recv.Checked = false;
                            checkBoxAHT21Recv.Checked = false;
                            await Task.Delay(_settings.Interval * 3 / 4, token);
                        }
                        catch (SocketException ex)
```

```
{
        numOptimalCO2.Enabled = false;
        numAcceptableCO2.Enabled = false;
        numOptimalCO2.Value = 0;
        numAcceptableCO2.Value = 0;

        numOptimalTempLow.Enabled = false;
        numOptimalTempHigh.Enabled = false;
        numAcceptableTempLow.Enabled = false;
        numAcceptableTempHigh.Enabled = false;
        numOptimalTempLow.Value = 0;
        numOptimalTempHigh.Value = 0;
        numAcceptableTempLow.Value = 0;
        numAcceptableTempHigh.Value = 0;

        numOptimalHumLow.Enabled = false;
        numOptimalHumHigh.Enabled = false;
        numAcceptableHumLow.Enabled = false;
        numAcceptableHumHigh.Enabled = false;
        numOptimalHumLow.Value = 0;
        numOptimalHumHigh.Value = 0;
        numAcceptableHumLow.Value = 0;
    numAcceptableHumHigh.Value = 0;

    if (isMeasurementStarted)
    {
        buttonStartRecv.Text = "Запуск запроса измерений";
        //circularProgressBar1.Visible = false;
        isMeasurementStarted = false;
    }

    socket.Close();

    checkBoxSCD40Connected.Checked = false;
    checkBoxAHT21Connected.Checked = false;
    checkBoxSCD40Initialized.Checked = false;
    checkBoxAHT21Initialized.Checked = false;

    infoCO2.Text = "-";
    tableCO2.BackColor = Color.Silver;
```

```
            infoTemp.Text = "-";
            tableTemp.BackColor = Color.Silver;
            infoHum.Text = "-";
            tableHum.BackColor = Color.Silver;
            Text = "КЛИМАТЧЕК | Не подключен";

            addLog("Сервер принудительно закрыл соединение", 2);

            comboBox1.Enabled = false;

            buttonConnect.Text = "Подключение к серверу";
            buttonStartRecv.Enabled = false;
            isConnected = false;

            _ = MessageBox.Show(
                ex.Message,
                "Ошибка",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error,
                MessageBoxDefaultButton.Button1,
                0
            );

            this.Cursor = Cursors.Default;
            circularProgressBar1.Visible = false;
            buttonConnect.Enabled = true;
        }
        catch (Exception)
        {
            if (counterSensor1 > 1)
                co2Avg /= counterSensor1;
            else if (counterSensor2 < 1)
                co2Avg = 0;
            if (counterSensor2 > 1)
            {
                tempAvg /= counterSensor2;
                humAvg /= counterSensor2;
            }
            else if (counterSensor1 < 1)
                humAvg = -1;
```

```
                    break;
                }
            }

            //labelStatus.Text = "Остановлен";
        }

        public Form1()
        {
            InitializeComponent();

            ShowConsole(_settings.DebugMode);
            comboBox1.SelectedIndex = 0;
            addLog("КЛИМАТЧЕК beta запущено", 0);

            Text = "КЛИМАТЧЕК | Не подключен";
        }

        private void buttonSettings_Click(object sender, EventArgs e)
        {
            addLog("Вызвано окно настроек", 0);
            _ = new Form2(this, _settings).ShowDialog();
            addLog("Окно настроек закрыто", 0);
        }

        private async void buttonConnect_Click(object sender, EventArgs e)
        {
            circularProgressBar1.Visible = true;
            buttonConnect.Enabled = false;

            if (isConnected)
            {
                addLog("Попытка отключения...", 0);
                checkBoxAHT21Recv.Checked = checkBoxSCD40Recv.Checked =
false;

                if (isMeasurementStarted)
                {
                    addLog("Отключение от сервера во время запросов
измерений. Для сохранения результатов рекомендуется сначала останавливать процесс
запроса", 1);
```

```
//threadMutex.WaitOne();
//recvThread.Interrupt();
//recvThread.Join();
//threadMutex.ReleaseMutex();

cts?.Cancel();
if (requestTask != null)
{
    try
    {
        await requestTask; // Ждем завершения
    }
    catch (OperationCanceledException ex)
    {
        addLog(ex.Message, 3);
    }
}

numOptimalCO2.Enabled = false;
numAcceptableCO2.Enabled = false;
numOptimalCO2.Value = 0;
numAcceptableCO2.Value = 0;

numOptimalTempLow.Enabled = false;
numOptimalTempHigh.Enabled = false;
numAcceptableTempLow.Enabled = false;
numAcceptableTempHigh.Enabled = false;
numOptimalTempLow.Value = 0;
numOptimalTempHigh.Value = 0;
numAcceptableTempLow.Value = 0;
numAcceptableTempHigh.Value = 0;

numOptimalHumLow.Enabled = false;
numOptimalHumHigh.Enabled = false;
numAcceptableHumLow.Enabled = false;
numAcceptableHumHigh.Enabled = false;
numOptimalHumLow.Value = 0;
numOptimalHumHigh.Value = 0;
numAcceptableHumLow.Value = 0;
numAcceptableHumHigh.Value = 0;
```

```
                addLog("Успешная остановка", 3);

                buttonStartRecv.Text = "Запуск запроса измерений";
                //circularProgressBar1.Visible = false;
                isMeasurementStarted = false;
            }
            socket.Close();

            checkBoxSCD40Connected.Checked = false;
            checkBoxAHT21Connected.Checked = false;
            checkBoxSCD40Initialized.Checked = false;
            checkBoxAHT21Initialized.Checked = false;

            infoCO2.Text = "-";
            tableCO2.BackColor = Color.Silver;
            infoTemp.Text = "-";
            tableTemp.BackColor = Color.Silver;
            infoHum.Text = "-";
            tableHum.BackColor = Color.Silver;

            addLog("Успешное отключение", 3);
            Text = "КЛИМАТЧЕК | Не подключен";

            comboBox1.Enabled = false;

            buttonConnect.Text = "Подключение к серверу";
            buttonStartRecv.Enabled = false;
            isConnected = false;
        }
        else
        {
            comboBox1.SelectedIndex = 0;
            addLog("Попытка подключения к " + (_settings.UseLocalhost ?
"127.0.0.1" : _settings.ManualAddress) + "...", 0);
            this.Cursor = Cursors.WaitCursor;

            try
            {
                Random random = new Random();
                byte ack = (byte)random.Next();
                byte[] ackBuffer = { ack, (byte)'1' };
```

```
                socket = new Socket(SocketType.Stream,
ProtocolType.Tcp);

                socket.Connect(_settings.UseLocalhost ? "127.0.0.1" :
_settings.ManualAddress, 8080);

                addLog("Успешное подключение, отправка проверочного
байта (" + ack.ToString() + ")", 3);
                _ = socket.Send(ackBuffer);
                _ = socket.Receive(ackBuffer);

                if (ackBuffer[0] == ack)
                    addLog("Ответ от сервера (" + ackBuffer[0].ToString()
+ ") совпадает с проверочным байтом (" + ack.ToString() + "), проверка завершена",
3);
                else
                {
                    addLog("Ответ от сервера (" + ackBuffer[0].ToString()
+ ") не совпадает с проверочным байтом (" + ack.ToString() + "), проверка завершена",
2);

                    //this.Cursor = Cursors.Default;
                    //circularProgressBar1.Visible = false;
                    //buttonConnect.Enabled = true;

                    throw new Exception("Указан некорректный сервер.
Укажите в настройках верный адрес сервера.");
                }
            }
            catch (Exception ex)
            {
                addLog(ex.Message, 2);
                _ = MessageBox.Show(
                    ex.Message,
                    "Ошибка",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error,
                    MessageBoxDefaultButton.Button1,
                    0
                );
                this.Cursor = Cursors.Default;
```

```
                    circularProgressBar1.Visible = false;
                    buttonConnect.Enabled = true;

                    return;
                }

                byte[] buffer = new byte[1024];
                try
                {
                    _ = socket.Receive(buffer);
                    if (!((buffer[0] == 1 || buffer[0] == 0) &&
                        (buffer[1] == 1 || buffer[1] == 0) &&
                        (buffer[2] == 1 || buffer[2] == 0) &&
                        (buffer[3] == 1 || buffer[3] == 0)))
                    {
                        throw new Exception("Получено некорректное состояние
сервера.");
                    }
                }
                catch (Exception ex)
                {
                    addLog(ex.Message, 2);
                    _ = MessageBox.Show(
                        ex.Message,
                        "Ошибка",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error,
                        MessageBoxDefaultButton.Button1,
                        0
                    );
                    this.Cursor = Cursors.Default;
                    circularProgressBar1.Visible = false;
                    buttonConnect.Enabled = true;
                    return;
                }

                ServerState?  state  =  (ServerState?)BytesToObject(buffer,
typeof(ServerState));
                if (state != null)
                {
                    addLog("Получено корректное состояние сервера", 3);
```

```
                            checkBoxSCD40Connected.Checked                          =
state.Value.isSCD40Connected;
                            checkBoxAHT21Connected.Checked                          =
state.Value.isAHT21Connected;
                            checkBoxSCD40Initialized.Checked                        =
state.Value.isSCD40Initialized;
                            checkBoxAHT21Initialized.Checked                        =
state.Value.isAHT21Initialized;
                        }
                        else
                        {
                            addLog("Не удалось распознать ответ сервера, состояния
датчиков будут выставлены по умолчанию", 1);
                            checkBoxSCD40Connected.Checked = false;
                            checkBoxAHT21Connected.Checked = false;
                            checkBoxSCD40Initialized.Checked = false;
                            checkBoxAHT21Initialized.Checked = false;
                        }

                        Text = "КЛИМАТЧЕК | Подключен к " + (_settings.UseLocalhost
== true ? "локальному серверу" : "серверу " + _settings.ManualAddress);

                        this.Cursor = Cursors.Default;
                        buttonConnect.Text = "Отключение от сервера";
                        buttonStartRecv.Enabled = true;
                        isConnected = true;
                    }

                    circularProgressBar1.Visible = false;
                    buttonConnect.Enabled = true;
                }
            }
```

Файл Form2.cs

```
using System.Diagnostics;
using System.Net;
using System.Net.Sockets;
using WinFormsApp1.Properties;

namespace WinFormsApp1
{
    public partial class Form2 : Form
```

```
        {
            private Form1 parentForm;
            private Settings _settings;
            //private List<COMPort> comPorts;

            public Form2(Form1 f, Settings settings)
            {
                parentForm = f;
                _settings = settings;
                InitializeComponent();

                checkBoxLocalhost.Checked = _settings.UseLocalhost;
                numericUpDownDelay.Value = _settings.Interval;
                checkBoxConsole.Checked = _settings.DebugMode;
                textBoxAddress.Text = _settings.ManualAddress;

                textBoxAddress.Enabled = !_settings.UseLocalhost &&
!parentForm.isConnected;
                checkBoxLocalhost.Enabled = !parentForm.isConnected;
                buttonPing.Enabled = !parentForm.isConnected;
                buttonHelp.Enabled = !parentForm.isConnected;
                numericUpDownDelay.Enabled = !parentForm.isMeasurementStarted;
                if (parentForm.isConnected)
                {
                    consoleStatus.ForeColor = Color.Yellow;
                    consoleStatus.Text = "Для изменения настроек подключения " +
(parentForm.isMeasurementStarted ? "и измерения остановите запросы и" : "") + "
отключитесь от сервера";
                }
            }

            private void form2_onClose(object sender, FormClosingEventArgs e)
            {
                _settings.UseLocalhost = checkBoxLocalhost.Checked;
                _settings.Interval = (int)numericUpDownDelay.Value;
                _settings.DebugMode = checkBoxConsole.Checked;
                _settings.ManualAddress = textBoxAddress.Text;

                _settings.Save();
            }
```

```
            private void checkBoxConsole_CheckedChanged(object sender, EventArgs
e)
            {
                parentForm.ShowConsole(checkBoxConsole.Checked);
            }

            private    void    checkBoxLocalhost_CheckedChanged(object    sender,
EventArgs e)
            {
                if (checkBoxLocalhost.Checked)
                    textBoxAddress.Enabled = false;
                else
                    textBoxAddress.Enabled = true;
            }

            private void buttonHelp_Click(object sender, EventArgs e)
            {
                MessageBox.Show(
                    "Как  узнать  IP-адрес  устройства,  на  котором  запущен
сервер?\n" +
                    "1.  Если  сервер  запущен  на  этом  устройстве,  достаточно
выставить флаг \"Подключаться к этому компьютеру\"\n" +
                    "2. Если сервер запущен на другом устройстве:\n" +
                    "  - ;\n" +
                    "  - ;\n" +
                    "  - ;\n" +
                    "  - Обратиться к системному администратору.",
                    "Руководство по выбору IP-адреса",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information,
                    MessageBoxDefaultButton.Button1,
                    0
                );
            }

            private void buttonPing_Click(object sender, EventArgs e)
            {
                parentForm.addLog("Запуск проверки IP-адреса", 0);
                if   (!IPAddress.TryParse(textBoxAddress.Text,   out   IPAddress?
ipAddress))
                {
```

```
                    consoleStatus.ForeColor = Color.Red;
                    consoleStatus.Text = "Неверный IP-адрес";
                    parentForm.addLog("Указан неверный IP-адрес, проверка
завершена", 2);
                    return;
                }

                this.Cursor = Cursors.WaitCursor;
                Socket serverSocket = new Socket(SocketType.Stream,
ProtocolType.Tcp);
                Stopwatch watch = new Stopwatch();
                Random random = new Random();
                byte ack = (byte)random.Next();
                byte[] buffer = { ack, (byte)'0' };
                try
                {
                    watch.Start();
                    serverSocket.Connect(checkBoxLocalhost.Checked ? "127.0.0.1"
: textBoxAddress.Text, 8080);
                    parentForm.addLog("Успешное подключение, отправка
проверочного байта (0x" + ack.ToString() + ")", 3);

                    serverSocket.Send(buffer);
                    serverSocket.Receive(buffer);
                    watch.Stop();

                    if (buffer[0] == ack)
                    {
                        consoleStatus.ForeColor = Color.Green;
                        consoleStatus.Text = "Получен ответ от сервера! ~" +
watch.ElapsedMilliseconds.ToString() + " мс";
                        parentForm.addLog("Ответ от сервера (0x" +
buffer[0].ToString() + ") совпадает с проверочным байтом (0x" + ack.ToString() +
"), проверка завершена", 3);
                    }
                    else
                    {
                        consoleStatus.ForeColor = Color.Red;
                        consoleStatus.Text = "Получен неверный ответ от сервера!
~" + watch.ElapsedMilliseconds.ToString() + " мс";
```

```
                        parentForm.addLog("Ответ от сервера (0x" +
buffer[0].ToString() + ") не совпадает с проверочным байтом (0x" + ack.ToString()
+ "), проверка завершена", 2);
                    }
                }
                catch (Exception ex)
                {
                    watch.Stop();
                    consoleStatus.ForeColor = Color.Red;
                    consoleStatus.Text = ex.Message + " ~" +
watch.ElapsedMilliseconds.ToString() + " мс";
                    parentForm.addLog("Вызвано исключение (см. консоль в
настройках), проверка завершена", 2);

                    serverSocket.Close();
                    this.Cursor = Cursors.Default;
                    return;
                }

                serverSocket.Close();
                this.Cursor = Cursors.Default;
            }
        }
    }
```

Приложение В.2 – Листинг программы сервера

Файл service.h

```
#ifndef service_h
#define service_h

struct Config {
    int maxConnections = 100;
    int delayRetry = 10;
    int delayRestart = 20;
};

void Log(const char* message, int id = 0, int type = 0);
bool Initialisation();
void ReadConfig();

void WINAPI ServiceCtrlHandler(DWORD ctrlCode);

void WINAPI ServiceMain(DWORD argc, LPSTR* argv);
bool StartService(bool* pServiceRunning, bool* pServerRunning);
void StopService();
void StopServiceByEvent();

#endif // service_h
```

Файл server.h

```
#ifndef server_h
#define server_h

struct Client {
    SOCKET socket;
    SOCKADDR_IN address;
    std::thread clientThread;
    bool active;

    Client(SOCKET s, sockaddr_in addr)
        : socket(s), address(addr), active(true) {
    }
};

struct ServerState {
    bool isSCD40Connected = false;
```

```
    bool isSCD40Initialized = false;
    bool isAHT21Connected = false;
    bool isAHT21Initialized = false;
};

struct SensorSCD40Data {
    uint16_t co2 = 0xFFFF;
    uint8_t co2_crc = 0xFF;
    double temperature = 0xFFFFFFFFFFFFFFFF;
    uint8_t temp_crc = 0xFF;
    double humidity = 0xFFFFFFFFFFFFFFFF;
    uint8_t hum_crc = 0xFF;
};

struct SensorAHT21Data {
    uint8_t status = 0xFF;
    double humidity = 0xFFFFFFFFFFFFFFFF;
    double temperature = 0xFFFFFFFFFFFFFFFF;
    uint8_t crc = 0xFF;
};

class SocketManager {
public:
    SOCKET _socket = { 0 };
private:
    int addrSize = sizeof(struct sockaddr_in);
    SOCKADDR_IN addr = { 0 };
    WSADATA wsaData = { 0 };
    bool wsaStarted = false;
    bool socketCreated = false;
    int port = 8080;

public:
    SocketManager();
    ~SocketManager();

    SOCKET* Assign();
    bool Bind();
    bool Listen();
    Client* Accept();
    void Close();
```

```
};

class ComPortManager {
private:
    HANDLE hComPort;
    std::string portName;
    bool isConnected;
    std::mutex comMutex;

    std::string FindCP210xPort();
public:
    ComPortManager() : hComPort(INVALID_HANDLE_VALUE), isConnected(false) {}
    ~ComPortManager();

    bool Connect();
    void Disconnect();
    bool Send(unsigned char command);
    bool Receive(std::vector<unsigned char>& data, size_t bytes);
};

class SCD40Manager {
public:
    SensorSCD40Data state;

    SCD40Manager() {};
    ~SCD40Manager() {};

private:
    void ParseWrite(const std::vector<unsigned char>& data);

public:
    void Foo(bool* running);
};

class AHT21Manager {
public:
    SensorAHT21Data state;

    AHT21Manager() {};
    ~AHT21Manager() {};
```

```
private:
    void ParseWrite(const std::vector<unsigned char>& data);

public:
    void Foo(bool* running);
};

void cls();
void UIFoo(bool* running);
void removeClient(SOCKET clientSocket);
void ClientFoo(SOCKET clientSocket, sockaddr_in clientAddr, bool* running);
bool ServerMain();

bool StartServer(bool* pServiceRunning, bool* pServerRunning, SocketManager*
pSocketManager, int delayRetry, int maxConnections);
void StartServerThread();
void ReceiveRequests();
void StopServer();

#endif // server_h
```

Файл main.cpp

```
char NAME[] = "MeasurerServer";

bool gServerRunning = false;
bool gServiceRunning = false;

SocketManager gSocketManager = SocketManager();
Config gConfig = {};

bool ServerMain() {
    if (!StartServer(&gServiceRunning, &gServerRunning, &gSocketManager,
gConfig.delayRetry, gConfig.maxConnections)) {
        Log("Служба остановлена в процессе запуска сервера", 2, 0);
        return true;
    }

    gServerRunning = true;

    StartServerThread();
    ReceiveRequests();
    StopServer();
```

```
        gServerRunning = false;

        return true;
    }

    void WINAPI ServiceMain(DWORD argc, LPSTR* argv) {
        if (!StartService(&gServiceRunning, &gServerRunning)) {
            return;
        }
        gServiceRunning = true;

        while (gServiceRunning) {
            ServerMain();
            if (gServiceRunning) {
                Log("Сервер завершил работу, но служба не остановлена, будет выполнен повторный запуск", 3, 2);
                //gSocketManager.~SocketManager();
                //gSocketManager = SocketManager();
                for (int i = 0; i < gConfig.delayRestart * 10; i++) {
                    Sleep(100);
                    if (!gServiceRunning)
                        break;
                }
            }
        }

        StopService();
        gServiceRunning = false;
        gServerRunning = false;
    }

    int main(int argc, char* argv[]) {
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);

        if (!Initialisation()) {
            Log("Не удалось инициализировать пакетные файлы в каталоге приложения. Требуется перезапуск службы.", 3, 3);
            return 1;
        }
```

```
        ReadConfig();

        if (argc > 1) {
            if (!strcmp(argv[1], "debug")) {
                gServiceRunning = true;
                while (ServerMain()) {
                    Log("Сервер завершил работу, но служба не остановлена, будет
выполнен повторный запуск", 3, 2);

                    for (int i = 0; i < gConfig.delayRestart * 10; i++) {
                        Sleep(100);
                        if (!gServiceRunning)
                            break;
                    }
                }
            }
            return 0;
        }
        else {
            SERVICE_TABLE_ENTRYA serviceTable[] = {
                {NAME, (LPSERVICE_MAIN_FUNCTIONA)ServiceMain},
                {NULL, NULL}
            };

            if (!StartServiceCtrlDispatcherA(serviceTable)) {
                int index = GetLastError();
                Log("Не удалось запустить обработчик службы!", index, 3);
                return index;
            }
            return 0;
        }
    }

    void WINAPI ServiceCtrlHandler(DWORD ctrlCode) {
        switch (ctrlCode) {
        case SERVICE_CONTROL_STOP:
            gSocketManager.Close();
            gServerRunning = false;
            StopServiceByEvent();
            break;
```

```
        default:
            break;
        }
    }

    bool Initialisation() {
        bool isReady = true;
        char pathBatStart[] = "Установить сервер.bat",
            pathBatStop[] = "Удалить сервер.bat",
            pathBatStatus[] = "Получить записи сервера.bat",
            pathBatRestart[] = "Перезапустить сервер.bat",
            pathConfig[] = "config.ini";

        char docBatStart[] = "@echo off\n"
            "chcp 1251 >nul\n"
            "sc create MeasurerServer binPath= \"%%~dp0CCS.exe\" start= delayed-
auto DisplayName= \"Measurer Server\"\n"
            "if %%errorLevel%% neq 0 (\n"
            "pause\n"
            "exit /b 1\n)\n"
            "sc description MeasurerServer \"Служба сервера ПАК измерения и
контроля микроклимата\"\n"
            "sc start MeasurerServer\n"
            "if %%errorLevel%% neq 0 (\n"
            "pause\n"
            "exit /b 1\n)\n"
            "echo Сервер установлен и будет запускаться автоматически при каждом
старте системы.\n"
            "pause",
            docBatStatus[] = "@echo off\n"
            "chcp 1251 >nul\n"
            "echo Состояние сервера:\n"
            "sc query MeasurerServer\n"
            "echo Последние 10 записей сервера:\n"
            "wevtutil                    qe                    Application
/q:\"*[System[Provider[@Name=\'MeasurerServer\']]]\" /f:text /c:10 /rd:true\n(\n"
            "echo Состояние сервера:\n"
            "sc query MeasurerServer\n"
            "echo Последние 10 записей сервера:\n"
            "wevtutil                    qe                    Application
/q:\"*[System[Provider[@Name=\'MeasurerServer\']]]\" /f:text /c:10 /rd:true\n"
```

```
") > \"%%~dp0states.log\"\n"
"echo Записи сервера скопированы в файл states.log\n"
"pause",
docBatRestart[] = "@echo off\n"
"chcp 1251 >nul\n"
"sc stop MeasurerServer\n"
"if %%errorLevel%% equ 1062 (\n"
"goto deli\n)\n"
"if %%errorLevel%% neq 0 (\n"
"pause\n"
"exit /b 1\n)\n"
"timeout /t 3\n"
":deli\n"
"sc delete MeasurerServer\n"
"if %%errorLevel%% neq 0 (\n"
"pause\n"
"exit /b 1\n)\n"
"sc create MeasurerServer binPath= \"%%~dp0CCS.exe\" start= auto
DisplayName= \"Measurer Server\"\n"
"if %%errorLevel%% neq 0 (\n"
"pause\n"
"exit /b 1\n)\n"
"sc description MeasurerServer \"Служба сервера ПАК измерения и
контроля микроклимата\"\n"
"sc start MeasurerServer\n"
"if %%errorLevel%% neq 0 (\n"
"pause\n"
"exit /b 1\n)\n"
"echo Сервер перезапущен успешно.\n"
"pause",
docBatStop[] = "@echo off\n"
"chcp 1251 >nul\n"
"sc stop MeasurerServer\n"
"if %%errorLevel%% equ 1062 (\n"
"goto deli\n)\n"
"if %%errorLevel%% neq 0 (\n"
"pause\n"
"exit /b 1\n)\n"
"timeout /t 3\n"
":deli\n"
"sc delete MeasurerServer\n"
```

```
            "if %%errorLevel%% neq 0 (\n"
            "pause\n"
            "exit /b 1\n)\n"
            "echo Служба удалена.\n"
            "pause",
            docConfig[] = "[Settings]\n"
            "MaxConnections=100\n"
            "SecDelayRetry=10\n"
            "SecDelayRestart=20";

        FILE* configFile = fopen(pathConfig, "r+");
        if (!configFile) {
            configFile = fopen(pathConfig, "w");
            fprintf(configFile, docConfig);
            std::cout << "Отсутствует файл конфигурации сервера.\n Файл конфигурации создан, работа программы будет завершена.\n";

            fclose(configFile);
            isReady = false;
        }

        fclose(configFile);

        FILE* batStart = fopen(pathBatStart, "r+");
        if (!batStart) {
            batStart = fopen(pathBatStart, "w");
            fprintf(batStart, docBatStart);
            std::cout << "Отсутствует пакетный файл установки сервера.\n Пакетный файл " << pathBatStart << " создан, работа программы будет завершена.\n";

            fclose(batStart);
            isReady = false;
        }

        FILE* otherBat = fopen(pathBatStop, "r+");
        if (!otherBat) {
            otherBat = fopen(pathBatStop, "w");
            fprintf(otherBat, docBatStop);

            std::cout << "Отсутствует пакетный файл удаления сервера.\n Пакетный файл " << pathBatStop << " создан.\n";
```

```
        }

        otherBat = fopen(pathBatStatus, "r+");
        if (!otherBat) {
            otherBat = fopen(pathBatStatus, "w");
            fprintf(otherBat, docBatStatus);

            std::cout << "Отсутствует пакетный файл получения записей сервера.\n
Пакетный файл " << pathBatStatus << " создан.\n";
        }

        otherBat = fopen(pathBatRestart, "r+");
        if (!otherBat) {
            otherBat = fopen(pathBatRestart, "w");
            fprintf(otherBat, docBatRestart);

            std::cout << "Отсутствует пакетный файл перезапуска сервера.\n
Пакетный файл " << pathBatRestart << " создан.\n";
        }

        if (!isReady) {
            std::cout << "Нажмите любую клавишу для выхода из программы";
            _getch();
        }

        return isReady;
    }

    void ReadConfig() {
        int test = 0;
        char configPath[] = ".\\config.ini";
        gConfig.maxConnections = GetPrivateProfileIntA("Settings",
"MaxConnections", 100, configPath);
        gConfig.delayRetry = GetPrivateProfileIntA("Settings", "SecDelayRetry",
10, configPath);
        gConfig.delayRestart = GetPrivateProfileIntA("Settings",
"SecDelayRestart", 20, configPath);
        test += 100;
    }
```

Файл service.cpp

```
    char _NAME[] = "MeasurerServer";
```

```
bool* _gServiceRunning = NULL;
bool* __gServerRunning = NULL;

SERVICE_STATUS gServiceStatus = { 0 };
SERVICE_STATUS_HANDLE gStatusHandle = NULL;
HANDLE gServiceStopEvent = INVALID_HANDLE_VALUE;

void Log(const char* message, int id, int type) {
    char buffer[1024] = { 0 };
    sprintf(buffer, "EventCreate /T %s /ID %d /L APPLICATION /SO %s /D
\"%s\"",
        !type ? "INFORMATION" : type == 1 ? "SUCCESS" : type == 2 ? "WARNING"
: "ERROR",
        id, _NAME, message);

    system(buffer);
}

bool StartService(bool* pServiceRunning, bool* pServerRunning) {
    if (!pServiceRunning || !pServerRunning)
        return false;
    _gServiceRunning = pServiceRunning;
    __gServerRunning = pServerRunning;

    gStatusHandle = RegisterServiceCtrlHandlerA(_NAME, ServiceCtrlHandler);
    if (!gStatusHandle) {
        Log("Ошибка запуска службы!", GetLastError(), 3);
        return false;
    }

    gServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    gServiceStatus.dwCurrentState = SERVICE_START_PENDING;

    SetServiceStatus(gStatusHandle, &gServiceStatus);
    Log("Служба запускается...", 1, 0);

    gServiceStopEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (!gServiceStopEvent) {
        gServiceStatus.dwCurrentState = SERVICE_STOPPED;
        gServiceStatus.dwWin32ExitCode = GetLastError();
```

```
            SetServiceStatus(gStatusHandle, &gServiceStatus);
            Log("Ошибка запуска службы!", GetLastError(), 3);
            return false;
        }

        gServiceStatus.dwCurrentState = SERVICE_RUNNING;
        gServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
        gServiceStatus.dwWin32ExitCode = 0;
        gServiceStatus.dwCheckPoint = 0;
        SetServiceStatus(gStatusHandle, &gServiceStatus);
        *_gServiceRunning = true;
        Log("Служба запущена успешно", 1, 1);

        return true;
    }

    void StopServiceByEvent() {
        if (gServiceStatus.dwCurrentState != SERVICE_RUNNING) {
            Log("Получен сигнал остановки службы, но служба уже остановлена!",
1, 2);
            *_gServiceRunning = false;
            return;
        }

        *_gServiceRunning = false;
        //Log("Служба остановлена успешно", 1, 1);
        gServiceStatus.dwControlsAccepted = 0;
        gServiceStatus.dwCurrentState = SERVICE_STOP_PENDING;
        gServiceStatus.dwWin32ExitCode = 0;
        gServiceStatus.dwCheckPoint = 4;

        SetServiceStatus(gStatusHandle, &gServiceStatus);
        SetEvent(gServiceStopEvent);
        return;
    }

    void StopService() {
        *_gServiceRunning = false;
        Log("Служба остановлена успешно", 1, 1);
        CloseHandle(gServiceStopEvent);
```

```
        gServiceStatus.dwCurrentState = SERVICE_STOPPED;
        gServiceStatus.dwWin32ExitCode = 0;
        gServiceStatus.dwCheckPoint = 3;

        SetServiceStatus(gStatusHandle, &gServiceStatus);
    }
```

Файл server.cpp

```
    SocketManager* socketManager = NULL;
    ComPortManager comPortManager = {};
    SCD40Manager* sensorSCD40Manager = NULL;
    AHT21Manager* sensorAHT21Manager = NULL;

    ServerState serverState = ServerState();
    SensorSCD40Data sensor1State = {};
    SensorAHT21Data sensor2State = {};

    SOCKET* serverSocket = NULL;
    std::thread uiThread;
    std::thread scd40Thread;
    std::thread aht21Thread;

    std::mutex clientsMutex;
    std::mutex servstateMutex;
    std::mutex sensor1Mutex;
    std::mutex sensor2Mutex;
    std::mutex i2cMutex;

    std::vector<Client*> clients;

    bool* _gServerRunning = NULL;
    bool* __gServiceRunning = NULL;
    int gMaxConnections = 0;

    bool StartServer(bool* pServiceRunning, bool* pServerRunning, SocketManager*
pSocketManager, int delayRetry, int maxConnections) {
        if (!pServiceRunning || !pServerRunning || !pSocketManager)
            return false;
        _gServerRunning = pServerRunning;
        __gServiceRunning = pServiceRunning;
        socketManager = pSocketManager;
        gMaxConnections = maxConnections;
```

```
        while (!comPortManager.Connect()) {
            Log("Возникла ошибка при запуске сервера, будет выполнена повторная
попытка", 2, 2);
            for (int i = 0; i < delayRetry * 10; i++) {  // 50 * 100 = 5000 мс
                Sleep(100);
                if (!*__gServiceRunning)
                    return false;
            }
            //Sleep(5000);
        }

        while (NULL == (serverSocket = socketManager->Assign())) {
            Log("Не удалось создать сокет", 2, 3);
            Log("Возникла ошибка при запуске сервера, будет выполнена повторная
попытка", 2, 2);
            for (int i = 0; i < delayRetry * 10; i++) {  // 50 * 100 = 5000 мс
                Sleep(100);
                if (!*__gServiceRunning)
                    return false;
            }
            //Sleep(5000);
        }
        //Log("Сокет создан успешно!", 2, 1);
        Sleep(100);

        while (!socketManager->Bind()) {
            Log("Не удалось захватить порт", 2, 3);
            Log("Возникла ошибка при запуске сервера, будет выполнена повторная
попытка", 2, 2);
            for (int i = 0; i < delayRetry * 10; i++) {  // 50 * 100 = 5000 мс
                Sleep(100);
                if (!*__gServiceRunning)
                    return false;
            }
            //Sleep(5000);
        }
        //Log("Порт успешно захвачен сокетом", 1, 1);
        Sleep(100);

        while (!socketManager->Listen()) {
```

```
            Log("Не удалось запустить прослушивание сокета", 2, 3);
            Log("Возникла ошибка при запуске сервера, будет выполнена повторная
попытка", 2, 2);
            for (int i = 0; i < delayRetry * 10; i++) {  // 50 * 100 = 5000 мс
                Sleep(100);
                if (!*__gServiceRunning)
                    return false;
            }
            //Sleep(5000);
        }

        if (!*__gServiceRunning)
            return false;
        //Log("Порт прослушивается сокетом", 2, 1);
        Sleep(100);

        return true;
    }

    void StartServerThread() {
        sensorSCD40Manager = new SCD40Manager();
        scd40Thread             =             std::thread([]{sensorSCD40Manager-
>Foo(_gServerRunning);});
        sensorAHT21Manager = new AHT21Manager();
        aht21Thread             =             std::thread([]{sensorAHT21Manager-
>Foo(_gServerRunning);});
        //Log("Потоки опроса датчиков запущены");

        //uiThread = std::thread(UIFoo, running);

        Log("Сервер запущен успешно!", 2, 1);
    }

    void ReceiveRequests() {
        char authBuffer[2] = { 0 };

        while (*_gServerRunning) {
            if (clients.size() >= gMaxConnections) {
                Sleep(0);
                continue;
            }
```

```
            Client* currentClient = socketManager->Accept();
            //  если  сокет  сервера  был  закрыт,  прекратить  принимать  новые
подключения
            if (currentClient == NULL) {
                {
                    std::lock_guard<std::mutex> lock(servstateMutex);

                    socketManager->Close();
                    *_gServerRunning = false;
                }
                break;
            }
            if (NULL == recv(currentClient->socket, authBuffer, 2, 0)) {
                {
                    std::lock_guard<std::mutex> lock(servstateMutex);

                    socketManager->Close();
                    *_gServerRunning = false;
                }
                break;
            }
            if (authBuffer[0] == NULL && authBuffer[1] == NULL) {
                {
                    std::lock_guard<std::mutex> lock(servstateMutex);

                    socketManager->Close();
                    *_gServerRunning = false;
                }
                break;
            }
            // запрос состояния датчиков
            if (authBuffer[1] == '0') {
                send(currentClient->socket, authBuffer, 2, 0);
                //delete [] &(currentClient->address);
                delete currentClient;
            }
            // периодические запросы измерения
            else if (authBuffer[1] == '1') {
                send(currentClient->socket, authBuffer, 2, 0);
```

```
                send(currentClient->socket,                (char*)&serverState,
sizeof(ServerState), 0);
                {
                    std::lock_guard<std::mutex> lock(clientsMutex);

                    currentClient->clientThread    =    std::thread(ClientFoo,
currentClient->socket, currentClient->address, _gServerRunning);
                    clients.push_back(currentClient);
                }
            }
        }
    }

    void StopServer() {
        scd40Thread.join();
        aht21Thread.join();

        for (auto* client : clients) {
            closesocket(client->socket);
            client->clientThread.join();
            delete& (client->address);
            delete client;
        }

        delete sensorAHT21Manager;
        sensorAHT21Manager = NULL;
        delete sensorSCD40Manager;
        sensorSCD40Manager = NULL;
        socketManager->Close();

        //uiThread.join();
        //Log("Сервер приостановлен", 2, 0);
    }

    SocketManager::SocketManager() {
        wsaStarted = WSAStartup(MAKEWORD(2, 2), &wsaData) == NULL ? true : false;
        int test = WSAGetLastError();
    }
    SocketManager::~SocketManager() {
        if (socketCreated) {
            socketCreated = false;
```

```
            closesocket(_socket);
            int test = WSAGetLastError();
        }
        if (wsaStarted) {
            WSACleanup();
            int test = WSAGetLastError();
            wsaStarted = false;
        }
    }

    SOCKET* SocketManager::Assign() {
        if (!wsaStarted)
            return NULL;
        _socket = socket(AF_INET, SOCK_STREAM, 0);
        int test = WSAGetLastError();
        if (_socket == INVALID_SOCKET) {
            //WSACleanup();
            return NULL;
        }
        addr.sin_addr.s_addr = INADDR_ANY;
        addr.sin_family = AF_INET;
        addr.sin_port = htons(port);
        socketCreated = true;

        return &_socket;
    }

    bool SocketManager::Bind() {
        if (!wsaStarted)
            return false;

        if (SOCKET_ERROR == bind(_socket, (SOCKADDR*)&addr, sizeof(addr))) {
            //closesocket(_socket);
            //socketCreated = false;
            //WSACleanup();
            return false;
        }

        return true;
    }
```

```
bool SocketManager::Listen() {
    if (!wsaStarted)
        return false;
    if (SOCKET_ERROR == listen(_socket, 0)) {
        //closesocket(_socket);
        //socketCreated = false;
        //WSACleanup();
        return false;
    }

    return true;
}

Client* SocketManager::Accept() {
    if (!wsaStarted)
        return NULL;
    SOCKADDR_IN* clientAddr = new SOCKADDR_IN();
    SOCKET clientSocket = accept(_socket, (struct sockaddr*)clientAddr,
&addrSize);
    Client* client = new Client(clientSocket, *clientAddr);

    if (!client->socket) {
        delete clientAddr;
        delete client;

        return NULL;
    }

    return client;
}

void SocketManager::Close() {
    if (socketCreated) {
        socketCreated = false;
        closesocket(_socket);
    }
    //if (wsaStarted) {
    //    WSACleanup();
    //    wsaStarted = false;
    //}
}
```

Приложение В.3 – Листинг модулей цифровой схемы ПЛИС

Файл i2c_master.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity i2c_master is
    generic (
        -- Частота системного синхросигнала
        CLK_FREQUENCY : natural := 50*1000000;
        -- Частота синхросигнала I2C
        I2C_FREQUENCY : natural := 400*1000
    );
    port (
        -- Синхросигнал
        clk       : in     std_logic;
        -- Инвертированный сброс
        reset_n   : in     std_logic;

        -- Линия последовательных данных I2C
        sda       : inout  std_logic;
        -- Линия синхронизации I2C
        scl       : inout  std_logic;

        -- Флаг запуска передачи
        i2c_start : in     std_logic;
        -- Флаг процесса передачи
        i2c_busy  : out    std_logic;
        -- Флаг окончания передачи
        i2c_done  : out    std_logic;
        -- Флаг ошибки подтверждения
        ack_error : buffer std_logic;
        -- Флаг ожидания
        i2c_stretch : in std_logic;

        -- Адрес ведомого
        address  : in     std_logic_vector(6 downto 0);
        -- Флаг типа операции: чтение/запись
        rw       : in     std_logic;
        -- Данные для записи
```

```
            data_wr  : in     std_logic_vector(7 downto 0);
            -- Полученные данные
            data_rd : out    std_logic_vector(7 downto 0)
        );
    end i2c_master;

    architecture rtl of i2c_master is
        -- Делитель для внутренней синхронизации (scl)
        constant divider     :  natural := (CLK_FREQUENCY/I2C_FREQUENCY)/4;
        -- Состояния ведущего
        type machine is (READY, INIT_FIRST_BIT, SEND_HEADER, SLV_ACK1, WR, RD,
SLV_ACK2, MSTR_ACK, STOP);
        -- Текущее состояние ведущего
        signal master_state  : machine;
        -- Внутренний синхросигнал данных
        signal data_clk      : std_logic;
        -- Предыдущее состояние внутреннего синхросигнала данных
        signal data_clk_prev : std_logic;
        -- Внутренний синхросигнал scl (грубо говоря, его внутреннее значение)
        signal scl_clk       : std_logic;
        -- Флаг разрешения ведущему удержания scl
        signal scl_ena       : std_logic := '0';
        -- Внутренний сигнал sda
        signal sda_int       : std_logic := '1';
        -- Флаг разрешения sda (его внутреннее значение, во всех случаях кроме
INIT_FIRST_BIT и STOP равен sda_int)
        signal sda_ena_n     : std_logic;
        -- Заголовок информации - содержит адрес ведомого и тип операции
        signal i2c_header    : std_logic_vector(7 downto 0);
        -- Буфер данных для отправки ведомому
        signal data_tx       : std_logic_vector(7 downto 0);
        -- Буфер данных для получения от ведомого
        signal data_rx       : std_logic_vector(7 downto 0);
        -- Счетчик разрядов данных
        signal bit_cnt       : integer range 0 to 7 := 7;
        -- Флаг удержания scl ведомым
        signal stretch       : std_logic := '0';                --identifies if
slave is stretching scl

    begin
```

```
        -- Синхронизация внутреннего сигнала данных(data_clk) и внутреннего
синхросигнала I2C(scl)
        process(clk)
            -- Счетчик для генерации синхросигналов
            variable count : natural range 0 to divider * 4;
        begin
            if clk'event and clk = '1' then
                -- Сбросить счетчик и флаг удержания scl ведомым
                if reset_n = '0' then
                    stretch <= '0';
                    count := 0;
                else
                    -- Сохранить прошлое состояние сигнала
                    data_clk_prev <= data_clk;
                    -- Организация цикла внутренней синхронизации
                    if count = (divider * 4 - 1) then
                        -- Конец цикла, счетчик обнуляется
                        count := 0;
                    else
                        -- Цикл в процессе. Если scl не удерживается ведомым -
увеличивать счетчик
                        if stretch = '0' then
                            count := count + 1;
                        end if;
                    end if;

                    -- Реализация цикла внутренней синхронизации - состоит из 4-
ех частей
                    case count is
                        -- В первой четверти цикла удерживать scl, сброс флага
удержания
                        when 0 to divider - 1 =>
                            scl_clk <= '0';
                            data_clk <= '0';
                            stretch <= '0';
                        -- Во второй четверти цикла установить высокий уровень
внутреннего синхросигнала данных
                        --  по переднему фронту data_clk выполняется удерживание
sda
                        when divider to divider * 2 - 1 =>
                            scl_clk <= '0';
```

```
                            data_clk <= '1';
                            stretch <= '0';
                        -- В третьей четверти цикла освободить scl для ведомого
                        --  ведомый считывает данные с sda и может удерживать scl
                        when divider * 2 to divider * 3 - 1 =>
                            scl_clk <= '1';
                            data_clk <= '1';
                            -- Если ведомый удерживает scl, установить флаг
удержания. Если иначе, то сбросить флаг
                            if (scl = '0') then
                                stretch <= '1';
                            else
                                stretch <= '0';
                            end if;
                        -- В четвертой четверти цикла установить низкий уровень
внутреннего синхросигнала данных
                        --  по заднему фронту data_clk выполняется чтение данных
с SDA, который удерживается ведомым
                        when others => --(divider * 3 to divider * 4 - 1)
                            scl_clk <= '1';
                            data_clk <= '0';
                            -- Если ведомый удерживает scl, установить флаг
удержания. Если иначе, то сбросить флаг
                            if (scl = '0') then
                                stretch <= '1';
                            else
                                stretch <= '0';
                            end if;
                    end case;
                end if;
            end if;
        end process;

        -- Чтение и запись данных (взаимодействие с sda)
        process(clk) begin
            if clk'event and clk = '1' then
                -- Сброс
                if reset_n = '0' then
                    -- Начальное состояние ведущего
                    master_state <= READY;
                    -- Во время сброса модуль занят
```

```
                    i2c_busy <= '1';
                    -- Сбросить флаг разрешения ведущему удержания scl
                    scl_ena <= '0';
                    -- Установить внутренний сигнал sda
                    sda_int <= '1';
                    -- Сбросить флаг ошибки подтверждения
                    ack_error <= '0';
                    -- Установить флаг окончания передачи
                    i2c_done <= '1';
                    -- Восстановить счетчик разрядов данных
                    bit_cnt <= 7;
                    -- Очистить буфер чтения данных
                    data_rd <= (others => '0');
                else
                    case master_state is
                        -- Модуль ожидает запрос на передачу
                        when READY =>
                            -- Передний фронт внутреннего синхросигнала данных
                            if data_clk = '1' and data_clk_prev = '0' then
                                -- Если получен запрос на передачу
                                if i2c_start = '1' then
                                    -- Установить флаг работы модуля
                                    i2c_busy <= '1';
                                    -- Сбросить флаг окончания работы
                                    i2c_done <= '0';
                                    -- Сформировать заголовочный байт из адреса
ведомого и типа операции
                                    i2c_header <= address & rw;
                                    -- Собрать данные для отправки ведомому
                                    data_tx <= data_wr;
                                    --  Перейти  к  инициализации  первого  бита
данных
                                    master_state <= INIT_FIRST_BIT;
                                -- Если запроса нет - продолжать ожидание
                                else
                                    i2c_busy <= '0';
                                    i2c_done <= '1';
                                    master_state <= READY;
                                end if;
                            end if;
```

```
                        -- Модуль инициализирует первый бит передаваемых данных
                        when INIT_FIRST_BIT =>
                            -- Передний фронт внутреннего синхросигнала данных
                            if data_clk = '1' and data_clk_prev = '0' then
                                -- Установить флаг работы модуля, если передача
продолжается
                                i2c_busy <= '1';
                                -- Инициализировать первый бит заголовка перед
началом передачи
                                sda_int <= i2c_header(bit_cnt);
                                -- Начать отправку заголовка (адрес ведомого +
тип операции)
                                master_state <= SEND_HEADER;
                            -- Задний фронт внутреннего синхросигнала данных
                            elsif data_clk = '0' and data_clk_prev = '1' then
                                -- Если это новая передача, то запустить её
                                if scl_ena = '0' then
                                    -- Разрешить ведущему удерживать scl
                                    scl_ena <= '1';
                                    --  Сбросить  флаг  ошибки  подтверждения
ведомого
                                    ack_error <= '0';
                                end if;
                            end if;

                        -- Модуль отправляет заголовок (адрес ведомого + тип
операции)
                        when SEND_HEADER =>
                            -- Передний фронт внутреннего синхросигнала данных
                            if data_clk = '1' and data_clk_prev = '0' then
                                -- Если счетчик разрядов данных достиг 0, то
передача данных выполнена
                                if bit_cnt = 0 then
                                    -- Освободить sda для ведомого, чтобы он
вернул подтверждение
                                    sda_int <= '1';
                                    -- Восстановить счетчик разрядов данных
                                    bit_cnt <= 7;
                                    -- Ожидать подтверждение от ведомого
                                    master_state <= SLV_ACK1;
```

```
                                -- Если счетчик разрядов не равен 0, продолжать
передачу
                                else
                                    -- Уменьшить счетчик разрядов
                                    bit_cnt <= bit_cnt - 1;
                                    -- Записать следующий разряд
                                    sda_int <= i2c_header(bit_cnt-1);
                                    -- Продолжить цикл отправки
                                    master_state <= SEND_HEADER;
                                end if;
                            end if;

                        -- Модуль принимает первое подтверждение от ведомого
(после отправки заголовочного байта)
                        when SLV_ACK1 =>
                            -- Передний фронт внутреннего синхросигнала данных
                            if data_clk = '1' and data_clk_prev = '0' then
                                -- В младшем разряде заголовка записан тип
операции
                                --  если 0 - операция записи
                                --  если 1 - операция чтения
                                -- Если операция - запись
                                if i2c_header(0) = '0' then
                                    -- Инициализировать первый разряд данных
                                    sda_int <= data_tx(bit_cnt);
                                    -- Начать операцию записи
                                    master_state <= WR;
                                -- Если операция - чтение
                                else
                                    -- Освободить sda для ведомого, чтобы он смог
передать ведущему данные
                                    sda_int <= '1';
                                    -- Начать операцию чтения
                                    master_state <= RD;                    --go
to read byte
                                end if;
                            -- Задний фронт внутреннего синхросигнала данных
                            elsif data_clk = '0' and data_clk_prev = '1' then
                                -- Если подтверждение от ведомого не получено -
ошибка!
                                if (sda /= '0' or ack_error = '1') then
```

```
                                    -- Установить флаг ошибки подтверждения от
ведомого
                                    ack_error <= '1';
                                    -- !добавить обработку ошибки!
                                end if;
                            end if;

    end rtl;
```

Файл SCD40Driver.vhd

```
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;

    entity SCD40Driver is
        generic (
            SCD40_DELAY : natural
        );
        port (
            clk   : in STD_LOGIC;
            reset : in STD_LOGIC;

            i2c_done         : in  STD_LOGIC;
            i2c_busy         : in  STD_LOGIC;
            i2c_start        : out STD_LOGIC := '0';
            i2c_error        : in  STD_LOGIC;
            i2c_headSCD40     : out STD_LOGIC_VECTOR(7 downto 0) := (others =>
'0');
            i2c_rd           : in  STD_LOGIC_VECTOR(7 downto 0);
            i2c_wr             : out STD_LOGIC_VECTOR(7 downto 0) := (others =>
'0');
            i2c_stretchSCD40 : out STD_LOGIC := '0';

            startSCD40   : in  STD_LOGIC;
            doneSCD40    : out STD_LOGIC := '0';
            commandSCD40 : in STD_LOGIC_VECTOR(7 downto 0);
            resultSCD40   : out STD_LOGIC_VECTOR(9*8-1 downto 0) := (others =>
'0')
        );
    end SCD40Driver;

    architecture Behavioral of SCD40Driver is
        type COMMAND_SEQUENCE is array (0 to 1) of STD_LOGIC_VECTOR(7 downto 0);
```

```
        constant address : STD_LOGIC_VECTOR(7 downto 0) := x"62";
        constant start_periodic_measurement : COMMAND_SEQUENCE := (x"21", x"B1");
        constant read_measurement : COMMAND_SEQUENCE := (x"EC", x"05");
        constant stop_periodic_measurement  : COMMAND_SEQUENCE := (x"3F", x"86");

        signal command_s  : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
        signal result_s   : STD_LOGIC_VECTOR(9*8-1 downto 0) := (others => '0');
        signal counter  : natural range 0 to 9*8-1 := 9*8-1;
        signal i2c_head_s : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
        signal i2c_wr_s   : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
        signal counter_timer : natural range 0 to SCD40_DELAY := 0;

        type states is (IDLE, COMMAND_STARTM0, COMMAND_STARTM, COMMAND_READM0,
WAIT_EXECUTION0, WAIT_EXECUTION1, WAIT_EXECUTION, COMMAND_READM1, COMMAND_READM20,
COMMAND_READM21, COMMAND_READM22, COMMAND_READM2, COMMAND_STOPM0, COMMAND_STOPM,
STOP0, STOP);
        signal state  : states := IDLE;
    begin
        resultSCD40 <= result_s;
        i2c_headSCD40 <= i2c_head_s;
        i2c_wr <= i2c_wr_s;

        process(clk) begin
            if clk'event and clk = '1' then
                if reset = '1' then
                    doneSCD40 <= '0';
                    command_s <= (others => '0');
                    i2c_head_s <= (others => '0');
                    result_s <= (others => '0');
                    i2c_head_s <= (others => '0');
                    i2c_wr_s <= (others => '0');
                    i2c_start <= '0';
                    counter <= 9*8-1;
                    counter_timer <= 0;
                    state <= IDLE;
                else
                    case state is
                        when IDLE =>
                            doneSCD40 <= '0';
                            if startSCD40 = '1' then
                                result_s <= (others => '0');
```

```
                command_s <= commandSCD40;
                counter <= 9*8-1;
                counter_timer <= 0;

                -- Команда запуска периодических измерений
                if commandSCD40(6 downto 5) = "01" then
                    i2c_head_s <= address(6 downto 0) & '0';
                    i2c_wr_s <= start_periodic_measurement(0);
                    i2c_start <= '1';
                    state <= COMMAND_STARTM0;
                    result_s <= x"A1111111111111111";
                -- Команда запроса результатов измерения
                elsif commandSCD40(6 downto 5) = "10" then
                    i2c_head_s <= address(6 downto 0) & '0';
                    i2c_wr_s <= read_measurement(0);
                    i2c_start <= '1';
                    state <= COMMAND_READM0;
                -- Команда остановки периодических измерений
                elsif commandSCD40(6 downto 5) = "11" then
                    i2c_head_s <= address(6 downto 0) & '0';
                    i2c_wr_s <= stop_periodic_measurement(0);
                    i2c_start <= '1';
                    state <= COMMAND_STOPM0;
                    result_s <= x"A33333333333333333";
                else
                    result_s <= x"A00000000000000000";
                    state <= STOP;
                end if;
            end if;

        -- Ожидаем когда модуль i2c начнет работу
        when COMMAND_STARTM0 =>
            if i2c_busy = '1' then
                state <= COMMAND_STARTM;
            end if;

        -- Отправка команды начала периодических измерений
        when COMMAND_STARTM =>
            if i2c_busy = '1' then
                i2c_wr_s <= start_periodic_measurement(1);
            else
```

```
                i2c_start <= '0';
                state <= STOP0;
            end if;

        --! Чтение данных с датчика
        --  Ожидание
        when COMMAND_READM0 =>
            if i2c_busy = '1' then
                state <= COMMAND_READM1;
            end if;
        --  Отправка команды чтение результата измерений
        when COMMAND_READM1 =>
            if i2c_busy = '1' then
                i2c_wr_s <= read_measurement(1);
            else
                i2c_stretchSCD40 <= '1';
                i2c_head_s <= address(6 downto 0) & '1';
                state <= WAIT_EXECUTION0;
            end if;

        --  Ожидаем когда модуль i2c начнет отправку последней
части команды
        when WAIT_EXECUTION0 =>
            if i2c_busy = '1' then
                state <= WAIT_EXECUTION1;
            end if;

        --  Ожидаем когда модуль i2c закончит отправку
        when WAIT_EXECUTION1 =>
            if i2c_busy = '0' then
                state <= WAIT_EXECUTION;
            end if;

        --  Задержка
        when WAIT_EXECUTION =>
            if counter_timer = SCD40_DELAY then
                i2c_stretchSCD40 <= '0';
                state <= COMMAND_READM20;
            else
                counter_timer <= counter_timer + 1;
            end if;
```

```
                        --  Ожидаем когда модуль i2c начнет отправку заголовка
                        when COMMAND_READM20 =>
                            if i2c_busy = '1' then
                                state <= COMMAND_READM21;
                            end if;

                        --  Ожидаем когда модуль i2c закончит чтение байта
                        when COMMAND_READM21 =>
                            if i2c_busy = '0' then
                                state <= COMMAND_READM22;
                            end if;

                        --  Ожидаем пока модуль i2c начнет чтение следующего байта
                        when COMMAND_READM22 =>
                            if i2c_busy = '1' then
                                result_s(counter downto counter - 7) <= i2c_rd;
                                -- Если будет прочитан последний байт, то заранее
сбросить флаг запуска модуля i2c
                                if counter = 7 then
                                    i2c_start <= '0';
                                end if;
                                state <= COMMAND_READM2;
                            end if;

                        --  Чтение поступающей с модуля i2c информации
                        when COMMAND_READM2 =>
                            -- read data
                            if i2c_busy = '0' then
                                --result_s(counter downto counter - 7) <= i2c_rd;

                                if counter = 7 then
                                    state <= STOP;
                                else
                                    counter <= counter - 8;
                                    state <= COMMAND_READM21; --!
                                end if;
                            end if;

                        --! Остановка периодических измерений
                        --  Ожидание начала отправки первой части команды
```

```
                        when COMMAND_STOPM0 =>
                            if i2c_busy = '1' then
                                state <= COMMAND_STOPM;
                            end if;
                        --  Отправка второй части команды
                        when COMMAND_STOPM =>
                            -- Если первая часть отправляется, подготовить вторую
часть к отправке
                            if i2c_busy = '1' then
                                i2c_wr_s <= stop_periodic_measurement(1);
                            -- Иначе отправить вторую часть и заранее завершить
работу модуля i2c
                            else
                                i2c_start <= '0';
                                state <= STOP0;
                            end if;

                        --! Завершение работы
                        --   (Используется при записи) Ожидание начала работы
датчика
                        when STOP0 =>
                            if i2c_busy = '1' then
                                state <= STOP;
                            end if;
                        --  Ожидание завершения работы датчика
                        when STOP =>
                            if i2c_busy = '0' then
                                doneSCD40 <= '1';
                                state <= IDLE;
                            end if;
    ----                        state <= IDLE;
                end case;
            end if;
        end if;
    end process;

end Behavioral;
```

Файл AHT21Driver.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity AHT21Driver is
    generic (
        AHT21_DELAY : natural
    );
    PORT (
        clk   : in STD_LOGIC;
        reset : in STD_LOGIC;

        i2c_done         : in  STD_LOGIC;
        i2c_busy         : in  STD_LOGIC;
        i2c_start        : out STD_LOGIC := '0';
        i2c_error        : in  STD_LOGIC;
        i2c_headAHT21     : out STD_LOGIC_VECTOR(7 downto 0) := (others =>
'0');
        i2c_rd           : in  STD_LOGIC_VECTOR(7 downto 0);
        i2c_wr             : out STD_LOGIC_VECTOR(7 downto 0) := (others =>
'0');
        i2c_stretchAHT21 : out STD_LOGIC := '0';

        startAHT21   : in  STD_LOGIC;
        doneAHT21    : out STD_LOGIC := '0';
        commandAHT21 : in STD_LOGIC_VECTOR(7 downto 0);
        resultAHT21   : out STD_LOGIC_VECTOR(7*8-1 downto 0) := (others =>
'0')
    );
end AHT21Driver;

architecture Behavioral of AHT21Driver is
    type COMMAND_SEQUENCE is array (0 to 2) of STD_LOGIC_VECTOR(7 downto 0);
    constant address : STD_LOGIC_VECTOR(7 downto 0) := x"38";
    constant trigger_measurement : COMMAND_SEQUENCE := (x"AC", x"33", x"00");

    signal command_s : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal result_s  : STD_LOGIC_VECTOR(7*8-1 downto 0) := (others => '0');
    signal counter  : natural range 0 to 7*8-1 := 7*8-1;
    signal i2c_head_s : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal i2c_wr_s   : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal counter_timer : natural range 0 to AHT21_DELAY := 0;

    type   states   is   (IDLE,   COMMAND_TRIGGERM0,   COMMAND_TRIGGERM1,
COMMAND_TRIGGERM20,   COMMAND_TRIGGERM2,   WAIT_EXECUTION0,   WAIT_EXECUTION1,
```

```
WAIT_EXECUTION, COMMAND_READM20, COMMAND_READM21, COMMAND_READM22, COMMAND_READM2,
STOP);
          signal state  : states := IDLE;
      begin
          resultAHT21 <= result_s;
          i2c_headAHT21 <= i2c_head_s;
          i2c_wr <= i2c_wr_s;

          process(clk) begin
              if clk'event and clk = '1' then
                  if reset = '1' then
                      doneAHT21 <= '0';
                      command_s <= (others => '0');
                      i2c_head_s <= (others => '0');
                      result_s <= (others => '0');
                      i2c_head_s <= (others => '0');
                      i2c_wr_s <= (others => '0');
                      i2c_start <= '0';
                      counter <= 7*8-1;
                      counter_timer <= 0;
                      state <= IDLE;
                  else
                      case state is
                          when IDLE =>
                              doneAHT21 <= '0';
                              if startAHT21 = '1' then
                                  result_s <= (others => '0');
                                  command_s <= commandAHT21;
                                  counter <= 7*8-1;
                                  counter_timer <= 0;

                                  -- Команда запроса результатов измерения
                                  if commandAHT21(6 downto 5) = "11" then
                                      i2c_head_s <= address(6 downto 0) & '0';
                                      i2c_wr_s <= trigger_measurement(0);
                                      i2c_start <= '1';
                                      state <= COMMAND_TRIGGERM0;
                                  else
                                      result_s <= x"B0000000000000";
                                      state <= STOP;
                                  end if;
```

```
            end if;

        -- Ожидаем когда модуль i2c начнет работу
        when COMMAND_TRIGGERM0 =>
            if i2c_busy = '1' then
                state <= COMMAND_TRIGGERM1;
            end if;

        -- Отправка части команды запуска измерения
        when COMMAND_TRIGGERM1 =>
            if i2c_busy = '1' then
                i2c_wr_s <= trigger_measurement(1);
            else
                state <= COMMAND_TRIGGERM20;
            end if;
        -- Ожидаем когда модуль i2c начнет отправку части команды
        when COMMAND_TRIGGERM20 =>
            if i2c_busy = '1' then
                state <= COMMAND_TRIGGERM2;
            end if;

        -- Отправка части команды запуска измерения
        when COMMAND_TRIGGERM2 =>
            if i2c_busy = '1' then
                i2c_wr_s <= trigger_measurement(2);
            else
                i2c_head_s <= address(6 downto 0) & '1';
                i2c_start <= '0';
                state <= WAIT_EXECUTION0;
            end if;

        -- Ожидаем когда модуль i2c начнет отправку последней
части команды
        when WAIT_EXECUTION0 =>
            if i2c_busy = '1' then
                state <= WAIT_EXECUTION1;
            end if;

        -- Ожидаем когда модуль i2c закончит отправку
        when WAIT_EXECUTION1 =>
            if i2c_busy = '0' then
```

```
                state <= WAIT_EXECUTION;
            end if;

        -- Задержка
        when WAIT_EXECUTION =>
            if counter_timer = AHT21_DELAY then
                i2c_start <= '1';
                state <= COMMAND_READM20;
            else
                counter_timer <= counter_timer + 1;
            end if;

        -- Ожидаем когда модуль i2c начнет отправку заголовка
        when COMMAND_READM20 =>
            if i2c_busy = '1' then
                state <= COMMAND_READM21;
            end if;

        -- Ожидаем когда модуль i2c закончит чтение байта
        when COMMAND_READM21 =>
            if i2c_busy = '0' then
                state <= COMMAND_READM22;
            end if;

        -- Ожидаем пока модуль i2c начнет чтение следующего байта
        when COMMAND_READM22 =>
            if i2c_busy = '1' then
                result_s(counter downto counter - 7) <= i2c_rd;
                -- Если будет прочитан последний байт, то заранее
сбросить флаг запуска модуля i2c
                if counter = 7 then
                    i2c_start <= '0';
                end if;
                state <= COMMAND_READM2;
            end if;

        -- Чтение поступающей с модуля i2c информации
        when COMMAND_READM2 =>
            -- read data
            if i2c_busy = '0' then
                --result_s(counter downto counter - 7) <= i2c_rd;
```

```
                                if counter = 7 then
                                    state <= STOP;
                                else
                                    counter <= counter - 8;
                                    state <= COMMAND_READM21; --!
                                end if;
                            end if;

                        when STOP =>
                            if i2c_busy = '0' then
                                doneAHT21 <= '1';
                                state <= IDLE;
                            end if;
----                            state <= IDLE;

                end case;

            end if;
        end if;
    end process;

end Behavioral;
```

Файл rx.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

entity rx is
    generic (
        --rate divider
        BAUD_DIVIDER : natural;
        --parity
        PARITY_FLAG  : boolean;
        PARITY_EVEN  : boolean
    );
    port (
        --clock
        clk   : in STD_LOGIC;
        --sync reset
```

```
        reset : in STD_LOGIC;

        --receiver port
        rx_serial : in STD_LOGIC;
        --receiver buffer
        rx_data : out STD_LOGIC_VECTOR(7 downto 0);--aa
        --receiver error flag
        rx_error  : out STD_LOGIC := '0';
        --receiver done flag
        rx_done   : out STD_LOGIC := '0'
    );
end entity;

architecture rtl of rx is
    --receiver buffer signal
    signal rx_data_s : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

    --receiver's states
    type RX_STATES is (IDLE, READ_START, READ_PARITY, READ_DATA, READ_STOP);
    --receiver current state
    signal rx_state : RX_STATES := IDLE;

    --divider counter
    signal baud_counter  : natural range 0 to BAUD_DIVIDER := 0;
    --receiver's bit index
    signal bit_index     : natural range 0 to 7 := 0;
    --sample time
    constant sample_time : natural := BAUD_DIVIDER/2;
    --parity bit buffer
    signal parity_bit    : STD_LOGIC := '0';

begin
    --rx_buffer - out
    rx_data <= rx_data_s;

    process (clk) begin
        if clk'event and clk = '1' then
            --reset
            if reset = '1' then
                rx_data_s <= (others => '0');
                rx_error <= '0';
```

```
            rx_done <= '0';
            rx_state <= IDLE;
            baud_counter <= 0;
            bit_index <= 0;
            parity_bit <= '0';
        else
            case rx_state is
                --when receiver is idle
                when IDLE =>
                    rx_done <= '0';
                    rx_error <= '0';
                    --incoming data
                    if rx_serial = '0' then
                        rx_state <= READ_START;
                    end if;
                --when incoming data, rx_serial = 0
                when READ_START =>
                    --pick bit in the middle of baud
                    if baud_counter = sample_time - 1 then
                        baud_counter <= 0;
                        --incoming data correct
                        if rx_serial = '0' then
                            rx_state <= READ_DATA;
                        --incoming data incorrect
                        else
                            rx_error <= '1';
                            rx_state <= IDLE;
                        end if;
                    --increment baud counter
                    else
                        baud_counter <= baud_counter + 1;
                    end if;
                --when receive data
                when READ_DATA =>
                    --pick bit in the middle of baud
                    if baud_counter = BAUD_DIVIDER - 1 then
                        baud_counter <= 0;
                        rx_data_s(bit_index) <= rx_serial;
                        --last bit received
                        if bit_index = 7 then
                            bit_index <= 0;
```

```
                                --if parity control - check parity bit
                                if PARITY_FLAG then
                                    rx_state <= READ_PARITY;
                                --if not parity control - stop reading
                                else
                                    rx_state <= READ_STOP;
                                end if;
                            --increment bit index
                            else
                                bit_index <= bit_index + 1;
                            end if;
                        --increment baud counter
                        else
                            baud_counter <= baud_counter + 1;
                        end if;
                    --when parity bit check
                    when READ_PARITY =>
                        --pick bit in the middle of baud
                        if baud_counter = BAUD_DIVIDER - 1 then
                            baud_counter <= 0;
                            parity_bit <= rx_serial;
                            --if even parity check
                            if PARITY_EVEN then
                                --error
                                if  parity_bit  /=  not  xor_reduce(rx_data_s)
then
                                    rx_error <= '1';
                                    rx_state <= IDLE;
                                --success
                                else
                                    rx_state <= READ_STOP;
                                end if;
                            --if odd parity check
                            else
                                --error
                                if parity_bit /= xor_reduce(rx_data_s) then
                                    rx_error <= '1';
                                    rx_state <= IDLE;
                                --success
                                else
                                    rx_state <= READ_STOP;
```

```
                                end if;
                            end if;
                        --increment baud counter
                        else
                            baud_counter <= baud_counter + 1;
                        end if;
                    --when received byte
                    when READ_STOP =>
                        --pick bit in the middle of baud
                        if baud_counter = BAUD_DIVIDER - 1 then
                            baud_counter <= 0;
                            --if data ended
                            if rx_serial = '1' then
                                rx_done <= '1';
                            --else error
                            else
                                rx_error <= '1';
                            end if;
                            rx_state <= IDLE;
                        --increment baud counter
                        else
                            baud_counter <= baud_counter + 1;
                        end if;
                end case;
            end if;
        end if;
    end process;
end architecture;
```

Файл tx.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

entity tx is
    generic (
        BAUD_DIVIDER : natural;
        PARITY_FLAG  : boolean;
        PARITY_EVEN  : boolean
    );
    port (
```

```
            clk   : in STD_LOGIC;
            reset : in STD_LOGIC;

            tx_serial : out STD_LOGIC := '1';
            tx_data : in STD_LOGIC_VECTOR(7 downto 0);
            tx_start : in STD_LOGIC;
            tx_busy   : out STD_LOGIC := '0'
        );
    end entity;

    architecture rtl of tx is
        type TX_STATES is (IDLE, WRITE_START, WRITE_DATA, WRITE_PARITY,
WRITE_STOP);
        signal tx_state : TX_STATES := IDLE;
        signal baud_counter  : natural range 0 to BAUD_DIVIDER := 0;
        signal bit_index     : natural range 0 to 8 := 0;
        signal tx_data_s : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    begin
        process(clk, reset) begin
            if reset = '1' then
                tx_serial <= '1';
                tx_busy <= '0';
                tx_state <= IDLE;
                baud_counter <= 0;
                bit_index <= 0;
                tx_data_s <= (others => '0');
            elsif clk'event and clk = '1' then
                case tx_state is
                    when IDLE =>
                        tx_serial <= '1';
                        tx_busy <= '0';
                        if tx_start = '1' then
                            tx_data_s <= tx_data;
                            tx_busy <= '1';
                            tx_serial <= '0';
                            tx_state <= WRITE_START;
                        end if;
                    when WRITE_START =>
                        if baud_counter = BAUD_DIVIDER - 1 then
                            baud_counter <= 0;
                            tx_serial <= tx_data_s(bit_index);
```

```
                tx_state <= WRITE_DATA;
            else
                baud_counter <= baud_counter + 1;
            end if;
        when WRITE_DATA =>
            if baud_counter = BAUD_DIVIDER - 2 then
                bit_index <= bit_index + 1;
                baud_counter <= baud_counter + 1;
            elsif baud_counter = BAUD_DIVIDER - 1 then
                baud_counter <= 0;
                if bit_index = 8 then
                    bit_index <= 0;
                    if PARITY_FLAG then
                        if PARITY_EVEN then
                            tx_serial <= xor_reduce(tx_data_s);
                        else
                            tx_serial <= not xor_reduce(tx_data_s);
                        end if;
                        tx_state <= WRITE_PARITY;
                    else
                        tx_serial <= '1';
                        tx_state <= WRITE_STOP;
                    end if;
                else
                    tx_serial <= tx_data_s(bit_index);
                end if;
            else
                baud_counter <= baud_counter + 1;
            end if;
        when WRITE_PARITY =>
            if baud_counter = BAUD_DIVIDER - 1 then
                baud_counter <= 0;
                tx_serial <= '1';
                tx_state <= WRITE_STOP;
            else
                baud_counter <= baud_counter + 1;
            end if;
        when WRITE_STOP =>
            if baud_counter = BAUD_DIVIDER - 1 then
                baud_counter <= 0;
                tx_data_s <= (others => '0');
```

```
                        tx_busy <= '0';
                        tx_state <= IDLE;
                    else
                        baud_counter <= baud_counter + 1;
                    end if;
            end case;
        end if;
    end process;
end architecture;
```

Файл top_measurer.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_measurer is
    generic (
        -- Частота системного синхросигнала
        CLK_FREQUENCY  : natural := 50*1000000;
        -- Частота синхросигнала I2C
        I2C_FREQUENCY  : natural := 400*1000;
        -- transfer rate
        UART_BAUD_RATE : natural := 115200;--9600;
        -- parity
        PARITY_FLAG    : boolean := false;
        PARITY_EVEN    : boolean := false
    );
    Port (
        -- Синхросигнал
        clk   : in STD_LOGIC;
        -- Сброс
        reset : in STD_LOGIC := '1';

        -- Линия последовательных данных I2C
        sda : inout std_logic;
        -- Линия синхронизации I2C
        scl : inout std_logic;

        -- rx
        rx : in STD_LOGIC;
        -- tx
        tx : out STD_LOGIC := '1'
    );
```

```
end top_measurer;

architecture Behavioral of top_measurer is
    -- Время чтения измерений с датчика SCD40 (~1 мс)
    constant SCD40_DELAY : natural := CLK_FREQUENCY/1000;
    -- Время измерения датчика AHT21 (~80 мс)
    constant AHT21_DELAY : natural := CLK_FREQUENCY/12;

    -- Буферы данных UART
    signal rx_buffer : STD_LOGIC_VECTOR(7 downto 0) := ((others => '0'));
    signal tx_buffer : STD_LOGIC_VECTOR(7 downto 0) := ((others => '0'));

    -- Флаг получения байта информации
    signal rx_done  : STD_LOGIC := '0';
    -- Флаг запуска передатчика
    signal tx_start : STD_LOGIC := '0';
    -- Флаг работы передатчика
    signal tx_busy  : STD_LOGIC := '0';

    -- I2C заголовок (адрес датчика и тип операции)
    signal i2c_header : STD_LOGIC_VECTOR(7 downto 0) := ((others => '0'));
    -- Буферы данных I2C
    signal data_rd    : STD_LOGIC_VECTOR(7 downto 0) := ((others => '0'));
    signal data_wr    : STD_LOGIC_VECTOR(7 downto 0) := ((others => '0'));

    -- Флаг окончания работы I2C модуля
    signal i2c_done    : STD_LOGIC := '0';
    -- Флаг работы I2C модуля
    signal i2c_busy    : STD_LOGIC := '0';
    -- Флаг запуска I2C модуля
    signal i2c_start   : STD_LOGIC := '0';
    -- Флаг ошибки I2C модуля
    signal ack_error   : STD_LOGIC := '0';
    -- Флаг удерживания I2C модуля
    signal i2c_stretch : STD_LOGIC := '0';

    -- UART модуль - для связи с компьютером
    -- Принимает от ПК команды (1 байт) и возвращает результат измерения (для
AHT21 - 7 байт, для SCD40 - 9 байт)
    component uart is
        generic (
```

```
            CLK_FREQUENCY : natural;
            BAUD_RATE     : natural;
            PARITY_FLAG   : boolean;
            PARITY_EVEN   : boolean
        );
        port (
            clk   : in STD_LOGIC;
            reset : in STD_LOGIC;

            rx_serial : in  STD_LOGIC;
            tx_serial : out STD_LOGIC;

            rx_data : out STD_LOGIC_VECTOR(7 downto 0);
            rx_done : out STD_LOGIC;

            tx_data  : in  STD_LOGIC_VECTOR(7 downto 0);
            tx_start : in  STD_LOGIC;
            tx_busy  : out STD_LOGIC
        );
    end component;

    -- Модуль управления драйверами датчиков
    --  Декодирует команду (номер датчика, номер команды), предоставляет
соответствующему датчику разрешение управления шиной I2C,
    --  отправляет результаты измерения на передатчик UART модуля
    component drivers is
        generic (
            SCD40_DELAY : natural;
            AHT21_DELAY : natural
        );
        port (
            clk   : in std_logic;
            reset : in std_logic;

            uart_done  : in  STD_LOGIC;
            uart_busy  : in  STD_LOGIC;
            uart_start : out STD_LOGIC;
            uart_rd    : in  STD_LOGIC_VECTOR(7 downto 0);
            uart_wr    : out STD_LOGIC_VECTOR(7 downto 0);

            i2c_done    : in  STD_LOGIC;
```

```
            i2c_busy    : in  STD_LOGIC;
            i2c_start   : out STD_LOGIC;
            i2c_error   : in  STD_LOGIC;
            i2c_head    : out STD_LOGIC_VECTOR(7 downto 0);
            i2c_rd      : in  STD_LOGIC_VECTOR(7 downto 0);
            i2c_wr      : out STD_LOGIC_VECTOR(7 downto 0);
            i2c_stretch : out STD_LOGIC
        );
    end component;

    -- Модуль I2C
    -- Обеспечивает связь между ПЛИС и датчиками по стандарту I2C
    component i2c_master is
        generic (
            CLK_FREQUENCY : natural;
            I2C_FREQUENCY : natural
        );
        port (
            clk     : in std_logic;
            reset_n : in std_logic;

            sda : inout std_logic;
            scl : inout std_logic;

            i2c_start   : in     std_logic;
            i2c_busy    : out    std_logic;
            i2c_done    : out    std_logic;
            ack_error   : buffer std_logic;
            i2c_stretch : in   std_logic;

            address : in  std_logic_vector(6 downto 0);
            rw      : in  std_logic;
            data_wr : in  std_logic_vector(7 downto 0);
            data_rd : out std_logic_vector(7 downto 0)
        );
    end component;
begin
    uart_inst : uart
     generic map (
        CLK_FREQUENCY => CLK_FREQUENCY,
        BAUD_RATE => UART_BAUD_RATE,
```

```
        PARITY_FLAG => PARITY_FLAG,
        PARITY_EVEN => PARITY_EVEN
    )
     port map (
        clk => clk,
        reset => not reset,
        rx_serial => rx,
        tx_serial => tx,
        rx_data => rx_buffer,
        tx_data => tx_buffer,
        rx_done => rx_done,
        tx_start => tx_start,
        tx_busy => tx_busy
    );

    drivers_inst : drivers
     generic map(
        SCD40_DELAY => SCD40_DELAY,
        AHT21_DELAY => AHT21_DELAY
    )
     port map (
        clk => clk,
        reset => not reset,
        uart_done => rx_done,
        uart_busy => tx_busy,
        uart_start => tx_start,
        uart_rd => rx_buffer,
        uart_wr => tx_buffer,

        i2c_done => i2c_done,
        i2c_busy => i2c_busy,
        i2c_start => i2c_start,
        i2c_error => ack_error,
        i2c_head => i2c_header,
        i2c_rd => data_rd,
        i2c_wr => data_wr,
        i2c_stretch => i2c_stretch
    );

    i2c_mstr_inst : i2c_master
     generic map(
```

```
            CLK_FREQUENCY => CLK_FREQUENCY,
            I2C_FREQUENCY => I2C_FREQUENCY
        )
         port map(
            clk => clk,
            reset_n => reset,
            sda => sda,
            scl => scl,
            i2c_start => i2c_start,
            i2c_busy => i2c_busy,
            i2c_done => i2c_done,
            ack_error => ack_error,
            address => i2c_header(7 downto 1),
            rw => i2c_header(0),
            data_wr => data_wr,
            data_rd => data_rd,
            i2c_stretch => i2c_stretch
        );


    end Behavioral;
```

Приложение В

(обязательная)

Программно-аппаратный комплекс измерения и контроля микроклимата

Презентация

1

Программно-аппаратный комплекс измерения и контроля микроклимата

Автор работы и докладчик:

студент группы 22ВВП2 Куракин Никита Николаевич

Руководитель работы:

к.т.н., доцент каф. ВТ Федюнин Роман Николаевич

Пенза, 2026

2

Актуальность работы

Микроклимат – состояние внутренней среды

Актуальность – необходимость контроля параметров микроклимата в помещениях в соответствии с *ГОСТ 30494-2011*

3

Цель и задачи работы

Цель работы – разработать программно-аппаратный комплекс измерения и контроля микроклимата, превосходящий бытовые и промышленные решения.

Задачи работы:

- Анализ аналогов систем измерения;
- Выбор компонентов для разработки комплекса;
- Описание архитектуры комплекса;
- Проектирование и реализация комплекса;
- Тестирование комплекса.

4

Анализ аналогов

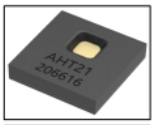
Характеристика	Производители систем измерения и контроля микроклимата		
	Научно-техническое предприятие «ТКА»	Testo Instruments Co. Ltd.	Xiaomi
Тип систем	Профессиональные приборы для определения показателей микроклимата	Комбинированные измерители, измерители-регистраторы	Бытовая система датчиков измерения микроклимата
Измеряемые параметры	Температура, влажность воздуха, скорость движения воздуха, освещённость, атмосферное давления	Температура, влажность воздуха, скорость движения воздуха, освещённость, атмосферное давления	Температура, влажность
Интерфейсы	Wi-Fi, USB	Bluetooth, USB, WLAN	Bluetooth
ПО	ПО для чтения сохранённых данных, ПО для мониторинга показаний	ПО для чтения и архивации данных	Мобильное ПО для мониторинга показаний
Особенности	Многие системы внесены в Госреестр средств измерений	Собственная экосистема для объединения нескольких измерителей	Собственная экосистема «Умный дом»
Недостатки	Набор датчиков для измерения определен производителем, нет возможности масштабирования функционала	Набор датчиков для измерения определен производителем	Набор датчиков для измерения определен производителем, нет возможности масштабирования функционала

5

Выбор компонентов

Датчики измерения параметров микроклимата:

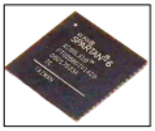
- AHT21 – датчик измерения температуры и относительной влажности воздуха:



- SCD40 – датчик измерения концентрации углекислого газа в воздухе:

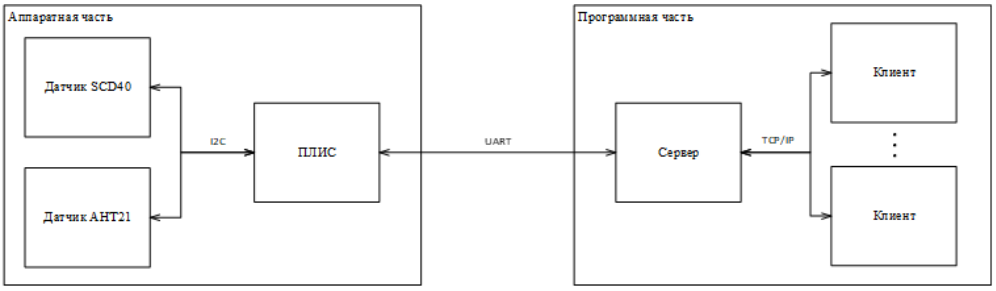
Открытая аппаратная платформа:

- **П**рограммируемая **Л**огическая **И**нтегральная **С**хема (**ПЛИС**) Spartan-6 XC6SLX16-2FTG256C



6

Архитектура комплекса



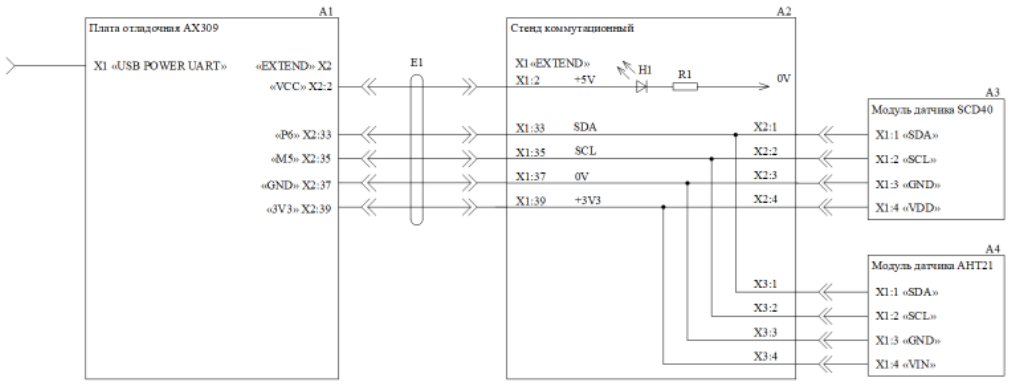
UART – последовательный асинхронный интерфейс связи между устройствами

I2C – последовательный синхронный интерфейс связи между устройствами

TCP/IP – сетевая модель передачи данных

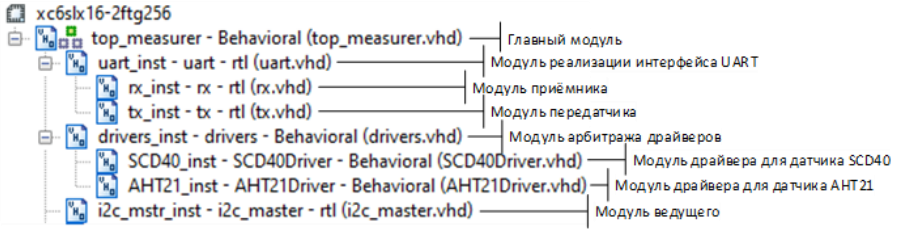
7

Разработка аппаратной части
Электрическая схема соединений



8

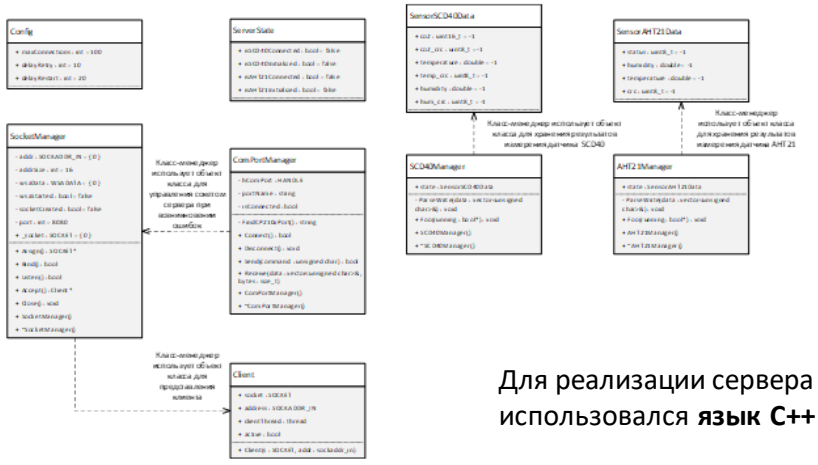
Разработка аппаратной части
Программируемая логическая интегральная схема



Для описания логических схем использовался **язык VHDL**

9

Разработка программной части
Серверные модули

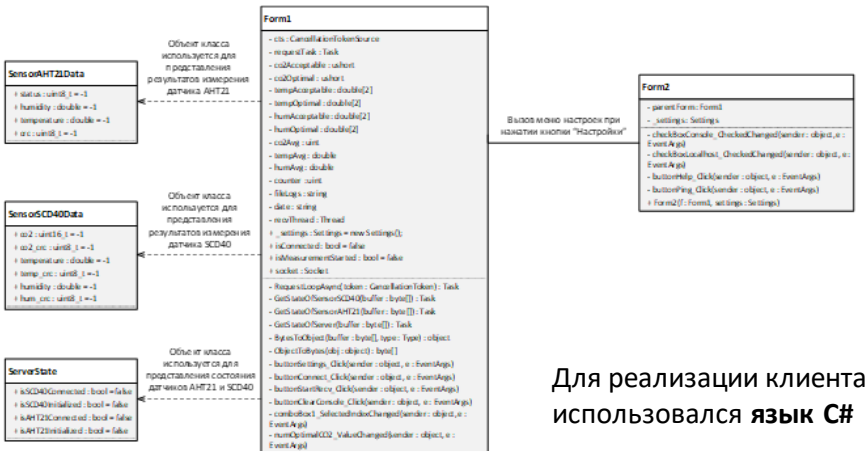


Для реализации сервера использовался **язык C++**

Изм.	Лист	№ докум.	Подпись	Дата	ПГУ 1.090301.41.001 ПЗ	Лист 165

10

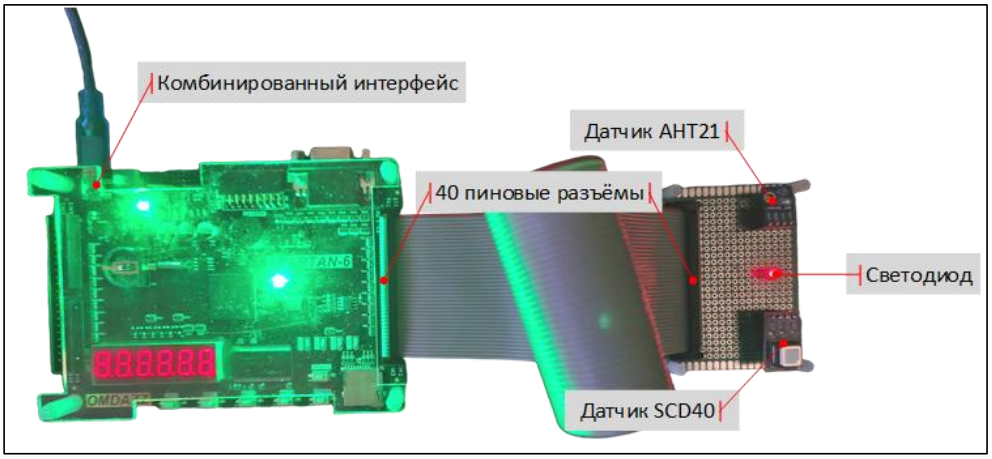
Разработка программной части
Клиентские модули



Для реализации клиента использовался **язык C#**

11

Демонстрация комплекса
Аппаратная часть



12

Демонстрация комплекса
Программная часть – сервер

CCS.exe — Исполняемый файл сервера

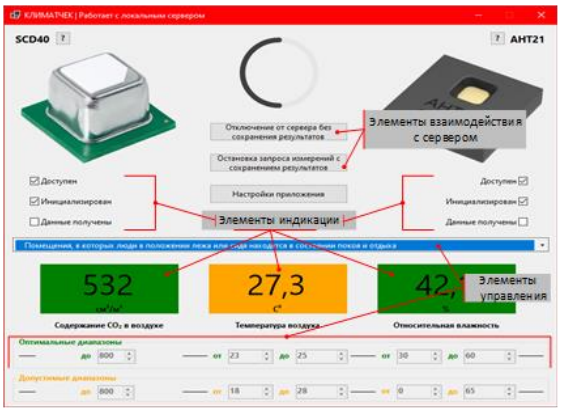
config.ini — Файл настроек

Пакетные файлы управления сервером:

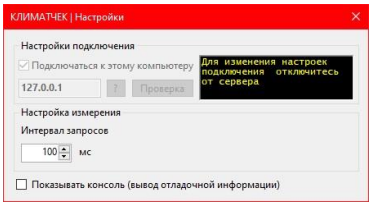
- Перезапустить сервер.bat
- Получить записи сервера.bat
- Удалить сервер.bat
- Установить сервер.bat

13

Демонстрация комплекса
Программная часть – клиент



Главное окно программы



Окно настроек

14

Тестирование комплекса

Сценарий	Ожидаемый результат	Реальный результат	Итог сценария
Установка сервера в виде службы при уже установленной службе	Событие об ошибке установки, установка не выполняется	Ошибка «Указанная служба уже существует», установка не выполнилась	Пройден
Удаление неустановленной службы сервера	Событие об ошибке удаления, удаление не выполняется	Ошибка «Указанная служба не установлена», удаление не выполнилось	Пройден
Отключение ПЛИС во время работы комплекса	Событие об ошибке соединения, закрытие соединения с клиентами, периодический опрос соединения	События «Потеряно соединение с портом», перезапуск сервера и ожидание подключения ПЛИС	Пройден

Реальные результаты сценария **совпадают** с ожидаемыми

15

Заключение работы

В результате выпускной квалификационной работы был **разработан программно-аппаратный комплекс**:

- Проанализированы аналоги систем;
- Проанализированы и выбраны компоненты разработки;
- Спроектирована архитектура комплекса;
- Разработаны модули комплекса;
- Проведено тестирование комплекса.

Перспективы:

- Расширение поддержки датчиков;
- Реализация веб интерфейса;
- Добавление графиков.

16

Спасибо за внимание!