

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«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

СИСТЕМА ОПРОСА ДАТЧИКОВ ПО
ИНТЕРФЕЙСУ I2C С ПЕРЕДАЧЕЙ
ДАННЫХ ПО UART И ЗАНЕСЕНИЕМ В
БАЗУ ДАННЫХ SQLITE

Студент

Гороз 17.05.2026
(подпись, дата)

Городничев Максим Игоревич
(ФИО полностью)

Руководитель

Юрова 17.05.2026
(подпись, дата)

Юрова О.В.
(фамилия, инициалы)

Нормоконтролёр

Бычков 20.05.2026
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

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

Заведующий кафедрой

Митрохин 22.05.2026
(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

08 июня 2026 № 13)

Секретарь ГЭК

Гурин 08.06.2026
(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 Городничев Максим Игоревич гр. 22ВВП2 факультета вычислительной техники
2. Тема работы: «Система опроса датчиков по интерфейсу I2C с передачей данных по UART и занесением в базу данных SQLite»

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

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О.В.

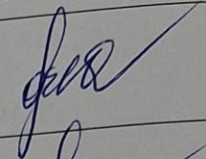
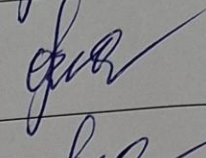
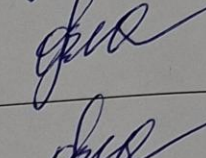
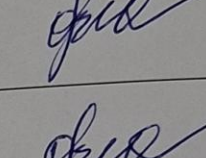
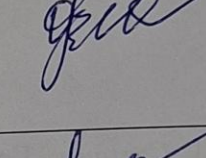
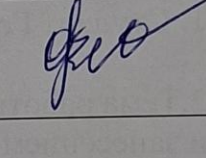
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать программно-аппаратную систему для опроса датчиков по интерфейсу I2C, передачи полученных данных на персональный компьютер по интерфейсу UART и их последующей записи в базу данных. В качестве аппаратной платформы использовать ПЛИС Spartan-6, разработку логики вести на языке VHDL. Реализовать клиентское приложение с использованием встраиваемой СУБД SQLite. Система должна обеспечивать корректное считывание данных, их математическую обработку и хранение истории измерений в структурированном виде.

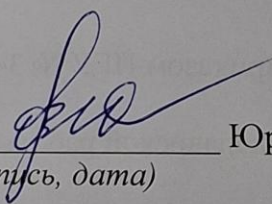
5. Перечень подлежащих разработке вопросов

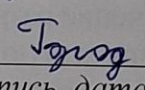
Требуется провести анализ предметной области и изучить существующие решения в сфере систем сбора данных. Обосновать выбор аппаратных средств и интерфейсов взаимодействия. Спроектировать архитектуру системы и разработать иерархию VHDL-модулей для управления периферией ПЛИС. Разработать алгоритмы работы интерфейсов I2C и UART. Спроектировать структуру базы данных и разработать программное обеспечение для приёма и парсинга данных. Выполнить тестирование системы.

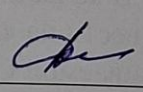
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	45	07.04.26 – 13.04.26	
2	Анализ темы, аналогов, постановка задачи	24	13.04.26 – 17.04.26	
3	Выбор и настройка инструментов разработки	50	17.04.26 – 24.04.26	
4	Проектирование архитектуры	30	24.04.26 – 28.04.26	
5	Разработка программного обеспечения для ПЛИС и ПК, тестирование системы	151	28.04.26 – 17.05.26	
6	Оформление пояснительной записки	24	17.05.26 – 21.05.26	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы 07.04.26  Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент 07.04.26  Городничев М.И.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
27.05.2026 (подпись, дата)

Реферат

Бакалаврская работа содержит 146 листов, 59 рисунков, 2 таблицы, 7 источников, 4 приложения.

ОТЛАДОЧНАЯ ПЛАТА, ПЛИС, VHDL, I2C, UART, C++, ДАННЫЕ, БАЗА ДАННЫХ, SQLITE, ДАТЧИК, SPARTAN-6, СИСТЕМА СБОРА ДАННЫХ, DAQ, COM-ПОРТ

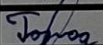
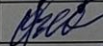
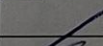
Цель работы – разработать масштабируемую систему, позволяющая собирать, обрабатывать и хранить данные с цифровых датчиков.

Объектами разработки являются:

– программное обеспечение (ПО) на программируемую логическую интегральную схему (ПЛИС) для сбора данных с датчиков по нажатию клавиш на клавиатуре и отправки данных на персональный компьютер (ПК);

– программа на ПК для подключения отладочной платы по COM-порту, настройки скорости интерфейса UART, отображения данных на экране, хранения данных в базе данных (БД).

В результате проделанной работы разработана система, полностью соответствующая поставленной задаче, позволяющая производить опрос цифровых датчиков по интерфейсу I2C, передавать данные на ПК по интерфейсу UART, где они обрабатываются, отображаются пользователю и сохраняются в БД.

					ПГУ 1.090301.35.001 ПЗ			
Изм.	Лист	№ докум.	Подпис	Дата	Система опроса датчиков по интерфейсу I2C с передачей данных по UART и занесением в базу данных SQLite Пояснительная записка	Лит.	Лист	Листов
Разраб.		Городничев		17.05.20			2	146
Провер.		Юрова О.В.		17.05.20				
Н. Контр.		Бычков А.С.		20.05.20				
Утверд.		Митрохин		20.05.20				
						ФВТ, каф. ВТ гр. 22ВВП2		

Содержание

Перечень сокращений и обозначений	5
Введение.....	7
1 Анализ предметной области	8
1.1 Назначение и постановка задачи	8
1.2 Обзор аналогов	8
1.2.1 Прямые аналоги	8
1.2.2 Косвенные аналоги	10
1.2.3 Альтернативные подходы на базе микроконтроллеров	12
1.3 Технические требования к системе.....	14
2 Архитектура системы и выбор инструментов разработки	15
2.1 Архитектура системы	15
2.2 Выбор инструментов разработки	16
2.2.1 Интерфейс I2C.....	16
2.2.2 Интерфейс UART	17
2.2.3 Выбор датчиков.....	18
2.2.4 Выбор отладочной платы и ПЛИС.....	20
2.2.5 Выбор языка программирования и среды разработки для ПЛИС	21
2.2.6 Выбор языка программирования и среды разработки для ПК.....	22
2.2.7 Выбор СУБД.....	23
2.3 Установка и настройка рабочих сред	24
2.3.1 Установка и настройка Xilinx ISE 14.7	24
2.3.2 Установка и настройка Visual Studio 2022.....	36
3 Реализация и тестирование системы.....	43
3.1 Структура БД.....	43
3.2 Описание аппаратной части.....	44
3.3 Разработка ПО для ПЛИС	45
3.4 Разработка ПО для ПК.....	64
3.5 Тестирование системы.....	67
4 Безопасность жизнедеятельности	71

4.1 Правила безопасности при использовании ПК.....	71
4.2 Правила безопасности при использовании стенда.....	72
Заключение	73
Список использованных источников	74
Приложение А – UML-диаграммы.....	75
Приложение Б – Схема электрическая соединений.....	80
Приложение В – Листинги программ	82
Приложение Г – Презентация.....	140

Перечень сокращений и обозначений

БД – база данных

ПК – персональный компьютер

ПЛИС – программируемая логическая интегральная схема

ПО – программное обеспечение

СУБД – система управления базами данных

ASCII – (англ. American Standard Code for Information Interchange) – американский стандартный код для обмена информацией

COM – (англ. Communication port) – коммуникационный порт

DAQ – (англ. Data Acquisition) – сбор данных

DIY – (англ. Do It Yourself) – «сделай сам» (подход к самостоятельной разработке и сборке устройств)

HDL – (англ. Hardware Description Language) – язык описания аппаратуры

I2C – (англ. Inter-Integrated Circuit) – последовательная шина для связи интегральных схем

IoT – (англ. Internet of Things) – интернет вещей

ISE – (англ. Integrated Synthesis Environment) – интегрированная среда синтеза

JTAG – (англ. Joint Test Action Group) – специализированный аппаратный интерфейс для отладки и программирования

RX – (англ. Receive) – линия приёма данных в интерфейсе UART

SCL – (англ. Serial Clock) – линия тактирования в интерфейсе I2C

SDA – (англ. Serial Data) – линия передачи данных в интерфейсе I2C

TX – (англ. Transmit) – линия передачи данных в интерфейсе UART

UART – (англ. Universal Asynchronous Receiver-Transmitter) – универсальный асинхронный приёмопередатчик

UCF – (англ. User Constraints File) – файл пользовательских ограничений

UML – (англ. Unified Modeling Language) – унифицированный язык моделирования

					ПГУ 1.090301.35.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

USB – (англ. Universal Serial Bus) – универсальная последовательная шина

VHDL – (англ. VHSIC Hardware Description Language) – язык описания аппаратуры интегральных схем

Wi-Fi – (Wireless Fidelity) – технология беспроводной передачи данных

					ПГУ 1.090301.35.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В условиях современной промышленной автоматизации и развития лабораторных измерительных комплексов непрерывно возрастает потребность в создании гибких и производительных систем сбора данных (DAQ). Такие системы являются фундаментальным элементом для контроля технологических процессов, мониторинга состояния оборудования и проведения научных экспериментов. Существующие на рынке профессиональные DAQ-системы часто обладают высокой стоимостью и избыточностью, в то время как решения на базе микроконтроллеров могут не обеспечить необходимого параллелизма и предсказуемости для решения задач, требующих одновременного опроса множества датчиков. Использование ПЛИС для этих целей позволяет создать аппаратно-программные, масштабируемые и реконфигурируемые решения, что и определяет актуальность данной работы.

Целью выпускной работы является разработка, макетирование и отладка программно-аппаратного комплекса для опроса внешних датчиков по интерфейсу I2C с последующей передачей данных на ПК по интерфейсу UART с целью дальнейшего хранения и использования.

Необходимо разработать стенд и программное обеспечение для сопряжения цифровых датчиков с ПЛИС с последующей передачей данных на ПК. Программная часть состоит из программы для ПЛИС на языке VHDL и программы для ПК на языке C++.

1 Анализ предметной области

1.1 Назначение и постановка задачи

Основным назначением данной работы является разработка прототипа масштабируемой и гибкой системы сбора данных, способной служить основой для построения бюджетных измерительных комплексов. Комплекс предназначен для считывания показаний с группы цифровых датчиков, подключённых по общей шине, и их последующей передачи на ПК для обработки, визуализации и долгосрочного хранения.

В рамках данной работы ставится задача спроектировать, реализовать и протестировать систему, состоящую из аппаратного модуля и программного обеспечения для ПК и ПЛИС.

1.2 Обзор аналогов

Для определения места разрабатываемой системы в ряду существующих решений и для формирования требований к ней был проведён анализ аналогов.

1.2.1 Прямые аналоги

В качестве прямых аналогов разрабатываемой платформы выступают профессиональные и промышленные системы сбора данных. Эти комплексы являются основным ориентиром, так как решают ту же фундаментальную задачу – сбор, обработку и передачу данных с датчиков, – но на более высоком промышленном уровне.

Ведущим производителем в этой области является компания National Instruments с их платформой CompactDAQ, а также такие компании, как Advantech и Siemens. Системы DAQ представляют собой высокопроизводительные модульные платформы, способные с высокой точностью и скоростью собирать данные с десятков и сотен различных датчиков одновременно. Они широко применяются в промышленности для контроля технологических процессов, в научных исследованиях и для автоматизации сложных испытательных стендов.

Пример такой системы представлен на рисунке 1.

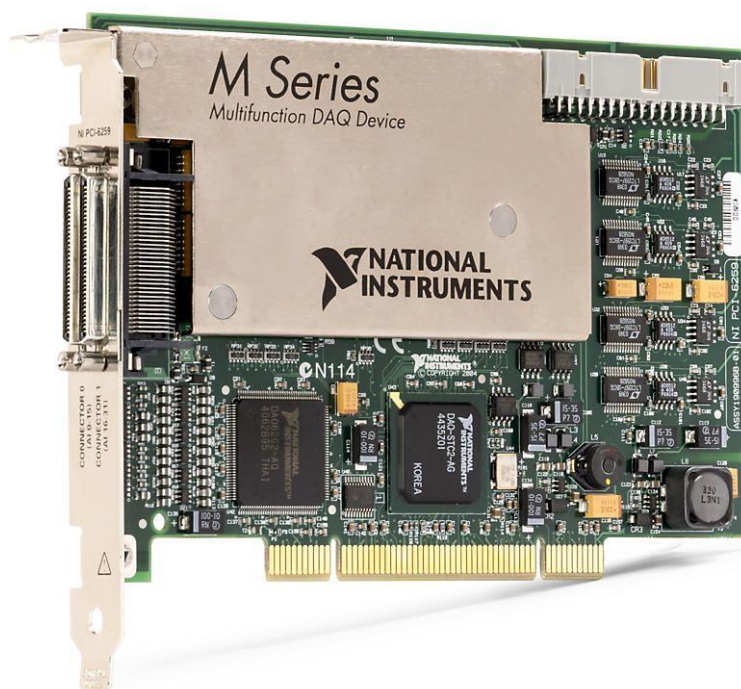


Рисунок 1 – DAQ-система

Ключевым отличием данных систем от разрабатываемого прототипа является их высокая стоимость, которая делает их экономически нецелесообразными для многих задач, а также их сложность и зачастую привязка к проприетарному ПО.

В тоже время, лабораторное контрольно-измерительное оборудование, такое как логические анализаторы и осциллографы с функцией декодирования протоколов, также можно считать узкоспециализированным аналогом. Эти приборы способны «опрашивать» и декодировать данные на цифровых шинах, включая I2C и UART. Однако их основное назначение – не длительный мониторинг и хранение данных, а кратковременная отладка и верификация работы аппаратного обеспечения. Примеры такого оборудования представлены на рисунках 2, 3.



Рисунок 2 – Осциллограф с функцией декодирования протоколов



Рисунок 3 – Логический анализатор

Таким образом, разработанная в данной работе система позиционируется как бюджетный и гибкий прототип, реализующий базовые принципы DAQ-систем. Она заимствует идеологию сбора, обработки и хранения данных у промышленных аналогов, но использует более доступную элементную базу и открытые инструменты разработки, что делает её оптимальным решением для прототипирования, образовательных и некритичных промышленных задач.

1.2.2 Косвенные аналоги

К косвенным аналогам можно отнести готовые потребительские устройства, которые выполняют функцию сбора данных в объёме параметров окружающей

среды. Однако, в отличие от промышленных систем, они предназначены исключительно для бытового использования и не обладают гибкостью, производительностью и масштабируемостью, необходимой для решения инженерных задач.

Одной группой таких устройств являются домашние цифровые метеостанции от производителей, таких как Bresser, La Crosse, Oregon Scientific. Эти устройства, как правило, состоят из внутреннего блока с дисплеем и одного или нескольких внешних датчиков. Они измеряют температуру, влажность и атмосферное давление. Их ключевым преимуществом является простота использования. Это готовые к работе решения, не требующие от пользователя специальных знаний. Однако их главным недостатком является закрытость системы. Пользователь не имеет доступа к «сырым» данным, не может изменять логику работы устройства, а возможности по хранению и анализу истории измерений сильно ограничены функциональностью конкретной модели. Пример такого устройства представлен на рисунке 4.



Рисунок 4 – Домашняя метеостанция

Другой группой косвенных аналогов являются датчики для системы «Умного дома», например, устройства экосистемы Xiaomi Aqara, Tuuya или работающие по протоколам Z-Wave/ZigBee. Эти модули также измеряют температуру, влажность и давление, но передают данные на центральный хаб. Преимуществом является их интеграция в более крупную систему автоматизации.

Недостатки схожи с метеостанциями: привязка к конкретной экосистеме, невозможность гибкой настройки и ограниченный доступ к данным, которые обычно обрабатываются в облаке производителя. Пример такого устройства представлен на рисунке 5.



Рисунок 5 – Датчик системы «Умный дом»

Таким образом, данные потребительские устройства, решая задачу сбора данных на бытовом уровне, не могут рассматриваться как аналоги разрабатываемой DAQ-системы. Они демонстрируют лишь общую востребованность задачи мониторинга физических параметров, но их архитектура и функциональность не могут быть взяты за основу при создании гибкого измерительного комплекса.

1.2.3 Альтернативные подходы на базе микроконтроллеров

Наиболее близкой по подходу к разработке к создаваемой системе является категория проектов, реализуемых в рамках DIY-сообщества. Эти решения, как правило, строятся на базе популярных и доступных отладочных плат с микроконтроллерами.

Самыми распространёнными платформами для таких проектов являются Arduino, ESP32/ESP8266 и однокристальные компьютеры, такие как Raspberry Pi.

независимых каналов, реализуются с помощью прерываний и программных ухищрений, что не гарантирует истинного параллелизма. Производительность такой системы ограничена тактовой частотой процессора и его архитектурой, что быстро становится узким местом при увеличении количества датчиков или повышении частоты их опроса.

В отличие от этого, использование ПЛИС позволяет реализовать истинно параллельную архитектуру. Каждый модуль системы может работать независимо и одновременно с другими, что обеспечивает предсказуемость, высокую скорость и аппаратную реконфигурируемость. Это является ключевым преимуществом и обоснованием выбора ПЛИС для создания гибкого и производительного ядра системы сбора данных.

1.3 Технические требования к системе

Аппаратная часть системы должна включать в себя отладочную плату на базе ПЛИС. Плата должна предоставлять доступ как к линиям питания, так и к программируемым контактам ПЛИС. Для реализации интерфейсов I2C и UART требуется наличие не менее четырёх свободных программируемых контактов. Кроме того, необходимы сами цифровые датчики, совместимые с протоколом I2C.

Программное обеспечение, функционирующее на ПК, должно быть совместимо с операционными системами семейства Windows. Для его работы требуются минимальные ресурсы, обеспечиваемые современным ПК: процессор с тактовой частотой не ниже 1 ГГц и не менее 2 Гб оперативной памяти. Программа должна иметь доступ к СОМ-порту для обмена данными с платой. Также требуется наличие на диске ПК не менее 50 Мб свободного пространства для установки приложения и хранения базы данных.

2 Архитектура системы и выбор инструментов разработки

2.1 Архитектура системы

Проектируемая система сбора данных представляет собой программно-аппаратный комплекс, который базируется на модульной двухуровневой архитектуре. Подобное логическое разделение позволяет эффективно распределить вычислительную нагрузку между аппаратной логикой и ресурсами ПК, а также закладывает основу для масштабируемости проекта в будущем. Общая структура взаимодействия компонентов разрабатываемого комплекса наглядно демонстрируется на диаграмме развёртывания, которая представлена в приложении А.4.

Нижний аппаратный уровень системы будет отвечать за непосредственный сбор данных и их первичную подготовку. Предполагается, что ядром данного уровня выступит отладочная плата с ПЛИС. Она будет выполнять роль ведущего устройства, которое по цифровой последовательной шине I2C осуществляет опрос подключённых ведомых цифровых датчиков. Для коммутации датчиков с отладочной платой планируется разработать макетную соединительную плату. Электрическая схема стенда представлена в приложении Б.1.

Связующим звеном между аппаратным и программным уровнями архитектуры выступит канал передачи данных, организованный на базе асинхронного интерфейса UART. Физическое сопряжение будет обеспечиваться аппаратным мостом, преобразующим сигналы UART в протокол USB. Благодаря этому на стороне ПК будет инициализирован виртуальный COM-порт, через который организуется дуплексный обмен пакетами данных и управляющими командами между отладочной платой и ПК.

Верхний программный уровень системы планируется реализовать в виде консольного клиентского приложения для ПК. Планируется, что данная программа возьмёт на себя задачи высокоуровневого управления измерительным комплексом, взаимодействия с пользователем и математической обработки

полученных «сырых» значений с датчиков. Приложение будет прослушивать открытый СОМ-порт, отправлять команды запроса конкретных параметров по нажатию заданных клавиш и выводить отформатированные результаты измерений в консоль. Предполагаемый алгоритм работы программы подробно описывается UML-диаграммой деятельности, которая представлена в приложении А.3.

Завершающим архитектурным элементом верхнего уровня выступает подсистема долговременного хранения информации. Для решения задачи сохранения истории измерений в клиентское приложение будет интегрирована встраиваемая система управления базами данных (СУБД) SQLite. Логика работы приложения подразумевает, что при каждом успешном цикле опроса датчиков программа будет автоматически генерировать запросы на добавление новых записей в локальный файл БД. Эти записи будут включать порядковый номер запроса, дату и время, а также итоговые высчитанные значения. Предложенный архитектурный подход позволит создать гибкий измерительный инструмент, полностью соответствующий поставленному заданию.

2.2 Выбор инструментов разработки

2.2.1 Интерфейс I2C

Для организации обмена данными между управляющим модулем и периферийными датчиками был выбран последовательный интерфейс I2C. Он обеспечивает синхронность, полудуплексность и позволяет объединять на одной шине множество устройств.

Интерфейс I2C подходит для этой задачи по нескольким причинам. Во-первых, он экономичен с точки зрения использования аппаратных ресурсов. Для работы шины с множеством устройств требуется всего две линии связи – SDA для данных и SCL для тактирования. Во-вторых, ключевым элементом протокола является система адресации. Каждое ведомое устройство на шине имеет уникальный адрес, что позволяет главному устройству целенаправленно общаться с любым из них. Именно этот механизм обеспечивает масштабируемость,

позволяя подключать к системе новые датчики без изменения аппаратной части управляющего модуля. Общая схема такой организации представлена на рисунке 7.

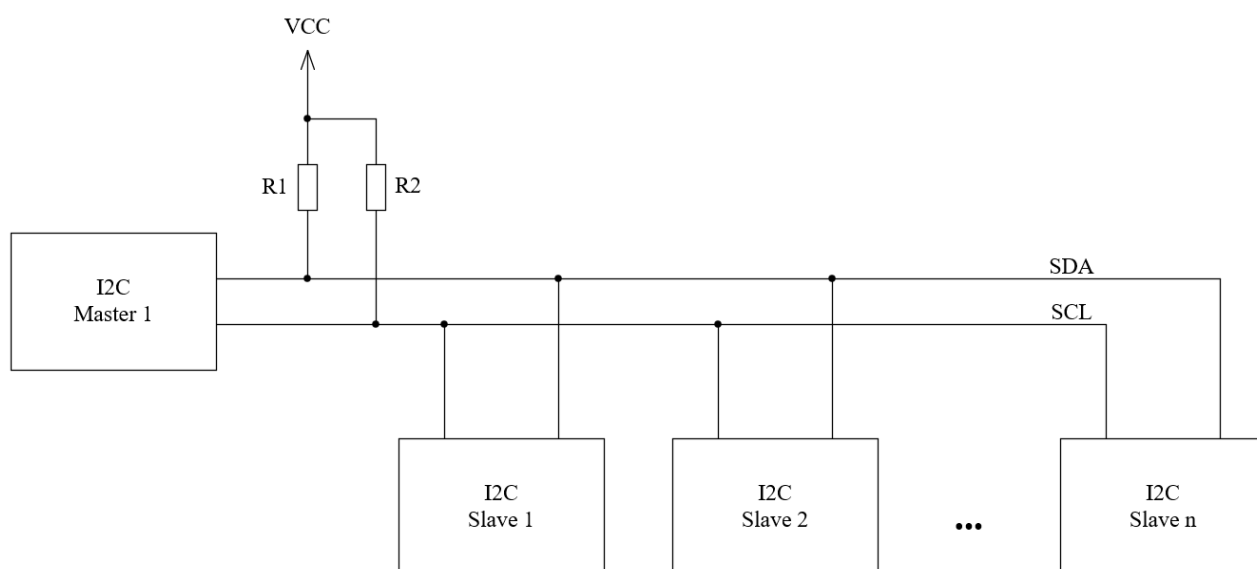


Рисунок 7 – Схема подключения устройств по шине I2C

Интерфейс I2C является общепринятым промышленным стандартом для подключения широкого спектра сенсоров и микросхем в промышленных и лабораторных условиях. Использование стандартного протокола гарантирует совместимость с большим количеством доступных на рынке датчиков, что является необходимым условием для создания гибкой и универсальной DAQ-системы.

2.2.2 Интерфейс UART

Для связи аппаратного модуля на базе ПЛИС с ПК был выбран интерфейс UART. Он является проверенным и надёжным решением для организации потоковой передачи данных между двумя устройствами.

В отличие от шинного интерфейса I2C, UART предназначен для соединений точка-точка. Обмен данными осуществляется в дуплексном режиме по двум линиям: TX для передачи данных и RX для приёма. Асинхронный характер интерфейса избавляет от необходимости использовать отдельную линию тактирования, что упрощает его реализацию. Для успешной коммуникации на

стороне приёмника и передатчика заранее должны быть согласованы параметры интерфейса, такие как скорость передачи данных, количество бит данных, наличие бита чётности и количество стоп-бит.

Интерфейс UART является чрезвычайно востребованным в промышленной автоматизации, встраиваемых системах и лабораторном оборудовании благодаря своей простоте, низким требованиям к ресурсам и предсказуемости. В современных системах для подключения к ПК применяется мост USB-UART. Специализированная микросхема на плате преобразует сигналы UART в протокол USB, создавая в операционной системе виртуальный COM-порт. Это позволяет любому современному ПК легко взаимодействовать с аппаратным модулем. Упрощённая схема взаимодействия показана на рисунке 8.

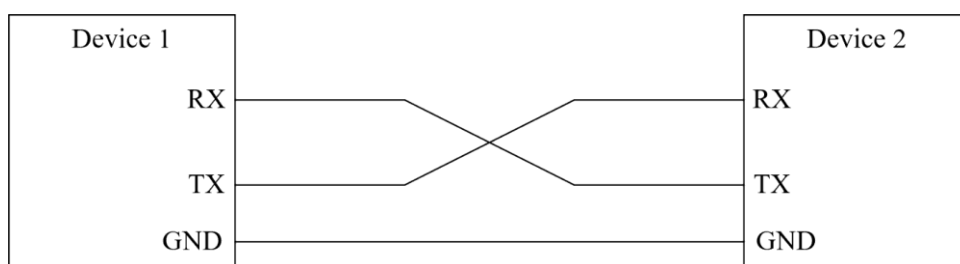


Рисунок 8 – Схема организации связи по UART

2.2.3 Выбор датчиков

Для практической проверки и демонстрации работоспособности измерительного тракта проектируемой системы были выбраны два широко распространённых и доступных цифровых датчика: АНТ21 и ВМР180. Важно подчеркнуть, что выбор этих датчиков носит демонстрационный характер. Разработанная система не привязана к измеряемым ими физическим величинам и может быть легко адаптирована для работы с любыми другими датчиками, поддерживающими интерфейс I2C, что и является одной из целей создания универсальной платформы.

Для измерения температуры и относительной влажности был выбран датчик АНТ21. Это высокоточный цифровой датчик компании ASAIR, который поставляется откалиброванным на заводе и не требует сложных вычислений для

получения конечных значений, что упрощает отладку базовой логики опроса данных [1].

Для измерения атмосферного давления был выбран датчик BMP180. Этот цифровой датчик производства компании Bosch Sensortec, в отличие от АНТ21, требует проведения ряда математических вычислений для получения итогового значения давления из «сырых» данных. Необходимость реализации этих вычислений на стороне ПК позволяет продемонстрировать возможности системы не только по сбору, но и по последующей обработке и преобразованию данных [2].

Таким образом, комбинация этих двух датчиков позволяет комплексно протестировать всю систему: от базового чтения данных до реализации более сложного протокола с последующей математической обработкой. Оба датчика могут быть подключены к одной шине I2C, что позволяет продемонстрировать масштабируемость шины на практике. Внешний вид используемых датчиков показаны на рисунках 9, 10.

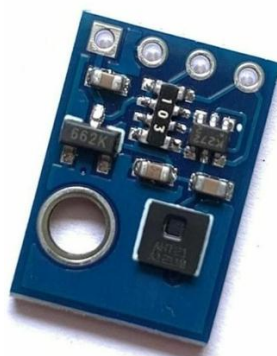


Рисунок 9 – Датчик АНТ21



Рисунок 10 – Датчик BMP180

2.2.4 Выбор отладочной платы и ПЛИС

В качестве центрального управляющего элемента выбрана ПЛИС. Этот выбор является фундаментальным для создания масштабируемой и высокопроизводительной системы сбора данных. В отличие от микроконтроллеров с их последовательным выполнением команд, ПЛИС представляет собой матрицу конфигурируемых логических блоков. Такая структура позволяет описывать и реализовывать аппаратуру, выполняющую задачи параллельно. Для системы это означает возможность одновременного, независимого и с предсказуемой задержкой опрашивать десятки различных датчиков, что невозможно реализовать на микроконтроллере без существенных компромиссов.

Для реализации проекта была выбрана готовая отладочная плата ALINX AX309. На ней установлена ПЛИС XC6SLX16 семейства Spartan-6 фирмы Xilinx. Применение покупной отладочной платы позволяет использовать стандартный набор интерфейсов и обеспечивает возможность масштабирования системы.

Данный выбор обусловлен несколькими факторами. Во-первых, плата ALINX AX309 содержит мост USB-UART, необходимый для связи с ПК. Во-вторых, микросхема XC6SLX16 обладает значительными ресурсами: более 14 тысяч логических ячеек и свыше 100 доступных пользователю контактов [3]. Эти ресурсы являются избыточными для демонстрационной задачи с двумя датчиками, однако они закладывают фундамент для дальнейшего масштабирования системы: добавления новых интерфейсов, реализации более сложных алгоритмов предварительной обработки данных прямо на ПЛИС и подключения значительно большего числа датчиков. Для выполнения сложных расчётов есть возможность построения контроллера внутри ПЛИС. Это полностью соответствует цели создания расширяемого прототипа DAQ-системы. Внешний вид используемой отладочной платы представлен на рисунке 11.

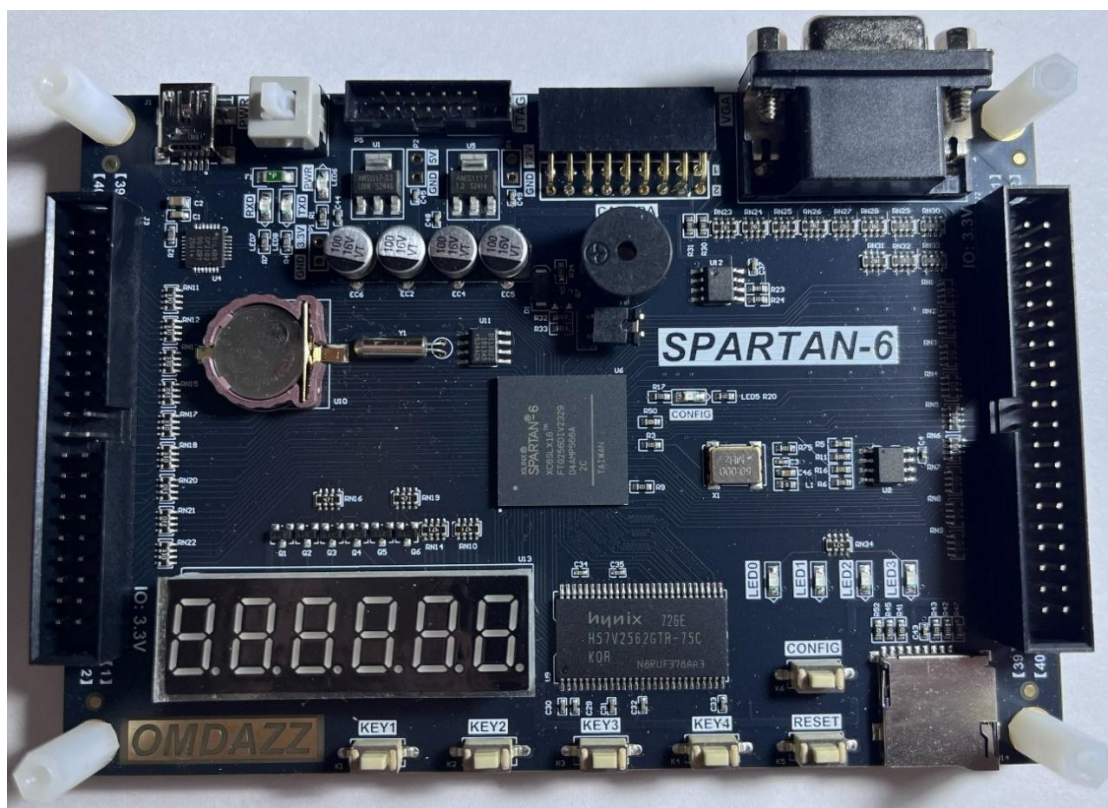


Рисунок 11 – Отладочная плата ALINX AX309

2.2.5 Выбор языка программирования и среды разработки для ПЛИС

Для программирования ПЛИС используются специализированные языки описания аппаратуры (HDL). В отличие от традиционных языков, которые описывают последовательность действий, HDL описывают структуру и поведение цифровых логических схем. Наиболее распространёнными и стандартизированными языками в этой области являются VHDL и Verilog.

Для данного проекта был выбран язык VHDL. VHDL является строго типизированным языком, что помогает избегать многих ошибок на этапе компиляции проекта. Его синтаксис, по сравнению с Verilog, является более многословным, однако это делает код более читаемым и самодокументируемым, что упрощает сопровождение и модификацию сложных проектов, каким и является ядро системы. VHDL позволяет описывать систему на разных уровнях абстракции, что даёт разработчику большую гибкость при проектировании как отдельных модулей, так и всей системы в целом.

В качестве среды разработки для ПЛИС Xilinx Spartan-6 была выбрана Xilinx

ISE 14.7. Данная версия является последним официальным выпуском от Xilinx, который обеспечивает полную поддержку семейства Spartan-6. ISE представляет собой комплексный пакет, включающий все необходимые инструменты: от текстового редактора и симулятора для проверки логики до синтезатора и утилит для генерации и загрузки файла прошивки в ПЛИС. Выбор данного программного обеспечения является единственно верным решением для работы с выбранной аппаратной платформой.

2.2.6 Выбор языка программирования и среды разработки для ПК

Для реализации клиентского приложения, предназначенного для взаимодействия с аппаратным модулем, был произведён выбор языка программирования C++. Данное приложение выполняет задачи приёма данных по СОМ-порту, их последующей обработки, визуализации в консоли и сохранения. Для решения этой задачи рассматривались три основных кандидата: C++, C# и Python.

Язык программирования Python, благодаря простоте синтаксиса и наличию библиотек для любых задач, является привлекательным инструментом для быстрого прототипирования. Однако, будучи интерпретируемым языком, он имеет два существенных недостатка для конечного продукта. Во-первых, он требует установки на компьютер пользователя самого интерпретатора Python и всех необходимых библиотек. Во-вторых, работа через слой интерпретации и использование автоматического сборщика мусора могут вносить непредсказуемые задержки, что нежелательно для системы, обрабатывающей потоки данных в реальном времени.

Язык программирования C# представляет собой современный, объектно-ориентированный язык, глубоко интегрированный в экосистему Windows. Он значительно упрощает и ускоряет процесс разработки по сравнению с C++ за счёт автоматического управления памятью и богатой библиотеки классов .NET. Однако это преимущество имеет обратную сторону: для работы приложения на C# требуется наличие на целевой машине установленного .NET Framework. Это

создаёт дополнительную зависимость и увеличивает «вес» приложения, что может быть нежелательно для системного, низкоуровневого ПО.

На этом фоне C++ был выбран как наиболее подходящее решение. Ключевым преимуществом C++ является то, что он компилируется непосредственно в нативный машинный код. Это позволяет создавать быстрые, компактные и полностью автономные исполняемые файлы, не требующие для своей работы никаких внешних зависимостей. Для конечного пользователя это означает максимальную простоту, достаточно запустить один файл. Кроме того, C++ предоставляет мощные и прямые средства для низкоуровневого взаимодействия с операционной системой, что критически важно для стабильной и предсказуемой работы с аппаратными ресурсами, такими как COM-порт.

В качестве интегрированной среды разработки была выбрана Microsoft Visual Studio 2022. Являясь стандартом для разработки на C++ под Windows, Visual Studio предоставляет не только редактор кода, но и отладчик, который незаменим при отладке протоколов обмена данными, а также инструменты для управления проектом.

2.2.7 Выбор СУБД

Для решения задачи долговременного хранения данных, получаемых от аппаратного модуля, требуется СУБД. При выборе СУБД для встраиваемых и локальных приложений, каким является разрабатываемый прототип DAQ-системы, рассматриваются два основных подхода: использование клиент-серверной СУБД или встраиваемой СУБД.

Клиент-серверные СУБД являются мощными, масштабируемыми системами, предназначенными для обслуживания множества клиентов в сети. Однако их использование для локального приложения неоправданно. Оно требует установки и настройки отдельного серверного процесса, что усложняет развёртывание и использование конечного приложения, а также потребляет дополнительные системные ресурсы.

В связи с этим, для данного проекта была выбрана встраиваемая СУБД

SQLite. SQLite представляет собой программную библиотеку, которая реализует полнофункциональный SQL-движок. В отличие от клиент-серверных СУБД, SQLite не требует отдельного серверного процесса и хранит всю базу данных в одном единственном файле на диске.

Такой подход имеет несколько ключевых преимуществ для нашей задачи. Он обеспечивает простоту развёртывания, так как для работы с базой данных достаточно лишь подключить библиотеку SQLite к C++ проекту, а само приложение будет автоматически создавать и работать с файлом. Кроме того, SQLite не требует администрирования и обладает транзакционным механизмом, гарантирующим целостность данных даже в случае сбоев. Наконец, популярность формата SQLite обеспечивает наличие множества сторонних графических утилит, таких как DB Browser for SQLite, которые позволяют легко просматривать и анализировать содержимое файла базы данных на этапе разработки, отладки и последующего анализа накопленной информации. Все эти факторы делают SQLite идеальным выбором для локальных приложений, где требуется надёжное хранение структурированных данных без усложнения архитектуры.

2.3 Установка и настройка рабочих сред

Для реализации программной части проекта требуется предварительная установка и настройка интегрированных сред разработки как для ПЛИС, так и для ПК.

2.3.1 Установка и настройка Xilinx ISE 14.7

Для начала установки необходимо скачать дистрибутив Xilinx ISE 14.7 с официального сайта. Особенностью данной версии является её работа внутри виртуальной среды. При установке Xilinx ISE 14.7 установщик предлагает установить VirtualBox автоматически, однако он использует устаревшую версию, что может привести к проблемам совместимости с современным оборудованием и драйверами.

В связи с этим было принято решение установить более новую и стабильную

версию ПО для виртуализации вручную. Для этого с официального сайта был загружен установщик VirtualBox 7.0.14. Подробный процесс установки представлен на рисунках 12 – 17.



Рисунок 12 – Окно приветствия мастера установки

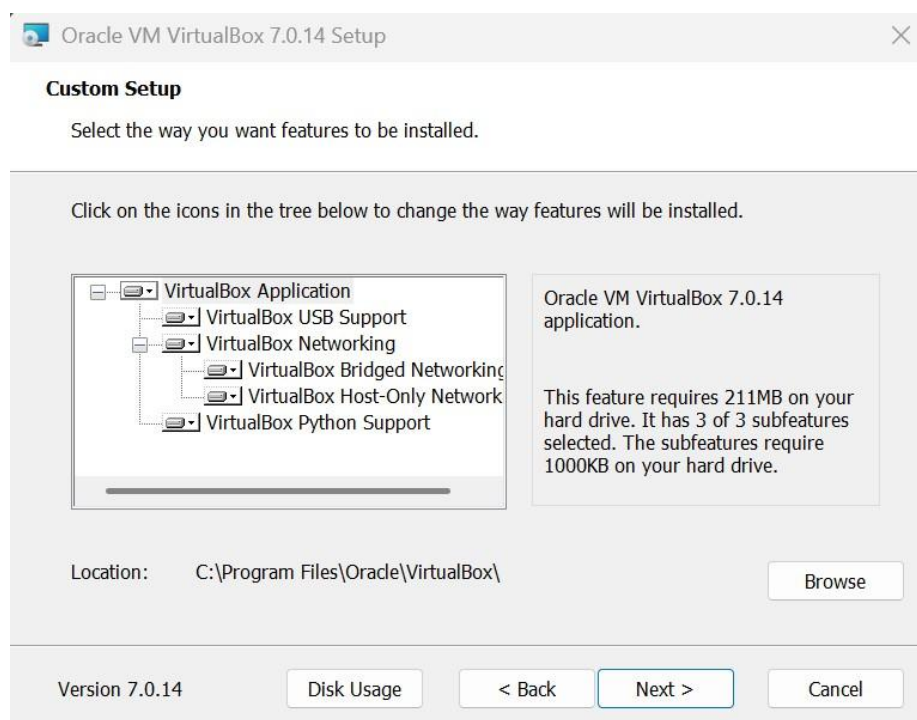


Рисунок 13 – Выбор компонентов и указание пути для установки



Рисунок 14 – Предупреждение о временном отключении сетевых соединений при установке сетевых интерфейсов

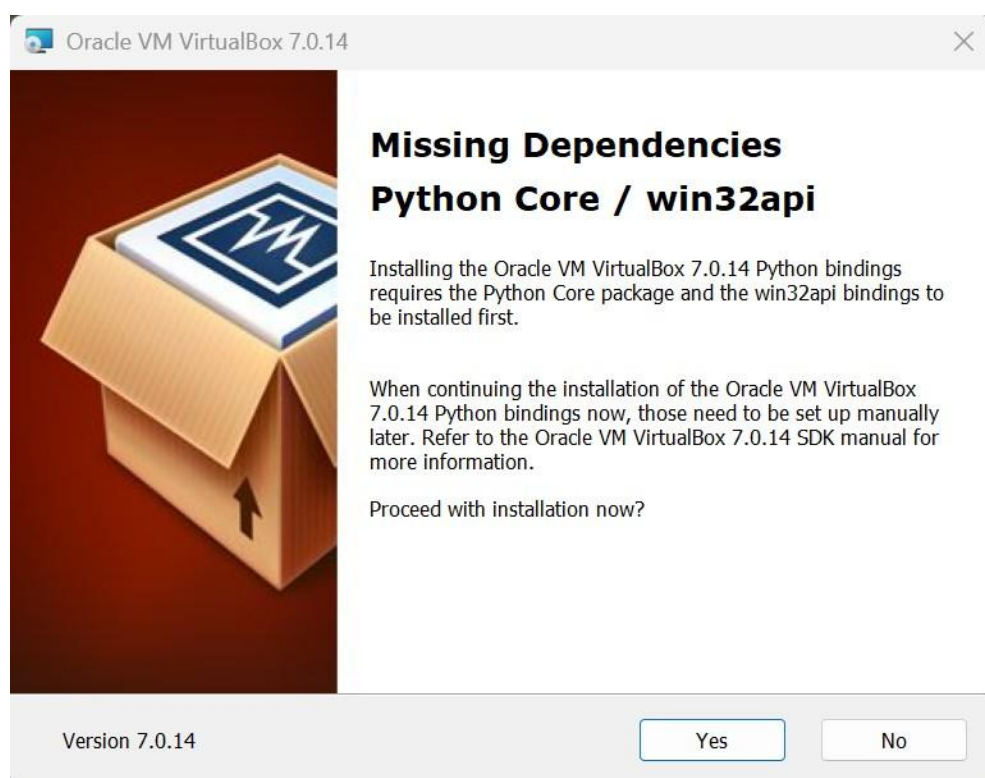


Рисунок 15 – Уведомление о необходимости установки недостающих зависимостей

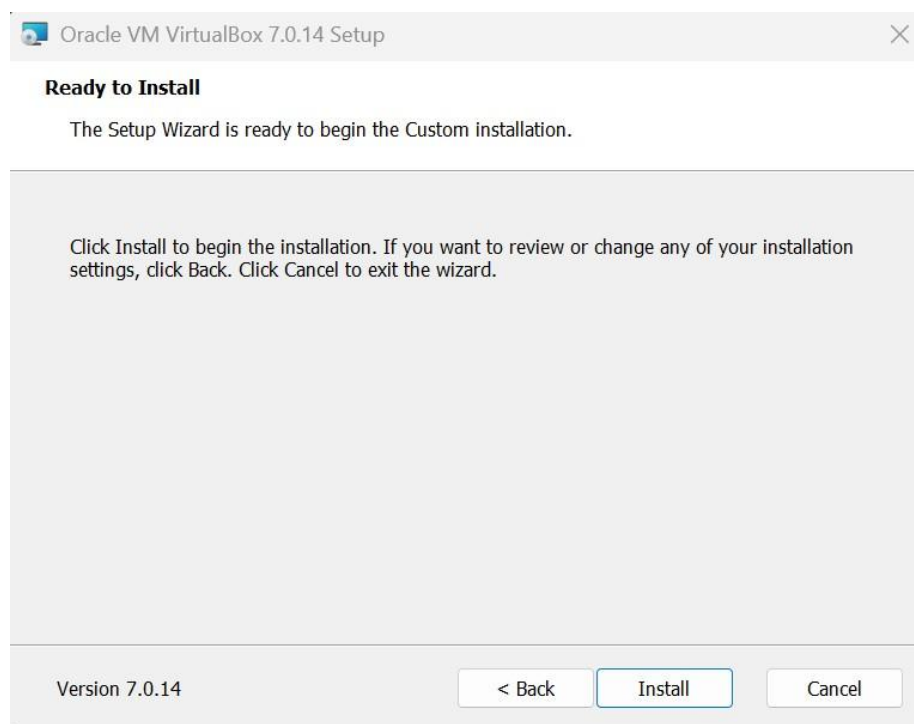


Рисунок 16 – Подтверждение готовности к началу установки



Рисунок 17 – Уведомление о завершении установки

После завершения инсталляции VirtualBox 7.0.14 выполняется установка самого программного комплекса Xilinx ISE 14.7. Подробный процесс установки представлен на рисунках 18 – 24.

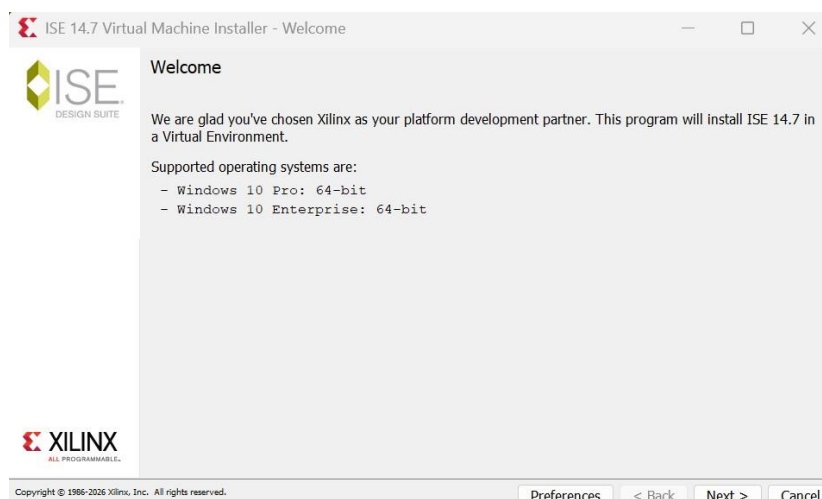


Рисунок 18 – Окно приветствия мастера установки

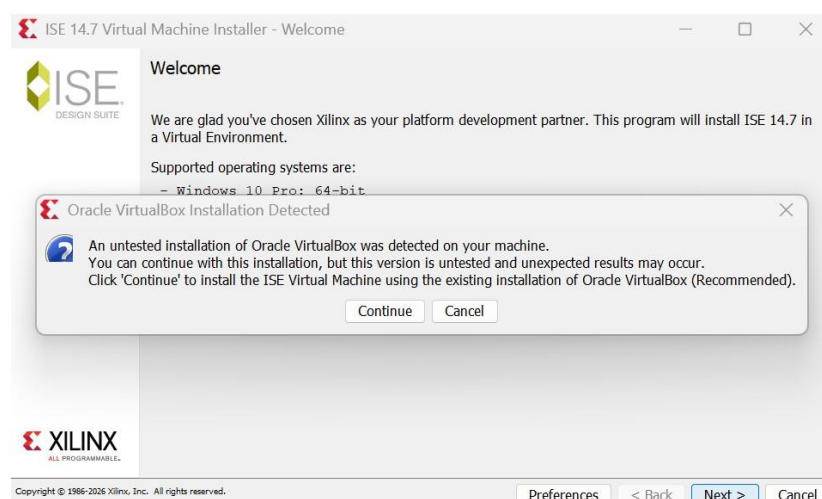


Рисунок 19 – Подтверждение использования уже установленной версии
VirtualBox

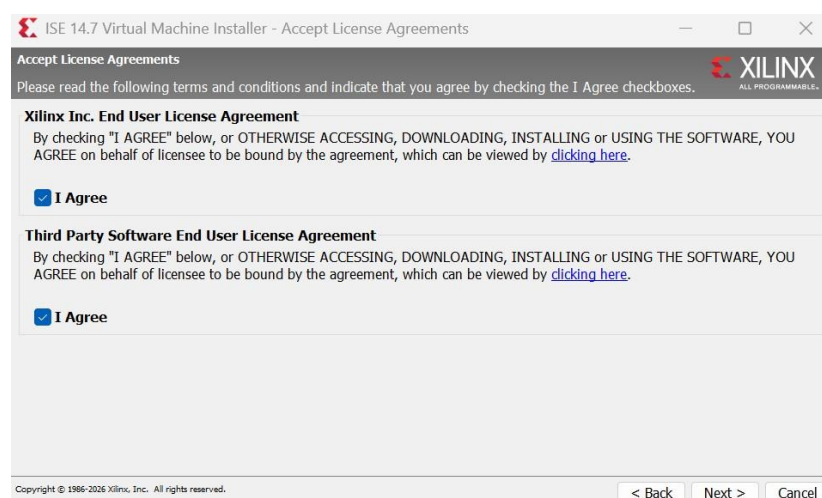


Рисунок 20 – Принятие условий лицензионных соглашений

					ПГУ 1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		28



Рисунок 21 – Выбор инструментов для установки

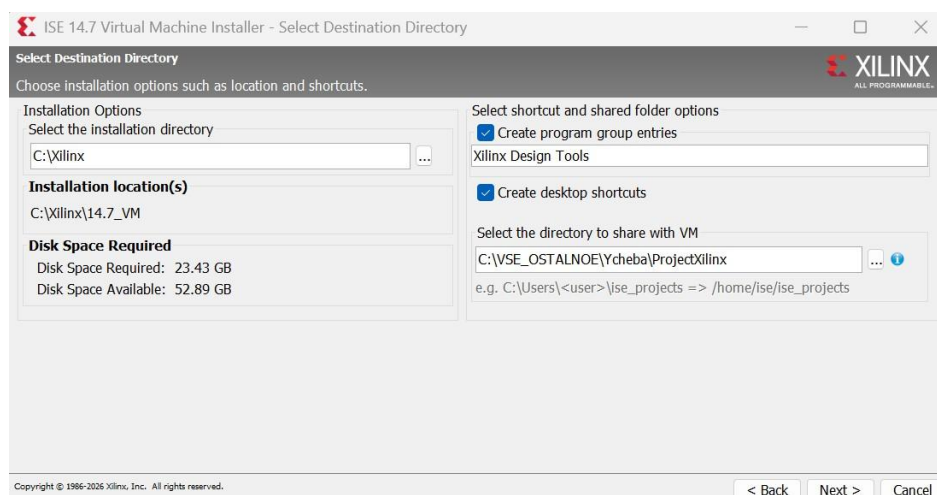


Рисунок 22 – Выбор директории для установки и настройка совместной папки



Рисунок 23 – Краткое описание установки

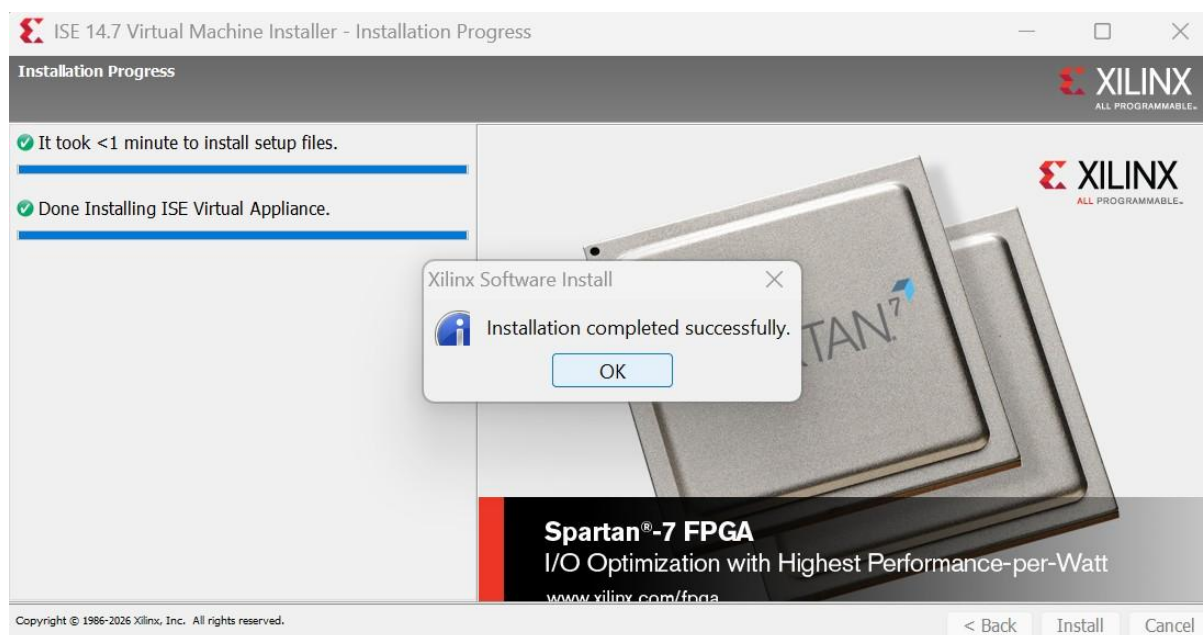


Рисунок 24 – Уведомление о завершении установки

По окончании установки Xilinx ISE 14.7 необходимо перейти к созданию и настройке проекта. Подробная настройка проекта представлена на рисунках 25 – 30.

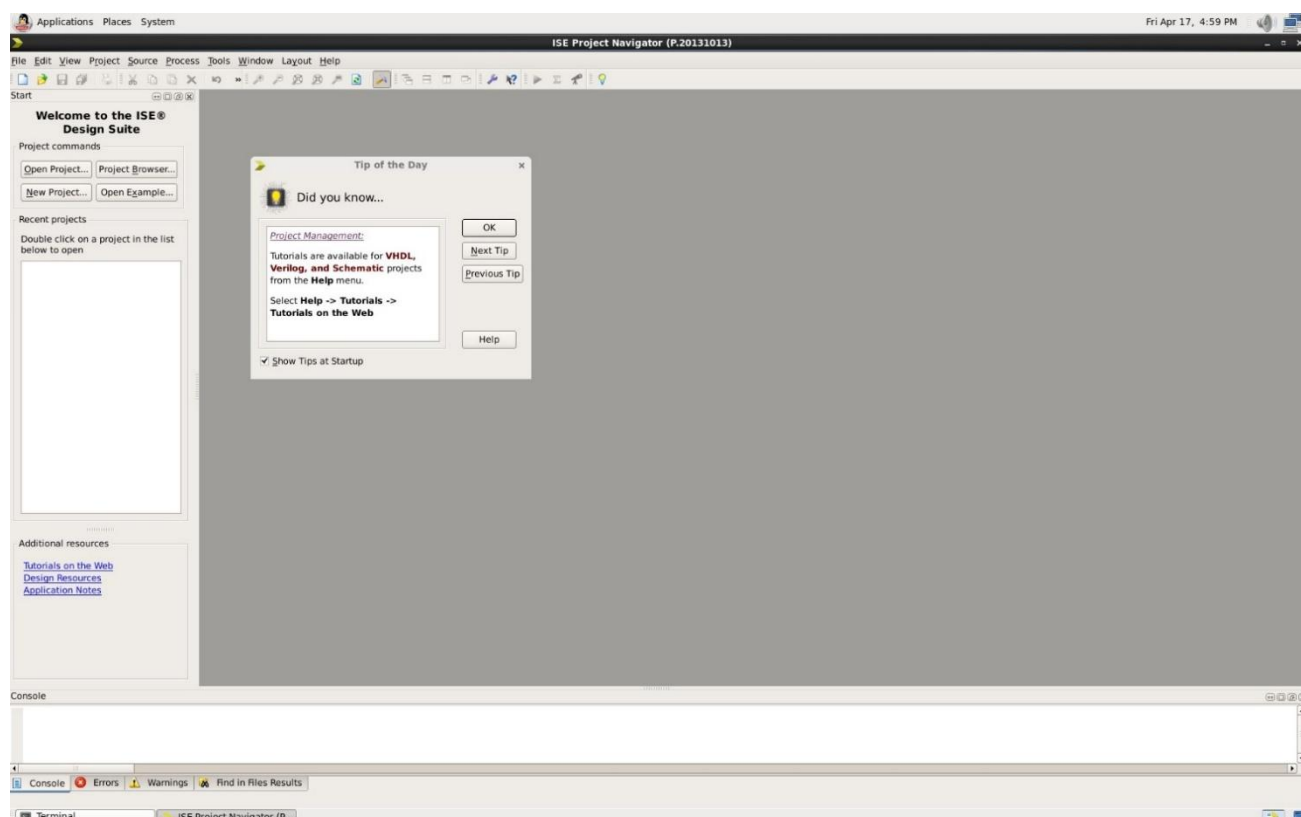


Рисунок 25 – Начальное окно ISE

Рисунок 26 – Создание нового проекта

Рисунок 27 – Указание местоположения и названия проекта

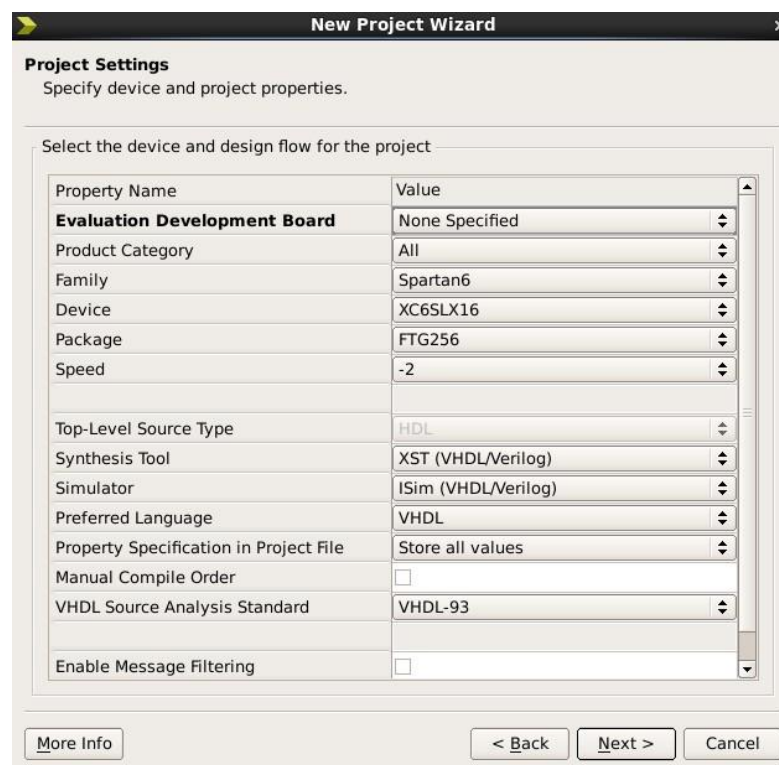


Рисунок 28 – Выбор устройства и схемы проектирования для проекта

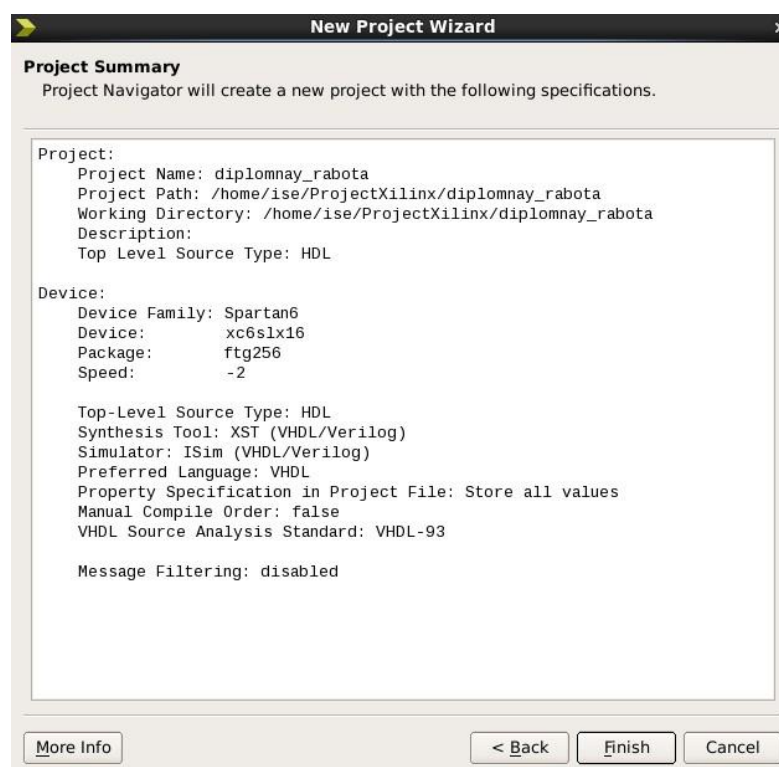


Рисунок 29 – Краткое описание проекта

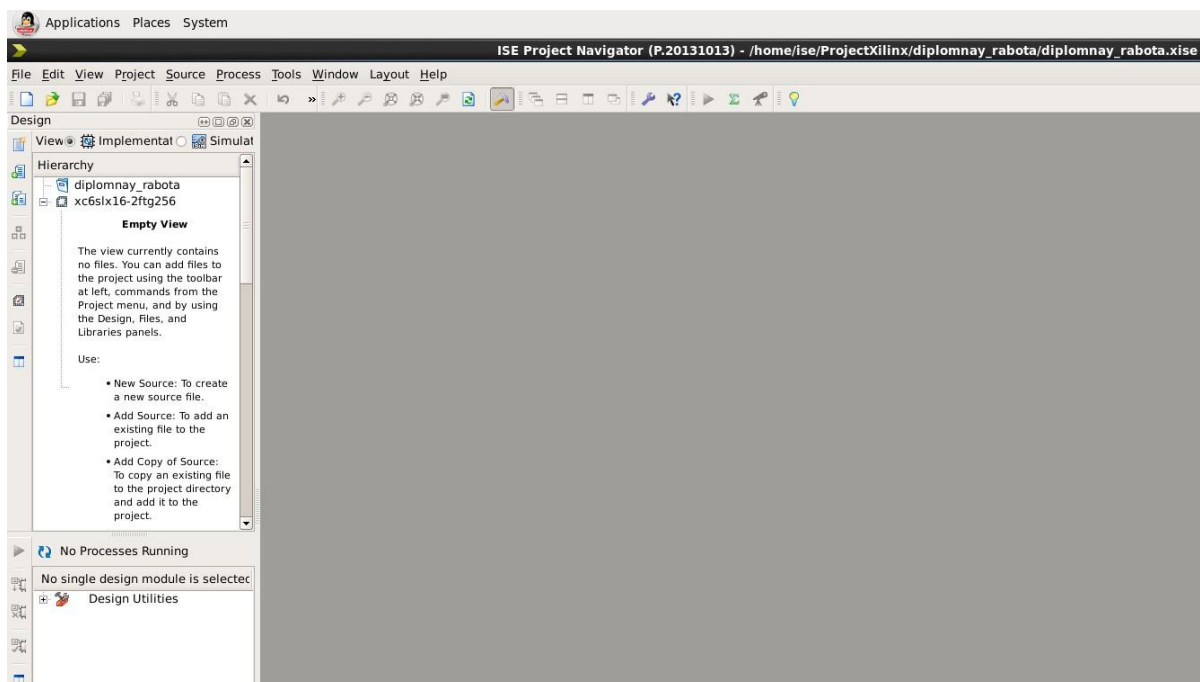


Рисунок 30 – Готовый проект

Так же для обеспечения связи между ПК и отладочной платой требуется установка соответствующих драйверов. Необходимо установить два основных драйвера: драйвер для работы COM-порта и драйвер кабеля JTAG. Подробная установка данных компонентов представлены на рисунках 31 – 39.

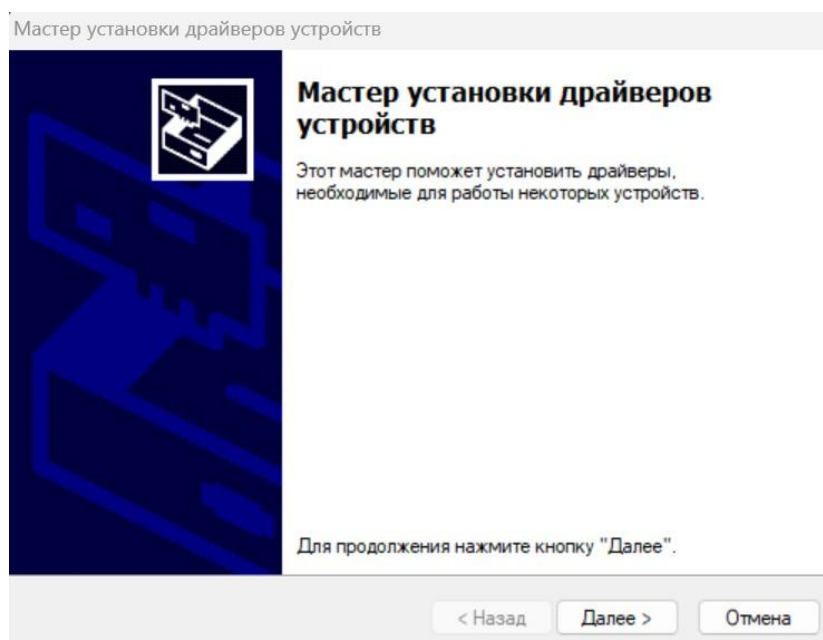


Рисунок 31 – Начальное окно мастера установки драйвера для COM-порта

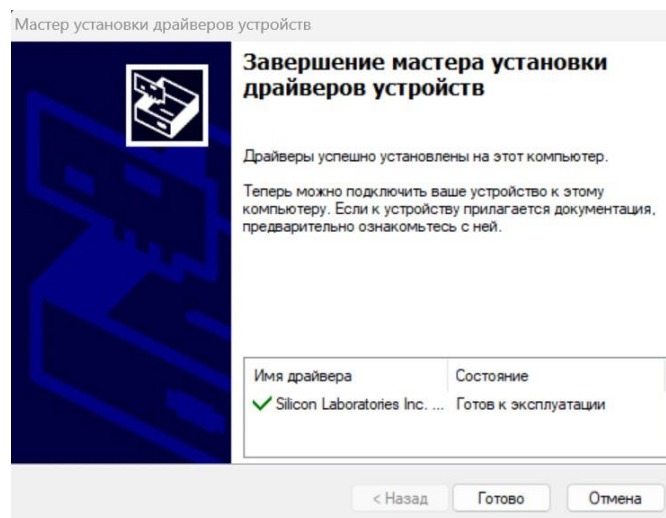


Рисунок 32 – Уведомление о завершении установки драйвера для СОМ-порта

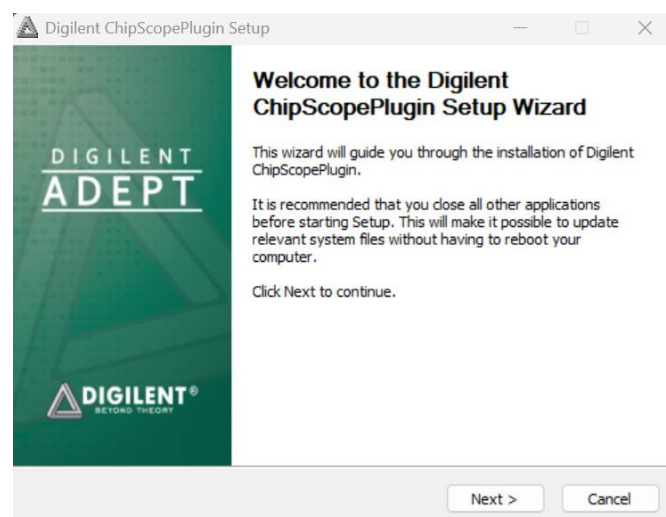


Рисунок 33 – Окно приветствия мастера установки драйвера кабеля JTAG

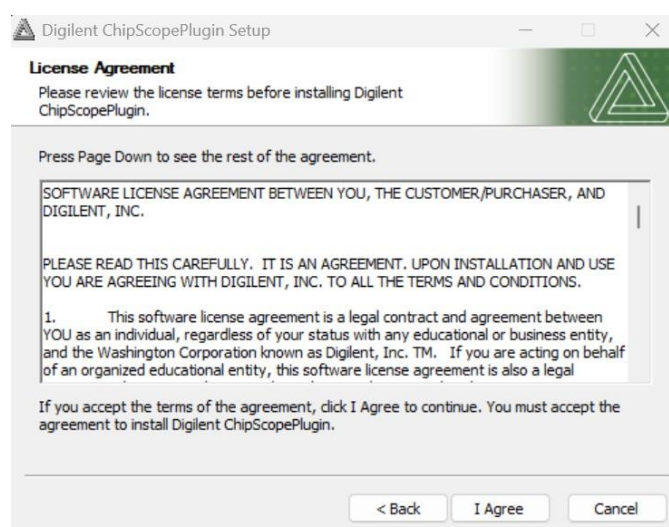


Рисунок 34 – Принятие условий лицензионного соглашения

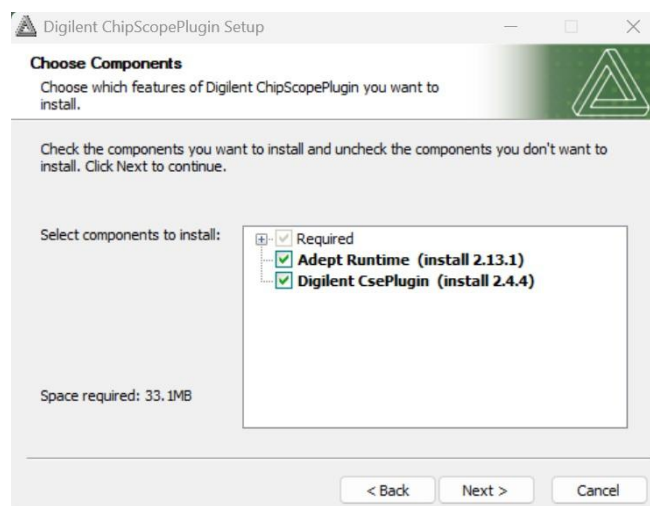


Рисунок 35 – Выбор компонентов для установки

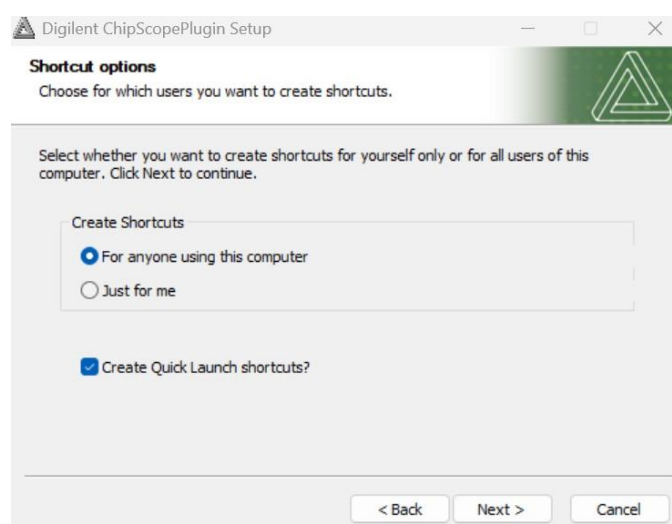


Рисунок 36 – Настройка быстрого доступа

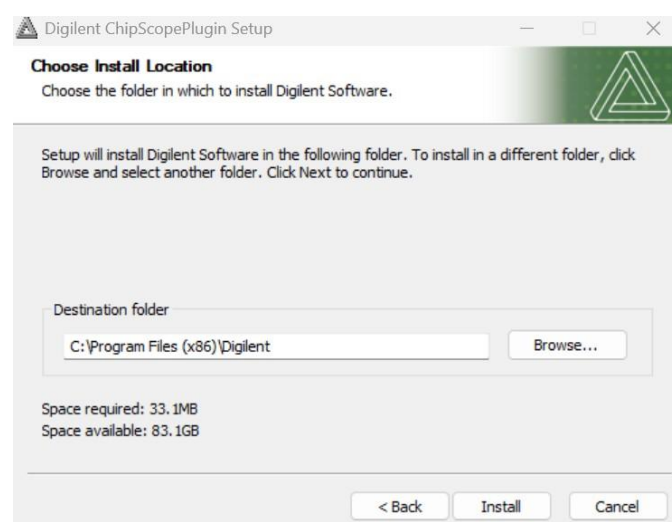


Рисунок 37 – Указание пути для установки

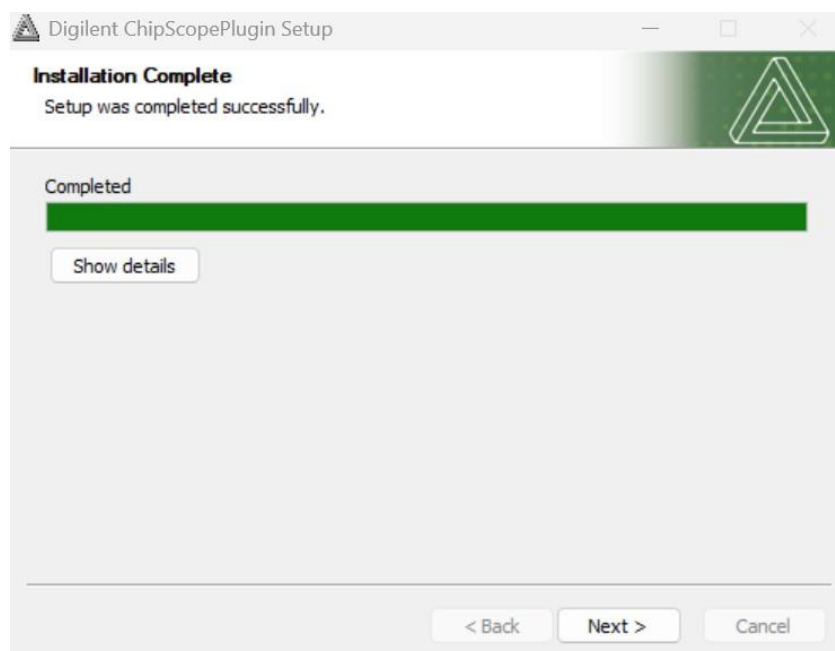


Рисунок 38 – Уведомление о завершении установки

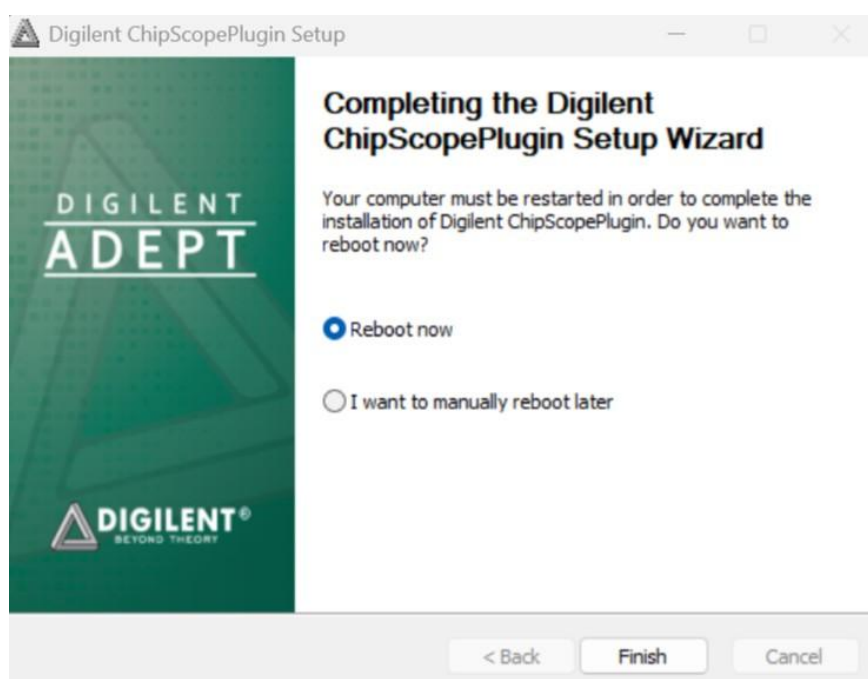


Рисунок 39 – Завершение работы мастера установки и выбор перезагрузки ПК

2.3.2 Установка и настройка Visual Studio 2022

Для начала установки необходимо скачать установщик Visual Studio Installer с официального сайта. Выбор был сделан в пользу Visual Studio 2022, так как новая версия Visual Studio 2026 вышла относительно недавно и может быть не стабильна. Подробный процесс установки представлен на рисунках 40 – 43.

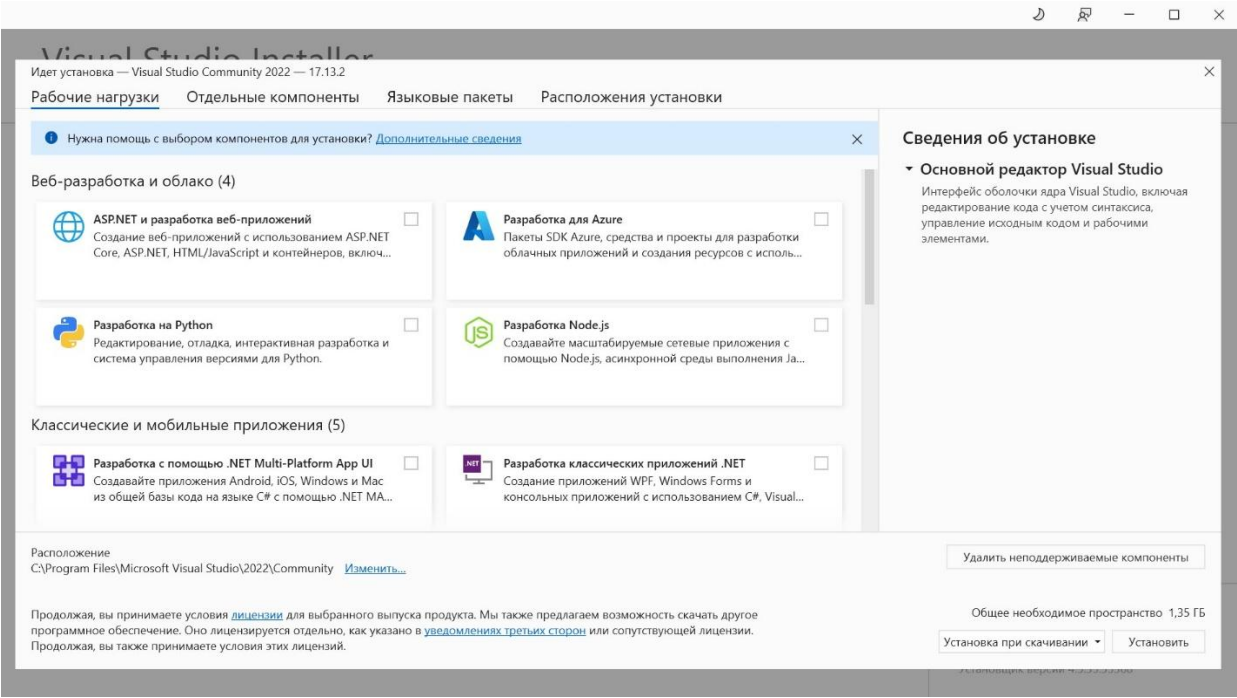


Рисунок 40 – Начальное окно установщика

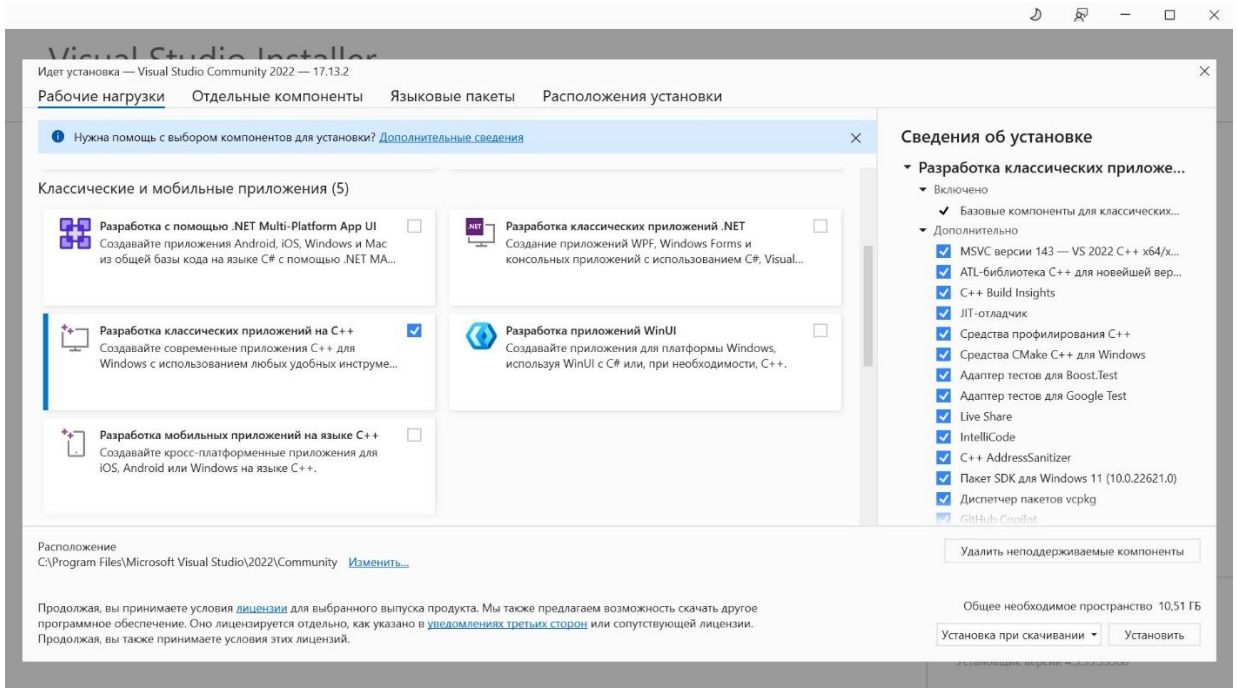


Рисунок 41 – Выбор компонентов для установки

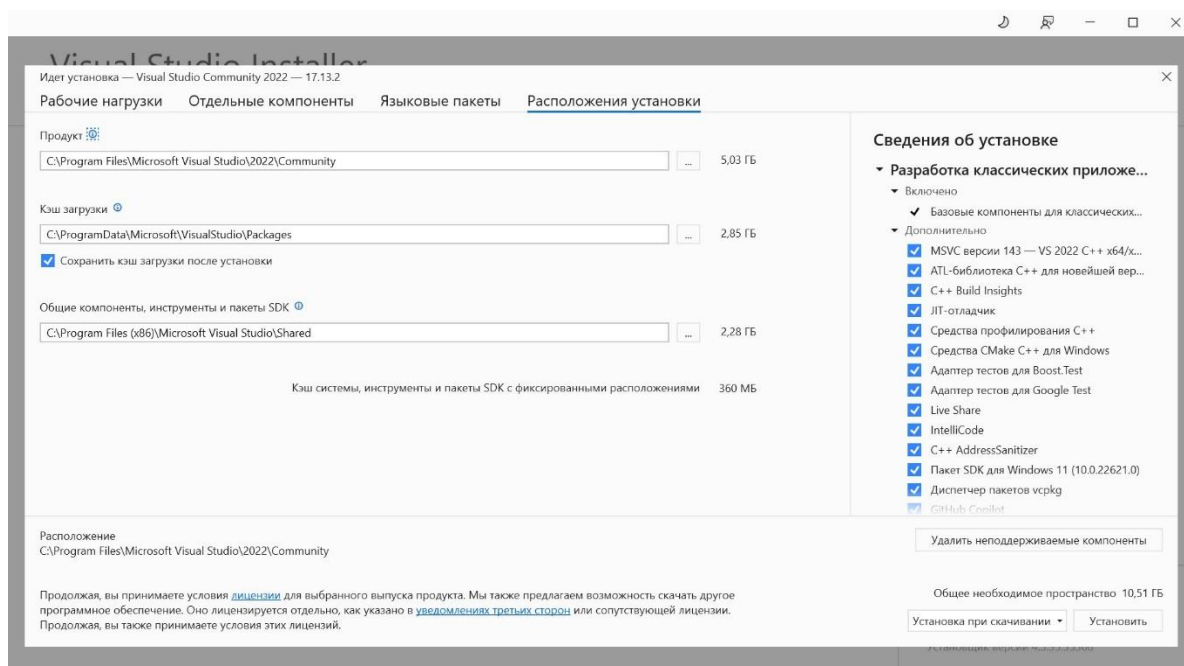


Рисунок 42 – Выбор расположения для установки

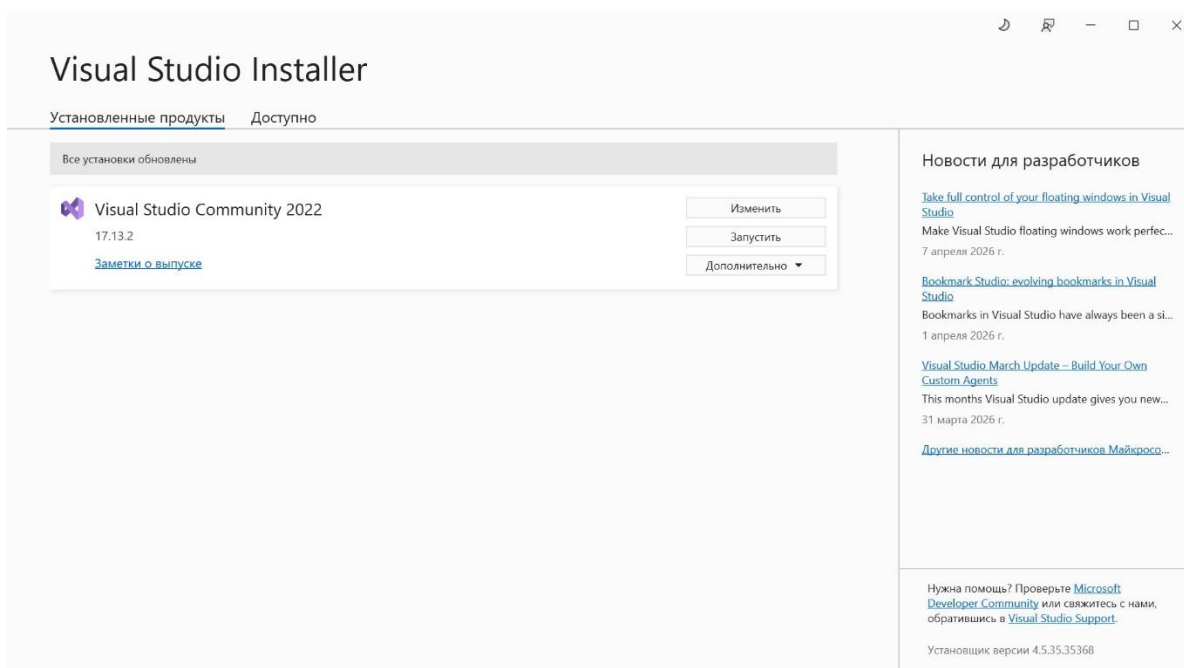


Рисунок 43 – Завершение установки

После завершения установки Visual Studio 2022 необходимо перейти к созданию и настройке проекта. Для обеспечения работы с данными в проекте используется встраиваемая СУБД SQLite. Поскольку стандартные библиотеки Visual Studio не включают средства для работы с данной БД, необходимо скачать

исходные файлы SQLite с официального сайта. Подробная настройка проекта представлена на рисунках 44 – 51.

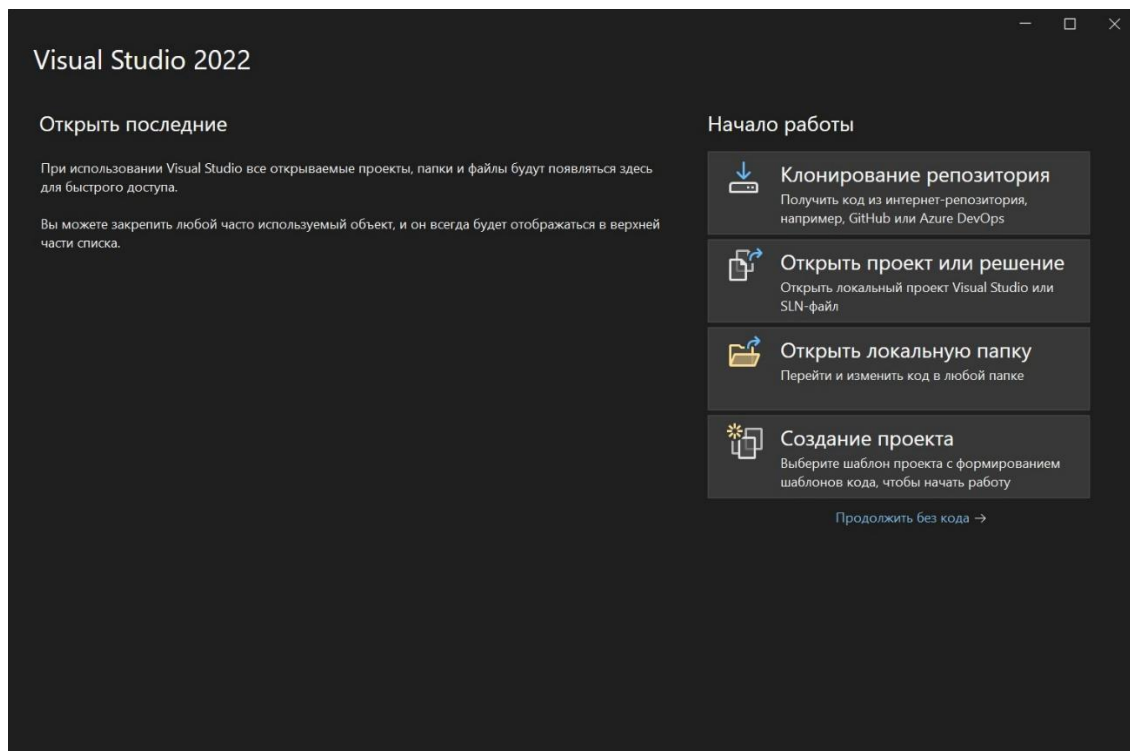


Рисунок 44 – Начальное окно Visual Studio 2022

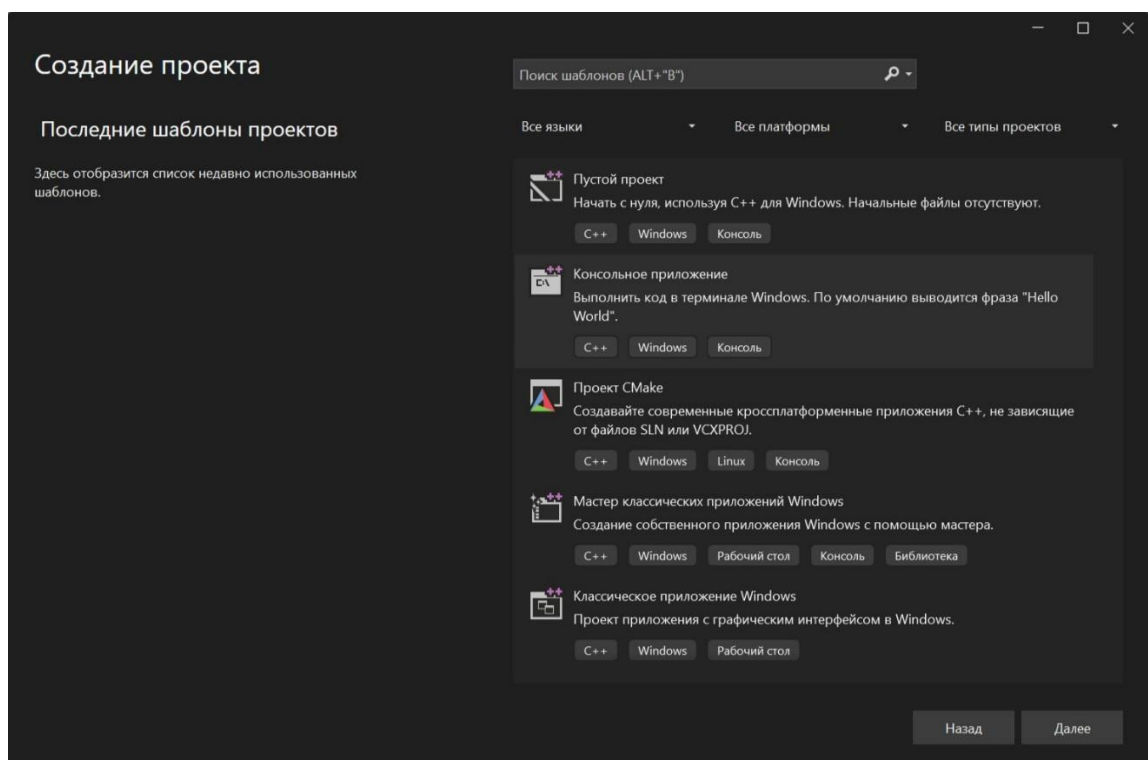


Рисунок 45 – Выбор шаблона для проекта

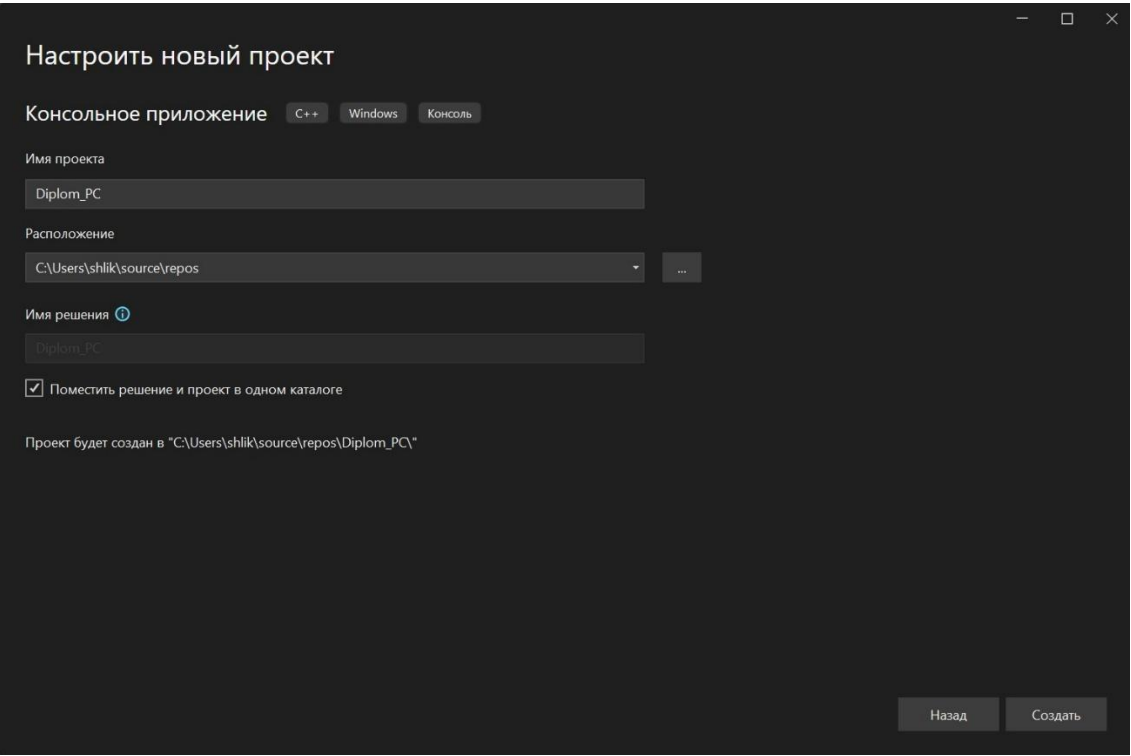


Рисунок 46 – Указание расположения и названия проекта

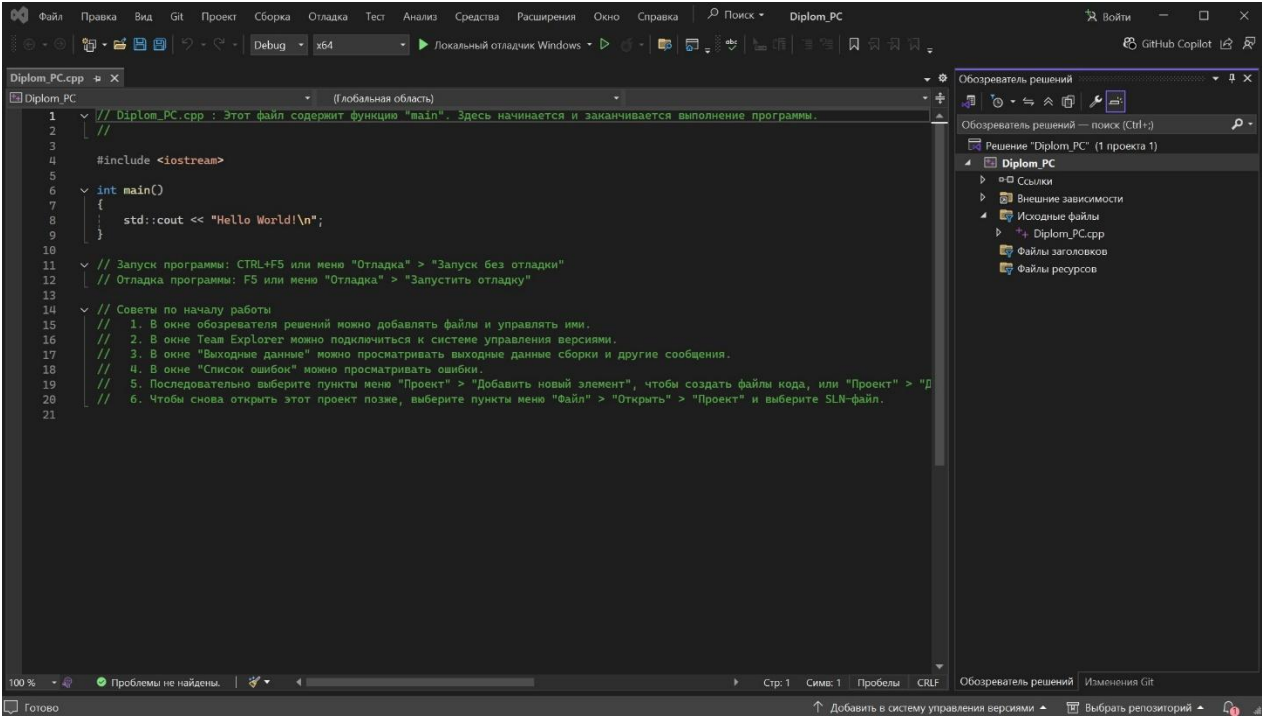


Рисунок 47 – Созданный проект

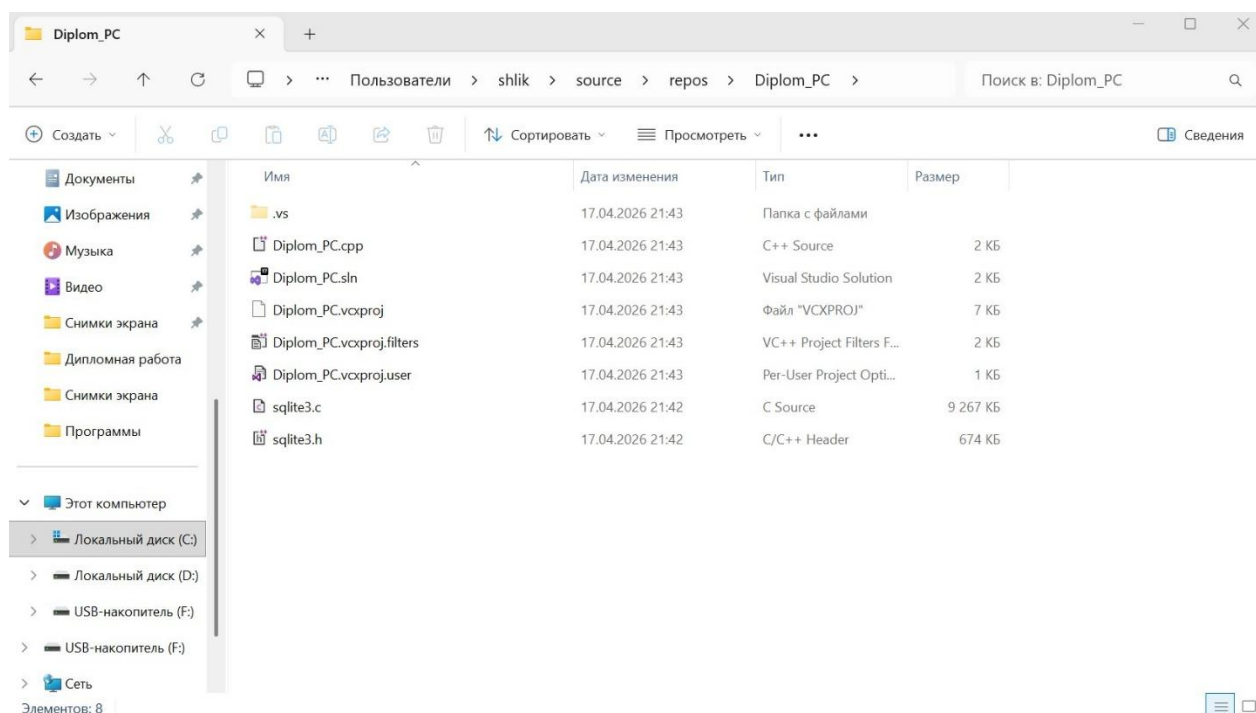


Рисунок 48 – Копирование в папку с проектом исходного и заголовочного файла для работы с БД SQLite

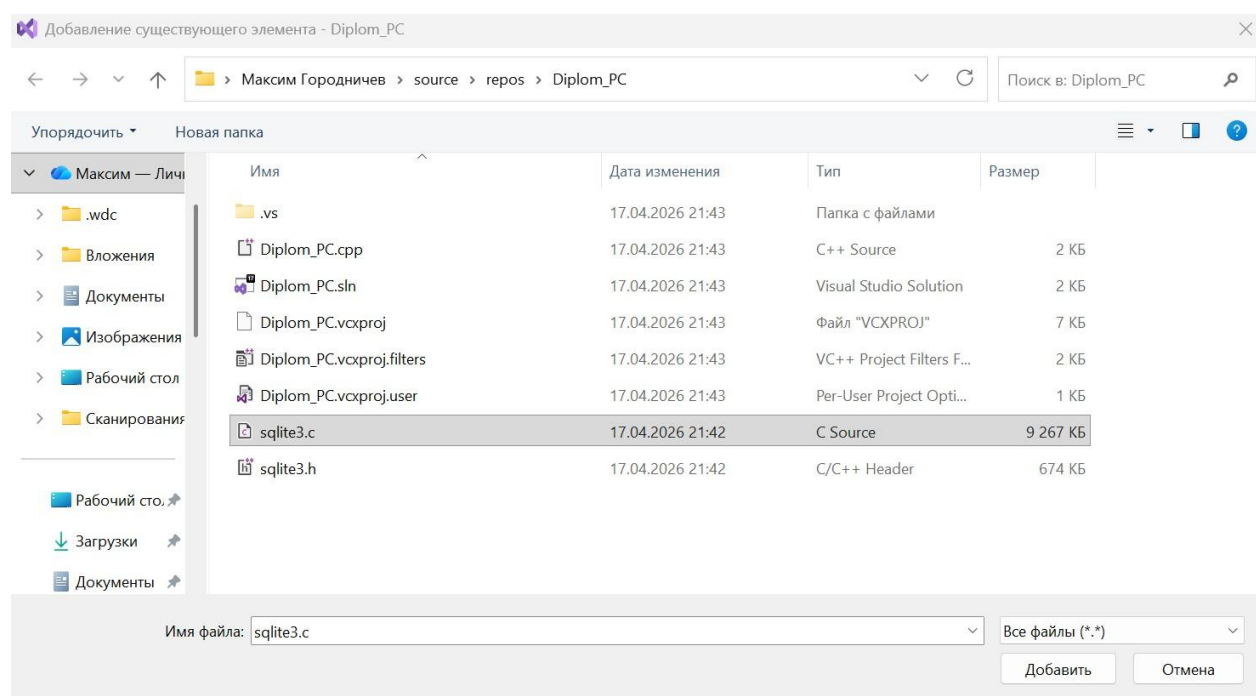


Рисунок 49 – Добавление исходного файла в проект

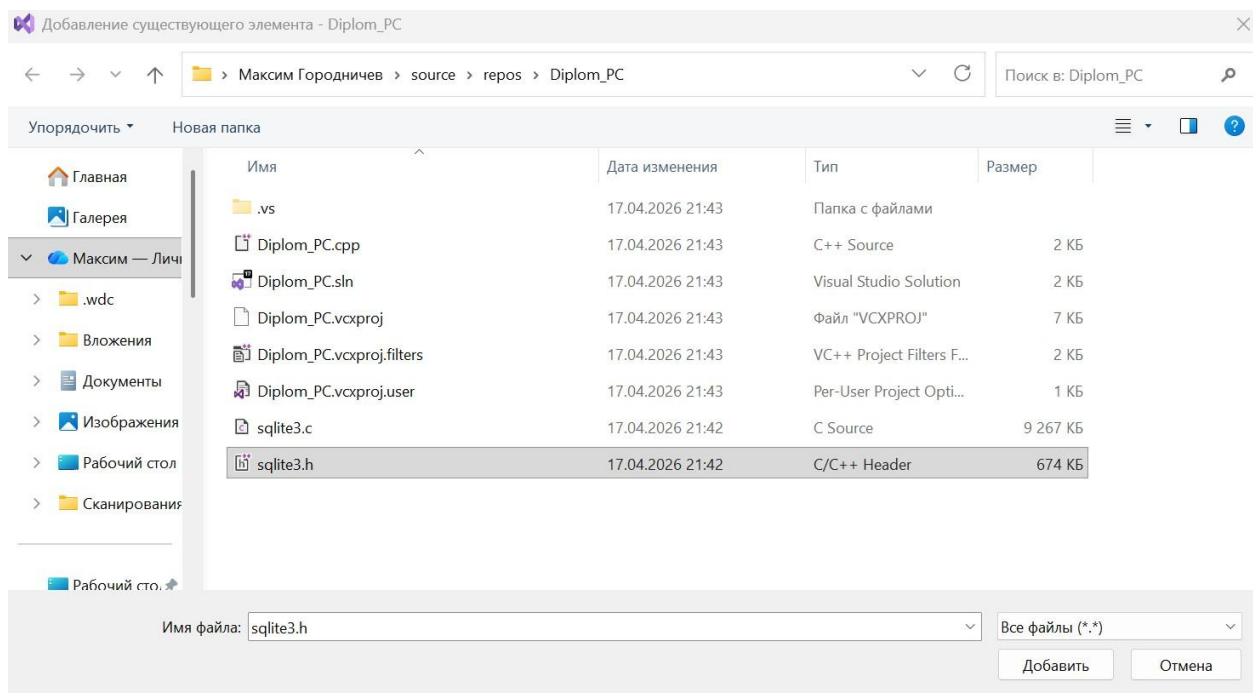


Рисунок 50 – Добавление заголовочного файла в проект

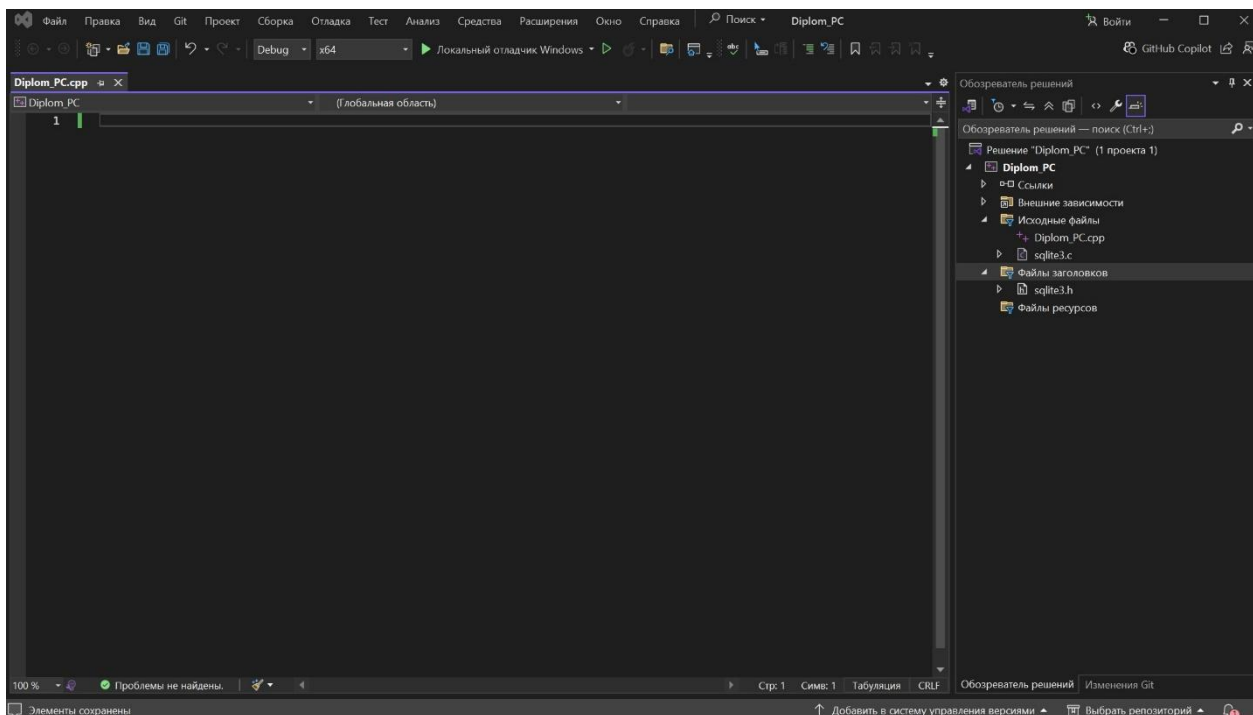


Рисунок 51 – Готовый проект

3 Реализация и тестирование системы

3.1 Структура БД

Вся информация о проведённых измерениях сохраняется в единственной таблице с именем SensorLog. Структура БД позволяет фиксировать численные показатели физических величин и контекстную информацию в виде порядкового номера запроса и точного времени его выполнения.

Приложение взаимодействует с БД в режиме реального времени. При получении и обработке каждой валидной строки данных от ПЛИС формируется запрос на добавление новой записи. Для оптимизации работы с дисковой подсистемой и предотвращения блокировок при записи используются специфические настройки журнала и синхронный режим работы.

В таблице SensorLog определены следующие поля:

- RequestNum – строковое поле для хранения порядкового номера измерения;
- DateTime – временная метка для фиксации даты и времени получения данных;
- Temperature(C) – числовое поле для хранения значения температуры;
- Pressure(Pa) – числовое поле для хранения значения давления;
- Humidity(%) – числовое поле для хранения значения влажности.

Графическое представление структуры используемой БД представлено на рисунке 52.

SensorLog
+ RequestNum
+ DateTime
+ Temperature(C)
+ Pressure(Pa)
+ Humidity

Рисунок 52 – Структура БД

3.2 Описание аппаратной части

Аппаратная часть представляет стенд, состоящий из ПК, разработанной и изготовленной макетной платы, покупной платы ALINX AX309 и датчиков АНТ21 и ВМР180. Электрическая схема стенда представлена в приложении Б.1.

Основой комплекса является отладочная плата AX309. Для вывода периферийных сигналов на плате предусмотрены разъёмы расширения. В рамках данного проекта используется соединитель «EXPANSION PORT J2». Связь отладочной платы с ПК для обмена данными и получения электропитания осуществляется с помощью кабеля USB, подключённого к встроенному мосту USB-UART [3,4].

Для коммутации датчиков разработана и изготовлена макетная соединительная плата. Внешний вид макетной платы представлен на рисунке 53.

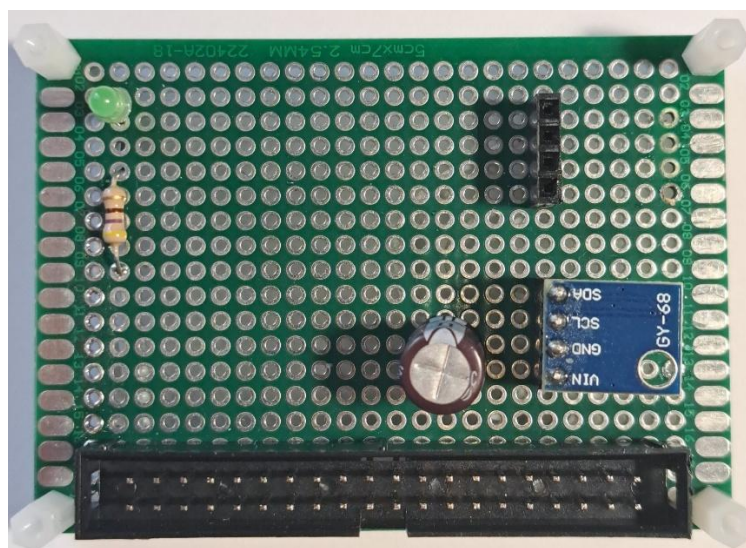


Рисунок 53 – Внешний вид макетной платы

Для коммутации макетной платы с отладочной платой AX309 используется соединитель ВН-40, к которому подключается плоский кабель. Со стороны порта расширения задействованы контакты 33, 35, 37 и 39 соединителей ВН [4].

Взаимодействие с датчиками реализовано через розетки PBS-4. Такое конструктивное решение повышает ремонтопригодность системы и позволяет легко заменять датчики при необходимости изменения конфигурации или в случае

выхода их из строя.

На макетной плате смонтированы цепи питания и светодиодный индикатор, обеспечивающий индикацию подачи питания. Для фильтрации помех и стабилизации напряжения питания в схему интегрирован электролитический фильтрующий конденсатор. Соединения выполнены монтажным проводом МГТФ-0,5 методом пайки. Вид со стороны монтажа представлен на рисунке 54.

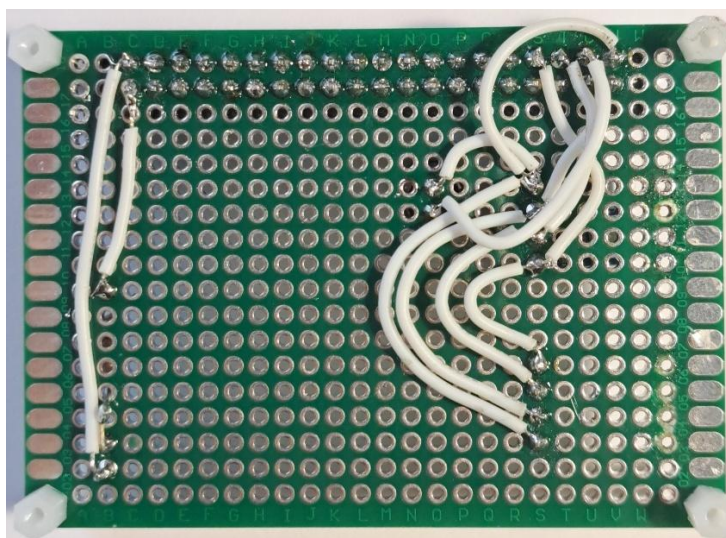


Рисунок 54 – Вид со стороны монтажа

3.3 Разработка ПО для ПЛИС

ПО для ПЛИС разработано на языке описания аппаратуры VHDL с применением структурного нисходящего подхода. Главным иерархическим элементом проекта выступает модуль верхнего уровня, который выполняет функцию объединения базовых функциональных подсистем в единую вычислительную структуру и задаёт внешний интерфейс связи ПЛИС с аппаратной периферией.

В блоке описания сущности модуля верхнего уровня задаются глобальные параметрические константы, определяющие временные характеристики разрабатываемой системы: частоту основного тактового генератора, равную 50 МГц, скорость передачи данных по интерфейсу UART, установленную на уровне 115200 бод, и рабочую частоту шины I2C, равную 400 кГц. Также в данном блоке

определяются порты ввода-вывода ПЛИС. К ним относятся системные линии тактирования и аппаратного сброса, линии асинхронного приёмопередатчика, контакты двунаправленной шины I2C для связи с датчиками, а также выводы интерфейса для интеграции с модулем часов реального времени DS1302. Фрагмент кода, описывающий интерфейс модуля верхнего уровня, представлен ниже.

```
entity top is
  generic (
    CLK_FREQ      : integer := 50_000_000;
    BAUD_RATE     : integer := 115_200;
    I2C_RATE      : integer := 400_000;
    USE_PARITY    : boolean := false;
    EVEN_PARITY   : boolean := false
  );
  port (
    clk           : in std_logic;
    button_rst    : in std_logic;

    rx_serial     : in  std_logic;
    tx_serial     : out std_logic;

    scl           : inout std_logic;
    sda           : inout std_logic;

    rtc_sclk      : out  std_logic;
    rtc_ce        : out  std_logic;
    rtc_io        : inout std_logic
  );
end top;
```

Внутренняя архитектура модуля верхнего уровня не содержит сложной поведенческой логики обработки данных, выступая преимущественно в роли структурной оболочки. Единственным функциональным процессом на данном уровне является аппаратное защёлкивание и инвертирование сигнала с кнопки платы. Это необходимо для устранения возможного метастабильного состояния и формирования надёжного синхронного сигнала глобального сброса для всех последующих подсистем. Фрагмент кода, реализующий безопасную обработку сигнала аппаратного сброса, представлен ниже.

```
button_reset_proc : process (clk)
begin
  if rising_edge(clk) then
```

```

        button_rst1 <= button_rst;
        rst <= not button_rst1;
    end if;
end process button_reset_proc;

```

Основное тело архитектуры представляет собой инициализацию и связывание между собой модуля связи с ПК по интерфейсу UART и центрального модуля управления комплексом. Данные подсистемы соединяются посредством внутренних шин данных шириной в один байт и управляющих флагов состояний.

Для организации надёжного двустороннего обмена данными между аппаратным комплексом и ПК был разработан модуль UART. На данном иерархическом этапе программная реализация представляет собой структурный модуль-оболочку, инкапсулирующую в себе независимые подсистемы приёма и передачи информации. Данный модуль предоставляет центральному контроллеру единый стандартизированный интерфейс для работы с последовательной линией связи.

Параметризация приёмопередатчика осуществляется через блок обобщений, что позволяет гибко настраивать системную тактовую частоту, скорость передачи данных в бодах, а также наличие и тип бита чётности без вмешательства в базовый код конечных автоматов. Эти конфигурационные параметры принимаются от модуля верхнего уровня и транслируются ниже по иерархии при инициализации внутренних компонентов. Фрагмент кода, описывающий внешний интерфейс объединяющего модуля UART, представлен ниже.

```

entity uart is
    generic (
        CLK_FREQ      : integer;
        BAUD_RATE      : integer;
        USE_PARITY      : boolean;
        EVEN_PARITY     : boolean
    );
    port (
        clk            : in  std_logic;
        rst            : in  std_logic;

        rx_serial      : in  std_logic;
        tx_serial      : out std_logic;

        rx_data        : out std_logic_vector(7 downto 0);
    );
end entity;

```

```

        rx_done      : out std_logic;

        tx_start     : in  std_logic;
        tx_data      : in  std_logic_vector(7 downto 0);
        tx_busy      : out std_logic
    );
end uart;

```

Внутренняя архитектура описываемого модуля не содержит собственной поведенческой логики. Её основная задача – структурное объявление и коммутация приёмника и передатчика. Физические однобитные линии приёма и передачи напрямую выводятся на порты ПЛИС, а внутренний обмен с модулем управления происходит по параллельным восьмибитным шинам данных с использованием аппаратных флагов старта, готовности принятого байта и занятости линии.

Внутренняя архитектура подсистемы приёмника базируется на синхронном конечном автомате, обеспечивающем надёжное извлечение данных из асинхронного последовательного потока. Для повышения помехоустойчивости интерфейса алгоритм осуществляет выборку значения логического уровня линии строго в середине битового интервала. Требуемая временная задержка формируется за счёт использования внутреннего счётчика, предельное значение которого вычисляется аппаратно на этапе компиляции как отношение тактовой частоты генератора к заданной скорости передачи.

Конечный автомат приёмника состоит из пяти состояний: ожидание, приём старт-бита, чтение данных, проверка бита чётности и валидация стоп-бита. После обнаружения спада напряжения на линии и подтверждения его стабильности, модуль последовательно считывает восемь бит информационной нагрузки в промежуточный сдвиговый регистр. При активации функции контроля чётности автомат вычисляет контрольную сумму принятого пакета с использованием встроенной функции `xor_reduce` и сверяет её с принятым битом чётности. В случае успешной валидации стоп-бита высокого уровня модуль формирует одиночный синхронный импульс `rx_done`, сигнализирующий о готовности принятого байта на выходной параллельной шине. Фрагмент кода, реализующий логику выборки

старт-бита и начала приёма данных, представлен ниже.

```
when START =>
  if baud_cnt = SAMPLE_TIME - 1 then
    baud_cnt <= 0;
    if rx_serial = '0' then
      bit_idx <= 0;
      state <= DATA;
      shift_reg <= (others => '0');
    else
      state <= IDLE;
    end if;
  else
    baud_cnt <= baud_cnt + 1;
  end if;
```

Логика работы подсистемы передатчика построена на аналогичном синхронном конечном автомате, обеспечивающем независимую отправку информации в последовательную линию. Процесс передачи инициируется модулем верхнего уровня путём выставления управляющего импульса tx_start и подачи байта данных на шину tx_data. При обнаружении стартового импульса автомат незамедлительно вычисляет значение бита чётности с помощью функции xor_reduce, выставляет флаг занятости для предотвращения перезаписи буфера и переходит к генерации битовой последовательности.

Алгоритм последовательно формирует на выходной линии tx_serial старт-бит низкого логического уровня, восемь бит информационной нагрузки, опциональный бит чётности и завершающий стоп-бит высокого уровня. Длительность удержания каждого бита на линии контролируется внутренним счётчиком, лимит которого рассчитывается исходя из заданных параметров тактовой частоты и скорости передачи. По завершении отсчёта длительности стоп-бита флаг занятости снимается, и автомат возвращается в состояние ожидания следующей команды. Фрагмент кода, реализующий логику перехода автомата передатчика из состояния ожидания в режим передачи, представлен ниже.

```
when IDLE =>
  tx_serial <= '1';
  tx_busy <= '0';
  parity_bit <= '0';
  if tx_start = '1' then
    p := xor_reduce(tx_data);
```

```

    if EVEN_PARITY then
        parity_bit <= p;
    else
        parity_bit <= not p;
    end if;
    baud_cnt <= 0;
    bit_idx <= 0;
    tx_busy <= '1';
    state <= START;
end if;

```

Логическим продолжением тракта передачи данных выступает центральный управляющий модуль, который берёт на себя функции аппаратной маршрутизации сигналов и арбитража доступа к интерфейсу связи. В иерархии проекта данный блок представляет собой промежуточное структурное звено. Архитектура модуля включает структурную инициализацию подсистемы декодирования пользовательских команд, генератора статического текстового интерфейса и контроллера взаимодействия с датчиками.

Собственная поведенческая логика управляющего модуля сводится к формированию комбинированного сигнала сброса и арбитражу выходной шины данных. Поскольку инициализировать отправку байтов в СОМ-порт ПК могут как генератор статического текста, так и контроллер датчиков, в архитектуре реализован синхронный аппаратный мультиплексор. Данный вычислительный процесс непрерывно отслеживает управляющие флаги готовности от обеих передающих подсистем и коммутирует соответствующие шины данных на вход передатчика UART, регламентируя очередность отправки и исключая возникновение коллизий на линии связи. Приоритет при отправке жёстко закреплён за контроллером датчиков. Если он выставляет флаг готовности, то текстовые сообщения переходят в режим ожидания.

Дополнительно в модуле предусмотрена логика комбинированного аппаратного сброса. При получении от пользователя команды сброса измерительная подсистема принудительно реинициализируется внутренним сигналом `rst_combined`, мгновенно прерывая текущие циклы работы интерфейса I2C и возвращая контроллер датчиков в исходное состояние. Фрагмент кода,

реализующий синхронный мультиплексор передачи данных, представлен ниже [5].

```
process (clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            con_tx_data <= (others => '0');
            con_tx_start <= '0';
        else
            if sensor_tx_start = '1' then
                con_tx_data <= sensor_tx_data;
                con_tx_start <= '1';
            elsif output_tx_start = '1' then
                con_tx_data <= output_tx_data;
                con_tx_start <= '1';
            else
                con_tx_start <= '0';
            end if;
        end if;
    end if;
end process;
```

Первая из упомянутых вложенных подсистем центрального контроллера реализована в виде аппаратного модуля обработки пользовательского ввода. Данный блок выполняет функцию командного декодера, транслирующего поступающие от приёмника UART «сырые» байты во внутренние управляющие сигналы измерительного комплекса.

Алгоритм работы модуля базируется на синхронном процессе, непрерывно отслеживающем флаг готовности принятого байта. При фиксации высокого логического уровня на соответствующей линии, поступающий восьмибитный вектор данных интерпретируется алгоритмом как символ в стандартной кодировке ASCII. В зависимости от принятого значения, процесс выставляет в высокий логический уровень строго один из выходных флагов запроса длительностью в один тактовый цикл. Для символа «1» формируется запрос на измерение температуры, для «2» – давления, для «3» – влажности, а для «4» – инициируется комплексный опрос всех доступных физических параметров. При получении символов «г» или «R» модуль выставляет сигнал сброса и принудительно обнуляет все остальные линии запросов. Во избежание ложных срабатываний, любые

нераспознанные символы игнорируются алгоритмом. Фрагмент кода, реализующий логику декодирования символьных команд и генерации управляющих импульсов, представлен ниже.

```

process (clk)
    variable cmd : character;
begin
    if rising_edge(clk) then
        if rst = '1' then
            bh_start_send <= '0';
            bh_temp_req <= '0';
            bh_pres_req <= '0';
            bh_hum_req <= '0';
            bh_all_req <= '0';
        else
            bh_start_send <= '0';
            bh_temp_req <= '0';
            bh_pres_req <= '0';
            bh_hum_req <= '0';
            bh_all_req <= '0';

            if bh_rx_done = '1' then
                cmd :=
character'val(to_integer(unsigned(bh_rx_data)));

                case cmd is
                    when '1' =>
                        bh_temp_req <= '1';

                    when '2' =>
                        bh_pres_req <= '1';

                    when '3' =>
                        bh_hum_req <= '1';

                    when '4' =>
                        bh_all_req <= '1';

                    when 'r' | 'R' =>
                        bh_start_send <= '1';
                        bh_temp_req <= '0';
                        bh_pres_req <= '0';
                        bh_hum_req <= '0';
                        bh_all_req <= '0';

                    when others =>
                        NULL;
                end case;
            end if;
        end if;
    end if;
end if;

```

```

    end if;
end process;

```

Вторая из упомянутых вложенных подсистем центрального контроллера предназначена для генерации статического текстового интерфейса пользователя, отправляемого в терминал персонального компьютера. Внутри модуля в виде строковых констант жёстко заданы приветственное сообщение с перечнем доступных клавиатурных команд и форматированная текстовая шапка для вывода таблицы результатов измерений.

Логика работы подсистемы построена на конечном автомате, последовательно перебирающем символы заданных константных строк. При поступлении внешнего сигнала сброса автомат переходит в активное состояние вывода первого символа строки меню. В этом состоянии алгоритм использует встроенную функцию `character'pos` для получения числового кода ASCII текущего символа, преобразует его в восьмибитный вектор и ожидает снятия флага занятости от передатчика UART. Как только линия освобождается, модуль выставляет сформированный байт на шину данных, формирует сигнал отправки `ot_tx_start`, инкрементирует внутренний указатель адреса символа и переходит во вспомогательное состояние ожидания. В этом состоянии автомат задерживается до тех пор, пока передатчик UART не подтвердит начало отправки выставлением флага занятости, после чего цикл повторяется для следующего символа. После полной посимвольной передачи меню автомат аналогичным образом выводит шапку таблицы, после чего возвращается в режим простоя до получения следующего сигнала сброса. Фрагмент кода, реализующий логику посимвольного вывода текстовой константы с проверкой занятости передатчика, представлен ниже.

```

when PRINT_MENU =>
    if ot_tx_busy = '0' then
        if text_index <= MENU_TEXT'length then
            ot_tx_data <=
std_logic_vector(to_unsigned(character'pos(MENU_TEXT(text_index)),
8));

            ot_tx_start <= '1';
            text_index <= text_index + 1;

```

```

        next_state_after_wait <= PRINT_MENU;
        state <= WAIT_BUSY;
    else
        state <= PRINT_HEADER;
        text_index <= 1;
    end if;
end if;

```

Третьей и наиболее масштабной подсистемой центрального модуля управления выступает контроллер измерительной периферии. Данный блок выполняет функции управления датчиками и вычислительного сопроцессора, отвечающего за форматирование итогового потока данных для передачи на ПК.

Архитектура контроллера включает в себя структурную инициализацию драйвера микросхемы часов реального времени DS1302, модулей взаимодействия с цифровыми датчиками АНТ21 и BMP180, а также специализированного арбитра шины I2C. Наличие аппаратного арбитра обусловлено тем, что оба измерительных датчика физически располагаются на единой двунаправленной шине данных, требующей строгого распределения прав доступа для исключения коллизий.

Поведенческая логика самого контроллера датчиков базируется на объёмном конечном автомате, насчитывающем несколько десятков состояний. При появлении разрешающего сигнала от модуля обработки кнопок автомат декодирует внутреннюю маску запроса и инициирует последовательный опрос необходимой периферии. Независимо от выбранного пользователем физического параметра, измерительный цикл всегда начинается с обращения к драйверу часов для фиксации текущих значений времени и даты. Затем, в соответствии с активной маской, подаются управляющие импульсы на запуск аппаратных преобразований внутри самих датчиков. В случае с BMP180 алгоритм дополнительно обеспечивает регламентированную временную задержку между командами измерения температуры и давления. После успешного считывания и сохранения ответов от датчиков во внутренних регистрах-защёлках, автомат переходит к вычислительной фазе.

В фазе вычислений и форматирования «сырые» данные преобразуются в читаемый вид для корректного вывода в консоль. Для вычисления физических

величин применяются встроенные алгоритмы аппаратной арифметики. Например, для получения реального значения температуры с датчика АНТ21 применяется формула

$$T = \left(\frac{S_T}{2^{20}} \right) \times 200 - 50,$$

где T – температура, °С;

S_T – выходной температурный сигнал [1].

В рамках ПЛИС данное вычисление эффективно реализовано через умножение вектора в тридцатидвухбитном аккумуляторе с последующим извлечением старших бит, что аппаратно эквивалентно делению на два в двадцатой степени. В то же время показания датчика ВМР180, математический аппарат которого требует выполнения сложных вычислений с калибровочными коэффициентами, передаются на ПК в виде исходных шестнадцатеричных значений для последующей программной постобработки. Фрагмент кода, реализующий аппаратную реализацию расчёта температуры и определения математического знака полученной физической величины, представлен ниже.

```
when S_TEMP_CALC =>
    math_32 <= resize(unsigned(latched_aht_temp) * 200, 32);
    state <= S_TEMP_SIGN;

when S_TEMP_SIGN =>
    v_temp_int <= to_integer(math_32(27 downto 20));
    if v_temp_int < 50 then
        calc_int <= 50 - v_temp_int;
    else
        calc_int <= v_temp_int - 50;
    end if;
```

После завершения математических операций конечный автомат побайтово генерирует строковые последовательности ASCII-символов. Алгоритм последовательно формирует порядковый номер измерения, текущую дату и время, а затем добавляет к вычисленным значениям датчиков соответствующие единицы измерения, разделители столбцов и пробелы для сохранения строгого визуального выравнивания таблицы. Сформированные байты с учётом контроля флага занятости отправляются в передатчик UART. Завершается измерительный цикл

отрисовкой замыкающей линии таблицы и генерацией символов перевода каретки, после чего автомат возвращается в исходное состояние простоя.

Важным структурным элементом контроллера датчиков выступает аппаратный арбитр шины I2C. Датчики АНТ21 и ВМР180 подключены параллельно к единым двунаправленным линиям тактирования и данных. Поскольку каждый датчик управляется независимым программным драйвером, одновременное обращение к шине приведёт к искажению данных и аппаратным коллизиям. Модуль `i2c_bus_arbiter` решает эту проблему, обеспечивая право доступа к физическому интерфейсу только для одного из драйверов в конкретный момент времени.

Архитектура арбитра включает модуль контроллера шины `i2c_master`, который выполняет всю низкоуровневую логику. Сам арбитр выступает в роли интеллектуального коммутатора, маршрутизирующего управляющие сигналы, адреса и данные между драйверами датчиков и мастером шины.

Поведенческая логика коммутации построена на базе конечного автомата, состоящего из состояния простоя, активности первого устройства и активности второго устройства. В режиме простоя арбитр непрерывно опрашивает флаги запроса доступа `dev1_ena` и `dev2_ena`. При инициации транзакции одним из драйверов, автомат переходит в соответствующее активное состояние, перенаправляя все сигналы активного драйвера на входы мастера шины. Одновременно для драйвера, находящегося в режиме ожидания, аппаратно выставляется флаг занятости, блокирующий его работу до полного завершения текущей транзакции активным устройством. Возврат в состояние простоя происходит только после синхронного снятия флага запроса драйвером и освобождения шины мастером. Фрагмент кода, реализующий логику переключения состояний автомата арбитража, представлен ниже.

```
process (clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            state <= IDLE;
```

```

else
    case state is
        when IDLE =>
            if dev1_ena = '1' then
                state <= DEV1_ACTIVE;
            elsif dev2_ena = '1' then
                state <= DEV2_ACTIVE;
            end if;

            when DEV1_ACTIVE =>
                if (dev1_ena = '0' and master_busy = '0') then
                    state <= IDLE;
                end if;

            when DEV2_ACTIVE =>
                if (dev2_ena = '0' and master_busy = '0') then
                    state <= IDLE;
                end if;

            when others =>
                state <= IDLE;
            end case;
        end if;
    end if;
end process;

```

Элементом управления физическим уровнем шины выступает контроллер `i2c_master`. Данный модуль инкапсулируется внутри аппаратного арбитра и абстрагирует высокоуровневые драйверы датчиков от жёстких временных требований протокола I2C. Модуль принимает стандартизированные параллельные запросы и преобразует их в последовательные битовые послышки, аппаратно генерируя тактовый сигнал SCL заданной частоты и управляя двунаправленной линией данных SDA.

В основе архитектуры контроллера лежит реализация двух синхронных процессов. Первый процесс выступает в роли таймера-генератора. Опираясь на системную тактовую частоту и заданную константу скорости I2C, он формирует внутренние сигналы `scl_clk` и `data_clk`, смещённые друг относительно друга по фазе на 90 градусов. Такая реализация позволяет надёжно фиксировать уровни на линии SDA ровно в середине тактового импульса SCL, что является ключевым требованием спецификации шины. Кроме того, таймер-генератор поддерживает функцию удлинения такта. Если ведомое устройство удерживает линию SCL в

низком уровне, то генератор приостанавливает работу до её освобождения.

Второй синхронный процесс представляет собой сложный конечный автомат, управляющий логикой транзакции. Автомат состоит из девяти состояний, охватывающих полный спектр операций протокола, начиная от генерации стартового условия и побитовой передачи байта адреса с битом чтения или записи, до приёма или передачи байта данных с последующей генерацией подтверждений и стопового условия. Фрагмент кода, реализующий логику перехода автомата при передаче или приёме информационного байта после успешного подтверждения адреса, представлен ниже.

```
when SLV_ACK1 =>
    if addr_rw(0) = '0' then
        sda_int <= data_tx(bit_cnt);
        state <= WR;
    else
        sda_int <= '1';
        state <= RD;
    end if;

when WR =>
    busy <= '1';
    if bit_cnt = 0 then
        sda_int <= '1';
        bit_cnt <= 7;
        state <= SLV_ACK2;
    else
        bit_cnt <= bit_cnt - 1;
        sda_int <= data_tx(bit_cnt - 1);
        state <= WR;
    end if;

when RD =>
    busy <= '1';
    if bit_cnt = 0 then
        if (ena = '1' and addr_rw = addr & rw) then
            sda_int <= '0';
        else
            sda_int <= '1';
        end if;
        bit_cnt <= 7;
        data_rd <= data_rx;
        state <= MSTR_ACK;
    else
        bit_cnt <= bit_cnt - 1;
        state <= RD;
```

```
end if;
```

Для получения точного времени и даты в состав измерительного комплекса интегрирован драйвер датчика часов реального времени DS1302. В отличие от модулей АНТ21 и ВМР180, он использует специализированный трёхпроводной синхронный интерфейс, включающий отдельные линии тактирования, выбора устройства и двунаправленную линию ввода-вывода данных. Поскольку этот интерфейс независим от основной шины I2C, драйвер функционирует как самостоятельный аппаратный узел, не требующий арбитража.

Логика работы драйвера базируется на конечном автомате, реализующем режим пакетного чтения. Данный режим позволяет за одну транзакцию извлечь все основные хронометрические регистры микросхемы, минимизируя накладные расходы на адресацию. Транзакция инициируется внешним запросом `read_req` от контроллера датчиков. Автомат переводит линию выбора `rtc_se` в высокий уровень и в состоянии `CMD_BITS` побитно выдаёт на линию данных команду пакетного чтения `0xBF`, синхронизируя передачу спадами аппаратно генерируемого тактового сигнала.

После успешной передачи восьми бит команды автомат переключает направление двунаправленной линии `rtc_io` на вход и переходит в состояние `READ_BITS`. В этом режиме алгоритм последовательно считывает 64 бита данных, фиксируя логические уровни по переднему фронту тактового сигнала. Принимаемые биты маршрутизируются во внутренний массив регистров `time_regs` с использованием вложенных счётчиков битов и байтов. По завершении приёма последней посылки модуль переходит в состояние `STOP`, опускает линию разрешения, выставляет параллельные данные на выходные порты и формирует импульс готовности `ready`. Фрагмент кода, реализующий логику переключения направления линии данных и перехода к чтению байтов после отправки команды, представлен ниже [6].

```
when CMD_BITS =>
  if sclk_edge = '1' then
    if sclk_reg = '0' then
      sclk_reg <= '1';
```

```

else
    sclk_reg <= '0';
    if bit_idx = 7 then
        io_dir <= '0';
        state <= READ_BITS;
        bit_idx <= 0;
        byte_idx <= 0;
    else
        bit_idx <= bit_idx + 1;
        io_out <= CMD_BURST_READ(bit_idx + 1);
    end if;
end if;
end if;

```

Для получения достоверных значений температуры и относительной влажности окружающей среды в комплекс интегрирован аппаратный драйвер датчика АНТ21. Данный модуль взаимодействует с датчиком посредством общей шины I2C через специализированный блок аппаратного арбитража, абстрагируясь от низкоуровневого тактирования и формирования стартовых или стоповых условий. Взаимодействие с датчиком осуществляется по фиксированному шестнадцатеричному адресу 0x38.

Логика работы драйвера построена на синхронном конечном автомате, алгоритм которого строго регламентирован технической документацией на АНТ21. Полный цикл измерения разбивается на инициализацию измерения, аппаратное ожидание конвертации и чтение результатов. Цикл начинается при поступлении внешнего сигнала start_meas от центрального контроллера датчиков. В фазе инициализации автомат переходит в режим записи и последовательно отправляет на шину команду триггера измерения 0xAC, за которой следуют калибровочные параметры 0x33 и 0x00. Завершив передачу команды, драйвер снимает флаг разрешения шины и переходит в состояние ожидания WAIT_MEAS.

Фаза ожидания критически важна для корректной работы датчика, так как внутреннему АЦП требуется время для преобразования физических величин. Длительность паузы формируется аппаратным счётчиком и составляет не менее 80 миллисекунд. По завершении паузы автомат переводит арбитр шины в режим чтения и иницирует приём семи байт данных. Алгоритм подсчитывает

количество принятых байтов и осуществляет побитовую распаковку. Байты со второго по первую половину четвёртого формируют двадцатибитное «сырое» значение влажности, а байты со второй половины четвёртого по шестой – двадцатибитное «сырое» значение температуры. Распакованные векторы фиксируются во внутренних регистрах, после чего драйвер выставляет флаг готовности data_ready и возвращается в исходное состояние. Фрагмент кода, реализующий логику распаковки полезной нагрузки из последовательно принимаемых байтов, представлен ниже [1].

```
if (busy_cnt > 0 and i2c_busy = '0') then
  case busy_cnt is
    when 1 =>
      null;

    when 2 =>
      raw_hum_reg(19 downto 12) <= i2c_data_rd;

    when 3 =>
      raw_hum_reg(11 downto 4) <= i2c_data_rd;

    when 4 =>
      raw_hum_reg(3 downto 0) <= i2c_data_rd(7 downto 4);
      raw_temp_reg(19 downto 16) <= i2c_data_rd(3 downto 0);

    when 5 =>
      raw_temp_reg(15 downto 8) <= i2c_data_rd;

    when 6 =>
      raw_temp_reg(7 downto 0) <= i2c_data_rd;

    when 7 =>
      null;
      state <= READ_STOP;

    when others =>
      null;
  end case;
end if;
```

Для измерения атмосферного давления используется драйвер датчика BMP180. Как и в случае с АНТ21, взаимодействие с ним реализовано через аппаратный арбитр шины I2C, что позволяет отделить логику высокоуровневого протокола от управления физическими уровнями сигналов. Связь с BMP180

осуществляется по фиксированному базовому адресу 0x77.

Особенностью датчика BMP180 является то, что он не способен измерять температуру и давление одновременно одной командой. Архитектура драйвера учитывает этот нюанс. Модуль спроектирован как универсальный конечный автомат, способный выполнить единичный цикл измерения на основе внешней команды `meas_cmd`, передаваемой от центрального контроллера.

Процесс начинается при получении сигнала `start_meas`. Автомат переходит в режим записи и обращается к внутреннему регистру управления датчика по адресу 0xF4. После подтверждения адреса регистра драйвер пересылает в него код требуемой операции и формирует стоповое условие. За этой транзакцией следует обязательная фаза ожидания конвертации `WAIT_MEAS`, реализованная на базе аппаратного счётчика.

После истечения времени ожидания автомат инициирует вторую I2C транзакцию для считывания результатов. Драйвер отправляет адрес регистра выходных данных 0xF6, переключает шину в режим чтения и последовательно принимает два байта. Старший байт записывается в старшую половину шестнадцатеричного регистра `result_reg`, а младший – в младшую. После приёма младшего байта драйвер снимает сигнал разрешения, формирует стоповое условие и выставляет флаг `data_ready`, возвращаясь в состояние простоя. Фрагмент кода, реализующий логику побайтового чтения результата с контролем занятости шины, представлен ниже [2].

```
when READ_MSB =>
    i2c_rw <= '1';
    busy_prev <= i2c_busy;
    if (busy_prev = '0' and i2c_busy = '1') then
        busy_cnt <= busy_cnt + 1;
    end if;
    if (busy_cnt = 1 and i2c_busy = '0') then
        result_reg(15 downto 8) <= i2c_data_rd;
        busy_cnt <= 2;
        state <= READ_LSB;
    end if;

when READ_LSB =>
    busy_prev <= i2c_busy;
```

```

if (busy_prev = '0' and i2c_busy = '1') then
    busy_cnt <= busy_cnt + 1;
end if;

if busy_cnt >= 3 then
    i2c_ena <= '0';
end if;

if (busy_cnt >= 3 and i2c_busy = '0') then
    result_reg(7 downto 0) <= i2c_data_rd;
    state <= READ_STOP;
end if;

```

Для обеспечения связи между логической структурой проекта и физическими выводами микросхемы ПЛИС используется файл пользовательских ограничений (UCF). Данный файл необходим для того, чтобы задействовать аппаратные ресурсы кристалла и подключить внутренние сигналы управления к внешним компонентам стенда и интерфейсным разъёмам.

В UCF файле определяются физические адреса контактов для каждой сигнальной линии, а также задаются электрические стандарты уровней напряжения, обеспечивающие корректное взаимодействие с периферийными модулями. Помимо назначения выводов, файл содержит временные спецификации для глобального тактового сигнала, что гарантирует стабильную синхронизацию всех вычислительных процессов внутри ПЛИС на заданной частоте. Содержимое UCF файла представлено ниже.

```

NET "rx_serial" LOC = "C11" | IOSTANDARD = LVCMOS33;
NET "tx_serial" LOC = "D12" | IOSTANDARD = LVCMOS33;

NET "scl" LOC = "M5" | IOSTANDARD = I2C;
NET "sda" LOC = "P6" | IOSTANDARD = I2C;

NET "rtc_sclk" LOC = "E13" | IOSTANDARD = LVCMOS33;
NET "rtc_ce" LOC = "C13" | IOSTANDARD = LVCMOS33;
NET "rtc_io" LOC = "D14" | IOSTANDARD = LVCMOS33;

NET "button_rst" LOC = "C3" | IOSTANDARD = LVCMOS33;

NET "clk" LOC = "T8" | TNM_NET = "clk" | IOSTANDARD = LVCMOS33;
TIMESPEC "TS_clk" = PERIOD "clk" 50000 kHz;

```

3.4 Разработка ПО для ПК

Программное обеспечение ПК разработано на языке C++ в среде Visual Studio 2022 и представляет собой консольное клиентское приложение. Главная задача программы заключается в установке асинхронного соединения с аппаратным модулем ПЛИС, обработке поступающего потока данных и обеспечении их надёжного хранения в локальной БД. Логически алгоритм работы приложения можно разделить на три последовательных этапа.

Первый этап включает инициализацию интерфейса связи. Программа запрашивает у пользователя номер коммуникационного порта и скорость передачи данных. Для конфигурации интерфейса UART применяются функции Windows API. Открытие порта осуществляется функцией CreateFileA, после чего настраиваются структура параметров DCB и таймауты чтения-записи. Фрагмент кода, реализующий конфигурацию последовательного порта, представлен ниже.

```
string portName = "\\.\COM" + portNum;

hSerial = CreateFileA(portName.c_str(), GENERIC_READ |
GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

if (hSerial == INVALID_HANDLE_VALUE) {
    cout << "-> Ошибка: Не удалось подключиться к COM" << portNum <<
". Возможно, он отключен или занят.\n\n";
    continue;
}

SetupComm(hSerial, 16384, 16384);
DCB dcb = { 0 };
dcb.DCBlength = sizeof(dcb);
GetCommState(hSerial, &dcb);

dcb.BaudRate = userBaudRate;
dcb.ByteSize = 8;
dcb.StopBits = ONESTOPBIT;
dcb.Parity = NOPARITY;
```

Вторым этапом выступает развёртывание подсистемы хранения информации. Приложение автоматически генерирует уникальное имя файла БД на основе текущего системного времени. Взаимодействие с локальной СУБД SQLite осуществляется через прямое подключение заголовочных файлов библиотеки.

Алгоритм формирует запрос на создание таблицы SensorLog, определяя структуру столбцов для фиксации номера измерения, временной метки и полученных физических величин. Фрагмент кода, реализующий алгоритм создания БД и настройку параметров журналирования, представлен ниже.

```
string db_filename = generate_db_filename();
sqlite3* db;
if (sqlite3_open(db_filename.c_str(), &db) != SQLITE_OK) {
    cerr << "Ошибка открытия БД SQLite!\n";
    CloseHandle(hSerial);
    return 1;
}

sqlite3_exec(db, "PRAGMA journal_mode = WAL; PRAGMA synchronous =
NORMAL;", nullptr, nullptr, nullptr);
sqlite3_busy_timeout(db, 5000);

const char* sql_create =
    "CREATE TABLE IF NOT EXISTS SensorLog ("
    "RequestNum TEXT, "
    "DateTime TEXT, "
    "\"Temperature(C)\" REAL, "
    "\"Pressure(Pa)\" REAL, "
    "\"Humidity(%)\" REAL);";
sqlite3_exec(db, sql_create, nullptr, nullptr, nullptr);
```

Третий и наиболее объёмный этап представляет собой главный рабочий цикл приложения. Он базируется на неблокирующем чтении буфера последовательного порта и асинхронном перехвате пользовательского ввода с клавиатуры. При нажатии клавиши функция `_kbhit()` регистрирует событие, и соответствующий ASCII-символ отправляется в ПЛИС с помощью функции `WriteFile`. Параллельно функция `ReadFile` принимает входящие текстовые послыки от платы.

Поступающие символы накапливаются в строковом буфере до обнаружения символа перевода строки. Полученная строка разбивается на отдельные части по символу вертикальной черты. Аппаратная часть передаёт необработанные шестнадцатеричные значения давления, которые программно преобразуются и подставляются в математическую функцию `calculate_bmp180_pressure` для вычисления абсолютного давления в Паскалях. В завершении цикла программа

осуществляет форматированный вывод рассчитанных показателей в консоль и формирует SQL-запрос INSERT INTO для записи строки в БД. Фрагмент кода, реализующий чтение порта, отправку команд и запись результатов в БД, представлен ниже [7].

```
while (true) {
    bool activity = false;

    if (_kbhit()) {
        char ch = _getch();
        if (ch == 27) break; // ESC
        WriteFile(hSerial, &ch, 1, &bytesWritten, NULL);
        activity = true;
    }

    if (ReadFile(hSerial, buffer, sizeof(buffer) - 1, &bytesRead,
        NULL) && bytesRead > 0) {
        activity = true;
        buffer[bytesRead] = '\\0';
        line_buffer += buffer;

        size_t pos;
        while ((pos = line_buffer.find('\\n')) != string::npos) {
            //...

            if (parts.size() >= 2 && parts[1].length() > 0 &&
                isdigit((unsigned char)parts[1][0])) {
                //...

                string sql = "INSERT INTO SensorLog (RequestNum,
                    DateTime, \\\"Temperature(C)\\\", \\\"Pressure(Pa)\\\", \\\"Humidity(%)\\\")
                    VALUES ('"
                    + num + "', '" + date + "', "
                    + to_string(clean_temp) + ", "
                    + to_string(calc_P) + ", "
                    + to_string(clean_hum) + ");";

                char* errMsg = nullptr;
                if (sqlite3_exec(db, sql.c_str(), nullptr, nullptr,
                    &errMsg) != SQLITE_OK) {
                    sqlite3_free(errMsg);
                }
                //...
            }
        }
    }
}
```

3.5 Тестирование системы

В начале процесса тестирования необходимо сформировать структурированный план проверок. Это позволит последовательно оценить корректность работы всех узлов системы. В рамках испытаний выделены четыре основных этапа. На первом этапе проверяется надёжность системы валидации пользовательского ввода при настройке параметров соединения. На втором этапе тестируется способность приложения устанавливать связь с аппаратной платформой и создавать локальную БД. Третий этап включает проверку алгоритмов обмена командами и корректность обработки получаемых данных. Четвёртый этап направлен на подтверждение целостности сохранённой информации в энергонезависимой памяти ПК. План тестирования представлен в таблице 1.

Таблица 1 – План тестирования

Номер этапа	Описание проверки	Ожидаемый результат
1	Валидация ввода параметров	Блокировка букв и знаков, вывод ошибок
2	Инициализация связи и создание БД	Открытие порта и создание файла БД
3	Функциональный опрос датчиков	Получение данных и вывод их на экран
4	Проверка хранения данных	Запись всех измерений в БД

На начальной стадии запуска клиентского приложения проверяется устойчивость программы к некорректным действиям пользователя согласно первому пункту плана. Алгоритм ввода параметров соединения оснащён программными фильтрами, блокирующими нечисловые символы и значения, выходящие за допустимые пределы конфигурации порта. Реакция приложения на ввод отрицательных чисел, чрезмерно длинных строк и попытку подключения к несуществующему коммуникационному порту представлена на рисунке 55.

```
C:\Users\Максим\source\repo x + v
Нажмите ESC для завершения работы
Введите номер COM-порта: -5
-> Ошибка: Введены недопустимые символы. Пожалуйста, введите только цифры.

Введите номер COM-порта: 5
Введите скорость UART: -115200
-> Ошибка: Введены недопустимые символы. Пожалуйста, введите только цифры.

Введите номер COM-порта: 5
Введите скорость UART: 9999999999999999999
-> Ошибка: Введено слишком большое значение для скорости.

Введите номер COM-порта: 7
Введите скорость UART: 115200
-> Ошибка: Не удалось подключиться к COM7. Возможно, он отключен или занят.

Введите номер COM-порта: |
```

Рисунок 55 – Проверка валидации ввода и обработки ошибок

Второй этап тестирования посвящён проверке первичной инициализации системы. После ввода верных параметров программа должна захватить порт и подготовить файл для записи результатов измерений. Подтверждение успешного подключения к порту на заданной скорости и сообщение об автоматическом создании файла БД представлены на рисунке 56.

```
Введите номер COM-порта: 5
Введите скорость UART: 115200

Успешное подключение к COM5 (Скорость: 115200)
База данных успешно создана: sensors_data_20260506_175016.db
Нажмите R для начала работы

|
```

Рисунок 56 – Подтверждение установки связи и создания БД

На третьем этапе проверяется функциональность опроса датчиков. После нажатия клавиши сброса в консоль выводится меню управления и шапка таблицы. Результаты последовательного выполнения одиночных запросов температуры, давления, влажности и комплексного опроса всех датчиков представлены на рисунке 57.

```

--- Sensor Dashboard ---
Use buttons: 1 = Temperature, 2 = Pressure, 3 = Humidity, 4 = All, R = Reset

```

No.	Time and Date	Temperature (C)	Pressure (Pa)	Humidity (%)
001	17:52:16 06.05.26	+22C		
002	17:52:34 06.05.26		98737 Па	
003	17:52:42 06.05.26			39%
004	17:52:48 06.05.26	+22C	98726 Па	39%

Рисунок 57 – Выполнение измерительных команд и вывод таблицы данных

Завершающим этапом тестирования является проверка работы подсистемы энергонезависимого хранения. После завершения работы программы в рабочей директории можно наблюдать сгенерированный файл. Наличие созданного файла БД в корневой папке проекта представлено на рисунке 58.

Имя	Дата изменения	Тип	Размер
.vs	02.03.2026 18:31	Папка с файлами	
Diplom	02.03.2026 21:11	Папка с файлами	
x64	02.03.2026 21:11	Папка с файлами	
Diplom.cpp	06.05.2026 17:40	C++ Source	14 КБ
Diplom.sln	02.03.2026 18:31	Visual Studio Solut...	2 КБ
Diplom.vcxproj	02.03.2026 21:11	Файл "VCXPROJ"	7 КБ
Diplom.vcxproj.filters	02.03.2026 21:11	VC++ Project Filte...	2 КБ
Diplom.vcxproj.user	02.03.2026 18:31	Per-User Project O...	1 КБ
sensors_data_20260506_175016.db	06.05.2026 17:53	Data Base File	8 КБ
sqlite3.c	02.03.2026 18:11	C Source	9 169 КБ
sqlite3.h	02.03.2026 18:11	C/C++ Header	657 КБ

Рисунок 58 – Директория проекта с файлом БД

Для проверки целостности сохранённой информации применяется специализированное ПО. Анализ БД подтверждает соответствие типов данных и записанных значений результатам в консоли терминала. Содержимое сформированной БД с результатами измерений представлено на рисунке 59.

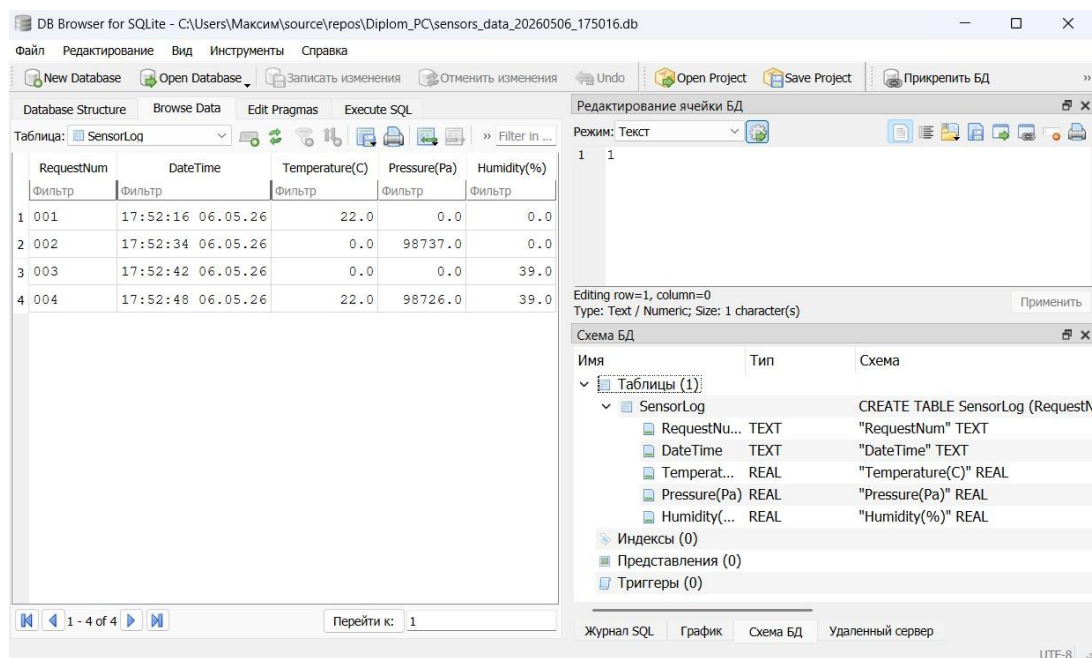


Рисунок 59 – Содержимое БД

Анализ полученных данных подтверждает полную работоспособность цепочки передачи информации и корректность алгоритмов парсинга. Приложение успешно выполняет обработку данных и распределяет измеренные величины по соответствующим столбцам БД. Результаты проведенных испытаний представлены в таблице 2.

Таблица 2 – Результаты тестирования

Номер этапа	Описание проверки	Ожидаемый результат	Фактический результат	Статус
1	Валидация ввода параметров	Блокировка букв и знаков, вывод ошибок	Некорректный ввод успешно заблокирован	Успешно
2	Инициализация связи и создание БД	Открытие порта и создание файла БД	Связь установлена, БД создана	Успешно
3	Функциональный опрос датчиков	Получение данных и вывод их на экран	Данные получены и корректно выведены на экран	Успешно
4	Проверка хранения данных	Запись всех измерений в БД	Информация успешно сохранена в файле	Успешно

4 Безопасность жизнедеятельности

При эксплуатации системы опроса датчиков необходимо соблюдать комплекс мер, направленных на обеспечение безопасности пользователя и сохранение работоспособности оборудования.

4.1 Правила безопасности при использовании ПК

В соответствии с ГОСТ 12.1.019-2017 электробезопасность при работе с вычислительной техникой обеспечивается путём обязательного защитного заземления оборудования. Питание системного блока и периферийных устройств осуществляется от сети переменного тока напряжением 220 В. Подключение всех компонентов системы должно производиться через розетки с заземляющим контактом, соединённым с внешним контуром заземления.

Требования к организации рабочего места с ПК регламентируются единым стандартом ГОСТ Р 50923-96. Согласно данному нормативному документу, рабочее пространство должно быть организовано с учётом минимизации статического напряжения мышц шейно-плечевой области и спины. Экран монитора следует располагать на оптимальном расстоянии от 600 до 700 мм от глаз пользователя. Освещённость рабочей поверхности стола должна быть равномерной и составлять от 300 до 500 лк, при этом оборудование необходимо размещать перпендикулярно источникам естественного света для исключения появления мешающих бликов.

При эксплуатации техники категорически запрещается использовать соединительные кабели с видимыми повреждениями изоляции или неисправные разъёмы. Не допускается ограничение доступа воздуха к вентиляционным отверстиям системного блока и монитора. Соблюдение установленных нормативных правил позволяет обеспечить безопасные условия труда и сохранить работоспособность технических средств в процессе разработки ПО или использования уже готового ПО.

4.2 Правила безопасности при использовании стенда

В соответствии с требованиями ГОСТ 12.1.019-2017 опасными для работающего персонала считаются электрические системы с напряжением более 36 В постоянного тока. Питание макетной платы и всех подключенных модулей датчиков осуществляется через стандартный кабель USB от системного блока ПК с максимальным напряжением 5 В. Данный уровень напряжения классифицируется как абсолютно безопасный и не представляет угрозы поражения электрическим током. Следовательно, эксплуатация стенда не требует организации специализированного заземления самой платы или применения дополнительных средств индивидуальной защиты от поражения током.

Несмотря на электробезопасность устройства для человека, открытые электронные компоненты стенда обладают высокой чувствительностью к электростатическим разрядам. Согласно общим требованиям ГОСТ IEC 61340-5-1-2019 для предотвращения пробоя микроструктур ПЛИС и датчиков необходимо соблюдать правила защиты чувствительных электронных устройств. Работу с отладочной платой следует проводить на ровной диэлектрической поверхности, а перед прямым касанием контактов или микросхем рекомендуется снять накопленный статический заряд.

Перед подачей напряжения на стенд требуется проводить обязательный визуальный осмотр устройства на предмет отсутствия механических повреждений, дефектов изоляции соединительных проводов и наличия посторонних токопроводящих предметов на плате. Любые изменения в аппаратной конфигурации стенда, включая подключение интерфейсов I2C и UART, замену датчиков или переподключение сигнальных линий, должны производиться исключительно при физически отключенном кабеле питания. Строгое соблюдение описанных правил гарантирует безопасность и предотвращает риск короткого замыкания или выхода из строя аппаратной части проекта.

Заключение

В ходе выполнения выпускной работы был успешно спроектирован и реализован программно-аппаратный комплекс для сбора, обработки и долговременного хранения данных с датчиков. Разработанный стенд позволяет обеспечить все основные функции, представленные в назначении.

В рамках проекта была создана отказоустойчивая аппаратная архитектура на базе ПЛИС, обеспечивающая синхронизацию процессов и опрос датчиков по интерфейсу I2C. Написанное для ПК клиентское ПО успешно решает задачи приёма данных по интерфейсу UART, математической постобработки и надёжного сохранения данных в локальной БД SQLite.

Заложенная архитектура программного и аппаратного обеспечения является гибко настраиваемой и масштабируемой. Использование модульного подхода при написании VHDL-кода и логическое разделение процессов в клиентском приложении позволяют оперативно изменять параметры системы или интегрировать новые датчики при изменении поставленной задачи.

Проведённое комплексное тестирование подтвердило высокую надёжность алгоритма валидации и обмена информацией. Разработанный стенд является удобным, практичным и безопасным решением и может применяться в широком перечне систем в зависимости от целей.

					ПГУ 1.090301.35.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. АНТ21_v1.0_datasheet [электронный ресурс]. – URL: https://iot-kmutnb.github.io/blogs/sensors/aht2x/АНТ21_v1.0_datasheet.pdf (дата обращения 11.04.2026).

2. BMP180 [электронный ресурс]. – URL: https://seeeddoc.github.io/Grove-IMU_10DOF/res/BMP180.pdf (дата обращения 11.04.2026).

3. AX309_English_UG.1.0 [электронный ресурс]. – URL: https://github.com/alinxalinx/AX309/blob/master/DOC/AX309_English_UG.1.0.pdf (дата обращения 19.04.2026).

4. AX309 [электронный ресурс]. – URL: <https://github.com/alinxalinx/AX309/blob/master/SCH/AX309.pdf> (дата обращения 02.05.2026).

5. Сара Л. Харрис, Дэвид Харрис Цифровая схемотехника и архитектура компьютера: RISC-V / пер. с англ. В. С. Яценкова, А. Ю. Романова; под ред. А. Ю. Романова. – М.: ДМК Пресс, 2021. – 810 с.: ил. – ISBN 978-5-97060-961-3.

6. DS1302 [электронный ресурс]. – URL: <https://amperkot.ru/static/3236/uploads/datasheets/DS1302.pdf?ysclid=mp1ez43w81384481321> (дата обращения 13.05.2026).

7. Седжвик Роберт Фундаментальные алгоритмы на C++. Анализ/Структура данных/Сортировка/Поиск: Пер. с англ./Роберт Седжвик. – К.: Издательство «ДиаСофт», 2001. – 688 с. – ISBN 966-7393-89-5.

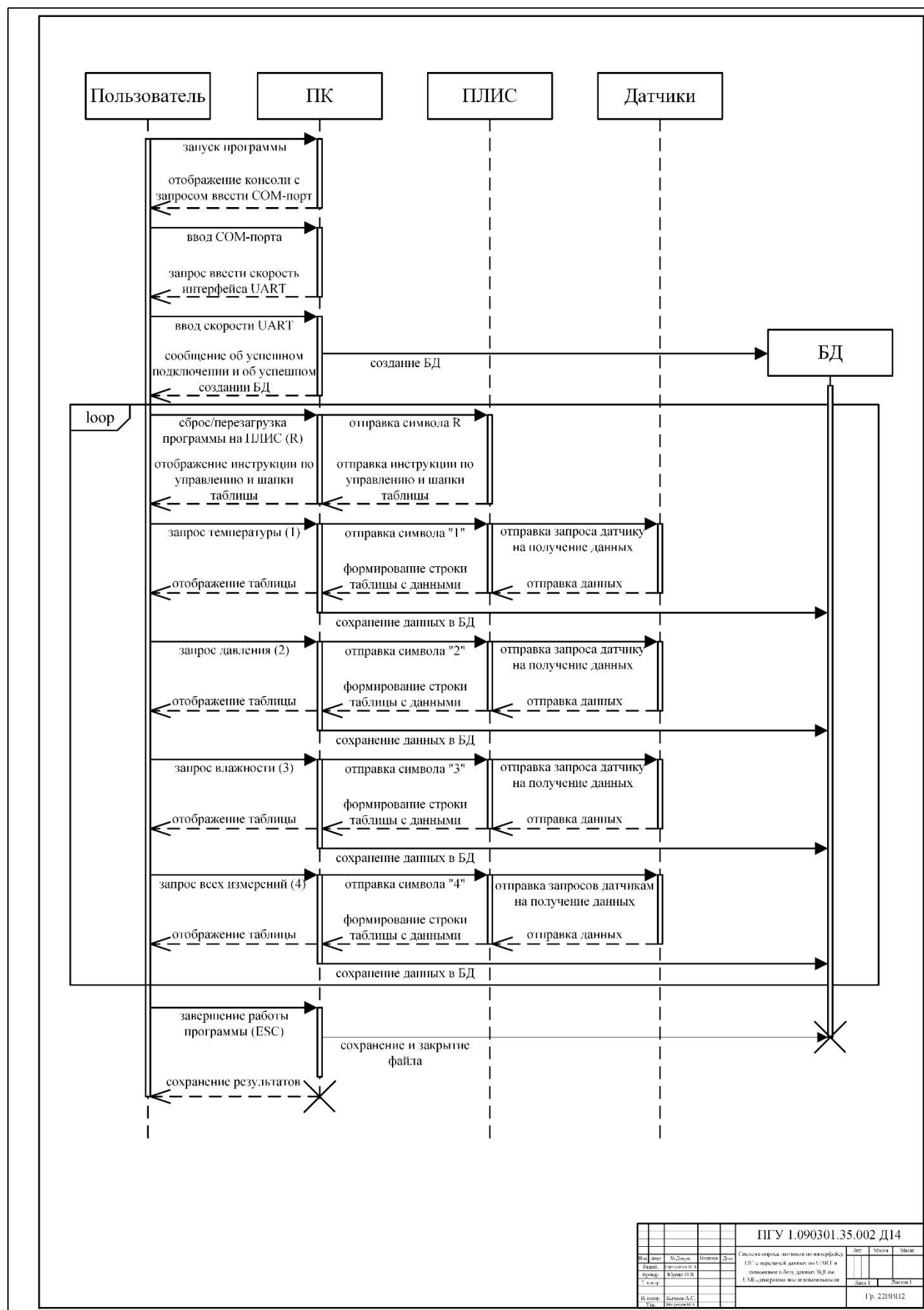
Приложение А
(обязательное)

Система опроса датчиков по интерфейсу I2C с передачей данных по UART и
занесением в базу данных SQLite

UML-диаграммы

					ПГУ 1.090301.35.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

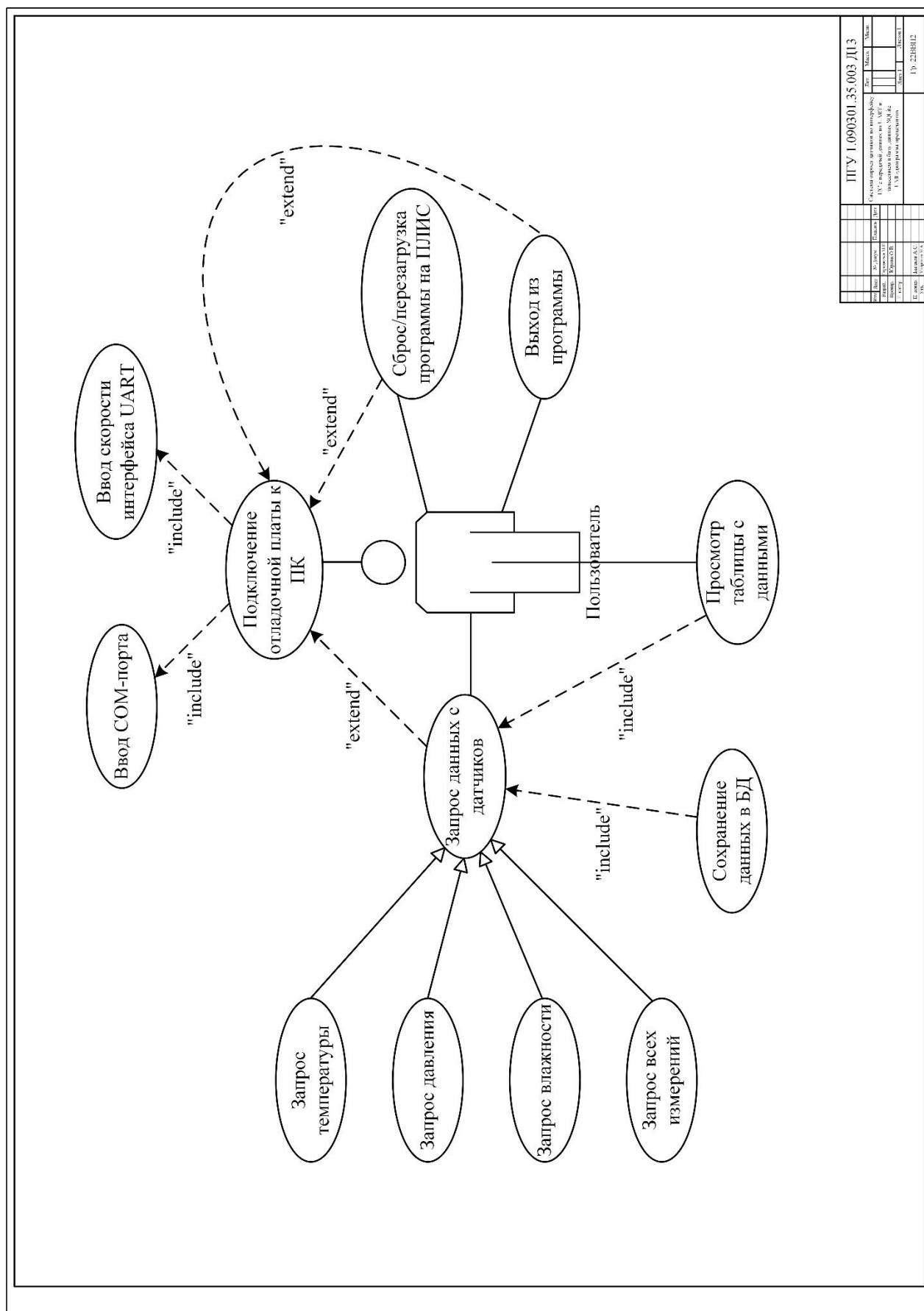
Приложение А.1 – UML-диаграмма последовательностей



					ПГУ 1.090301.35.002 Д14		

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.2 – UML-диаграмма прецедентов



ПГУ 1.090301.35.003 ДПЗ									
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.
И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.	З.о. зам.	С.о. зам.	И.о. зам.

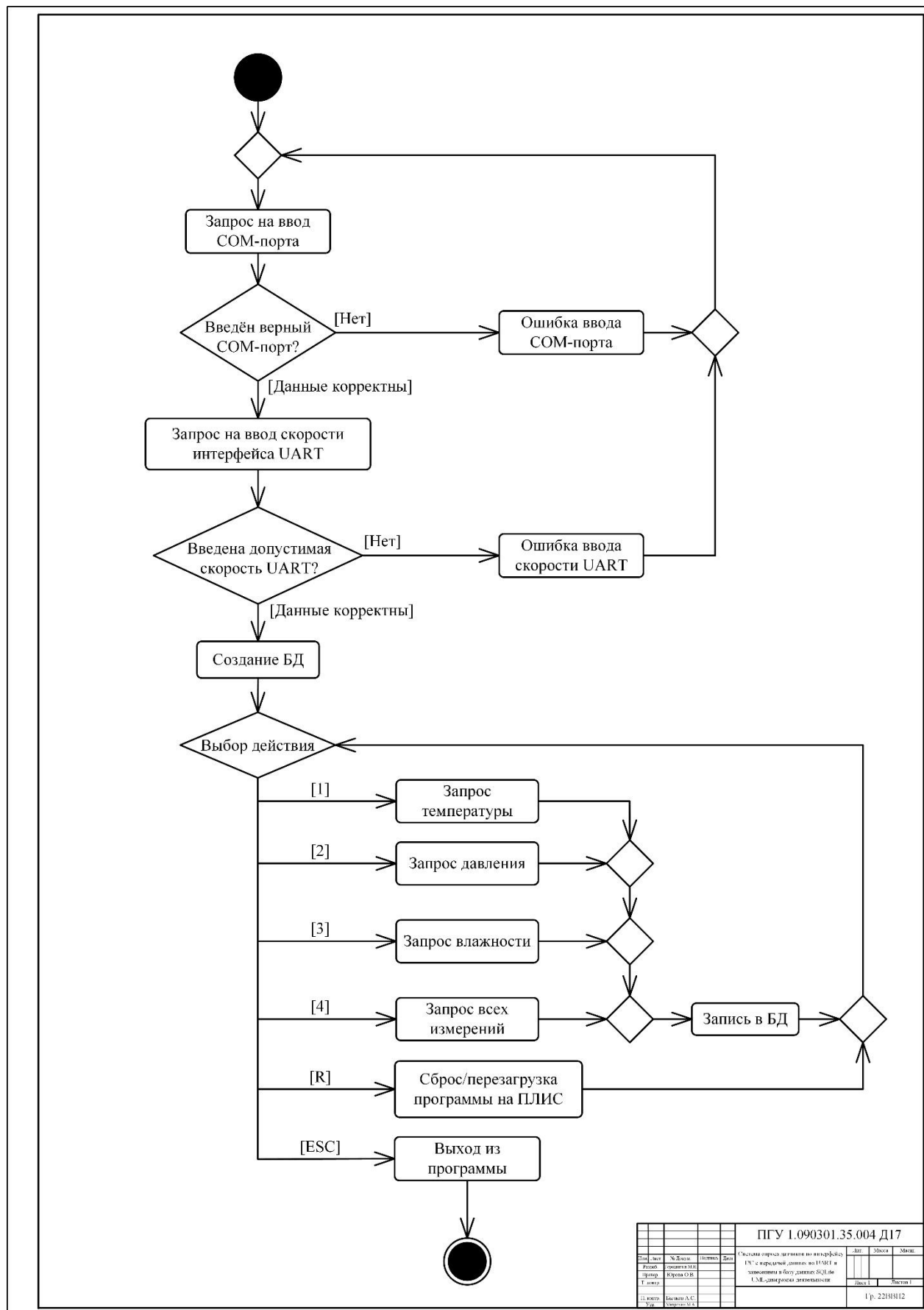
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.35.001 ПЗ

Лист

77

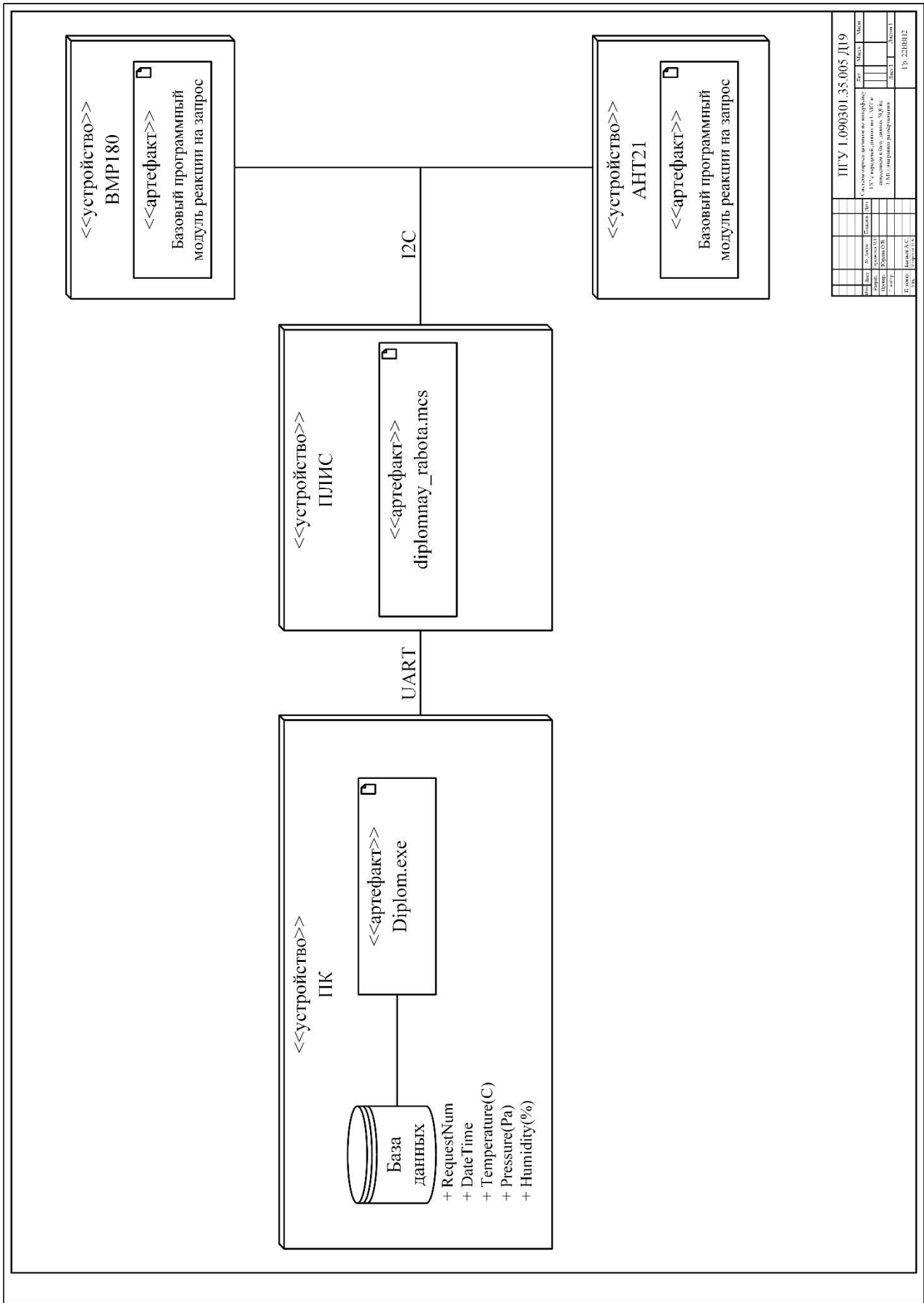
Приложение А.3 – UML-диаграмма деятельности



ПГУ 1.090301.35.004 Д17									
Изм.	Лист	№ докум.	Подпись	Дата	Система опроса датчиков по интерфейсу RS-485 с передачей данных по UART в архивному в базу данных SQL-де. UML-диаграмма деятельности				
Изм.	Лист	№ докум.	Подпись	Дата	Лист 1 из 1				
Изм.	Лист	№ докум.	Подпись	Дата	Ир. 2208/112				

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.4 – UML-диаграмма развёртывания



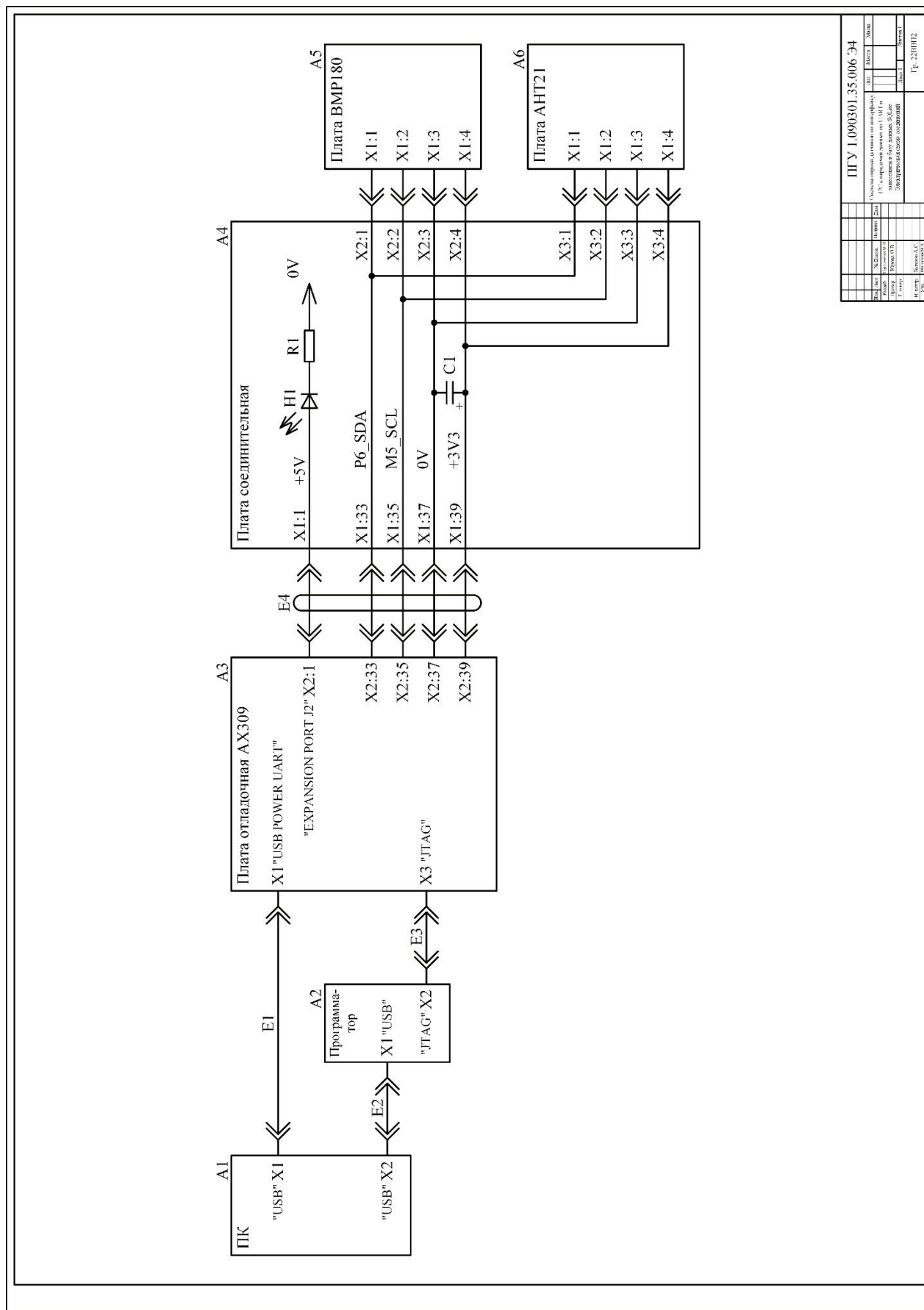
Приложение Б
(обязательное)

Система опроса датчиков по интерфейсу I2C с передачей данных по UART и
занесением в базу данных SQLite

Схема электрическая соединений

					ПГУ 1.090301.35.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 – Электрическая схема соединений



Приложение В

(обязательное)

Система опроса датчиков по интерфейсу I2C с передачей данных по UART и
занесением в базу данных SQLite

Листинги программ

					ПГУ 1.090301.35.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В.1 – Листинг ПО для ПЛИС

Файл top.vhd

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

entity top is
  generic (
    CLK_FREQ      : integer := 50_000_000;
    BAUD_RATE     : integer := 115_200;
    I2C_RATE      : integer := 400_000;
    USE_PARITY     : boolean := false;
    EVEN_PARITY   : boolean := false
  );
  port (
    clk           : in std_logic;
    button_rst    : in std_logic;

    rx_serial     : in  std_logic;
    tx_serial     : out std_logic;

    scl           : inout std_logic;
    sda           : inout std_logic;

    rtc_sclk      : out  std_logic;
    rtc_ce        : out  std_logic;
    rtc_io        : inout std_logic
  );
end top;

architecture Behavioral of top is

  signal rst           : std_logic;
  signal button_rst1   : std_logic;

  signal t_rx_data     : std_logic_vector(7 downto 0);
  signal t_tx_data     : std_logic_vector(7 downto 0);
  signal t_rx_done     : std_logic;
  signal t_tx_start    : std_logic;
  signal t_tx_busy     : std_logic;

begin

  button_reset_proc : process (clk)
  begin
    if rising_edge(clk) then
      button_rst1 <= button_rst;
      rst <= not button_rst1;
    end if;
  end process button_reset_proc;
```

					ПГУ 1.090301.35.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

uart_inst : entity work.uart
  generic map (
    CLK_FREQ    => CLK_FREQ,
    BAUD_RATE   => BAUD_RATE,
    USE_PARITY  => USE_PARITY,
    EVEN_PARITY => EVEN_PARITY
  )
  port map (
    clk      => clk,
    rst      => rst,

    rx_serial => rx_serial,
    tx_serial => tx_serial,

    rx_data   => t_rx_data,
    rx_done   => t_rx_done,

    tx_start  => t_tx_start,
    tx_data   => t_tx_data,
    tx_busy   => t_tx_busy
  );

control_inst : entity work.control
  generic map (
    CLK_FREQ => CLK_FREQ,
    I2C_RATE => I2C_RATE
  )
  port map (
    clk      => clk,
    rst      => rst,

    con_rx_data  => t_rx_data,
    con_rx_done  => t_rx_done,

    con_tx_data  => t_tx_data,
    con_tx_start => t_tx_start,
    con_tx_busy  => t_tx_busy,

    scl      => scl,
    sda      => sda,

    rtc_sclk  => rtc_sclk,
    rtc_ce    => rtc_ce,
    rtc_io    => rtc_io
  );

end Behavioral;

Файл uart.vhd

library ieee;

```

```

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uart is
    generic (
        CLK_FREQ      : integer;
        BAUD_RATE      : integer;
        USE_PARITY     : boolean;
        EVEN_PARITY    : boolean
    );
    port (
        clk            : in  std_logic;
        rst            : 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_start       : in  std_logic;
        tx_data        : in  std_logic_vector(7 downto 0);
        tx_busy        : out std_logic
    );
end uart;

```

architecture Behavioral of uart is

begin

```

    uart_rx_inst : entity work.uart_rx
        generic map (
            CLK_FREQ      => CLK_FREQ,
            BAUD_RATE     => BAUD_RATE,
            USE_PARITY    => USE_PARITY,
            EVEN_PARITY   => EVEN_PARITY
        )
        port map (
            clk            => clk,
            rst            => rst,

            rx_serial      => rx_serial,
            rx_data        => rx_data,
            rx_done        => rx_done
        );

```

```

    uart_tx_inst : entity work.uart_tx
        generic map (
            CLK_FREQ      => CLK_FREQ,
            BAUD_RATE     => BAUD_RATE,
            USE_PARITY    => USE_PARITY,

```

```

        EVEN_PARITY => EVEN_PARITY
    )
    port map (
        clk          => clk,
        rst          => rst,

        tx_serial    => tx_serial,
        tx_start     => tx_start,
        tx_data      => tx_data,
        tx_busy      => tx_busy
    );

end Behavioral;

```

Файл uart_rx.vhd

```

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

entity uart_rx is
    generic (
        CLK_FREQ      : integer;
        BAUD_RATE     : integer;
        USE_PARITY     : boolean;
        EVEN_PARITY    : boolean
    );
    port (
        clk            : in  std_logic;
        rst            : in  std_logic;

        rx_serial      : in  std_logic;
        rx_data         : out std_logic_vector(7 downto 0);
        rx_done         : out std_logic
    );
end uart_rx;

architecture Behavioral of uart_rx is

    constant BAUD_DIV      : integer := CLK_FREQ / BAUD_RATE;
    constant SAMPLE_TIME   : integer := BAUD_DIV / 2;
    type rx_state_t is (IDLE, START, DATA, PARITY, STOP);
    signal state            : rx_state_t := IDLE;
    signal baud_cnt         : integer range 0 to BAUD_DIV := 0;
    signal bit_idx          : integer range 0 to 7 := 0;
    signal shift_reg        : std_logic_vector(7 downto 0) := (others
=> '0');
    signal parity_bit       : std_logic;

begin

```

```

process (clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            state <= IDLE;
            baud_cnt <= 0;
            bit_idx <= 0;
            shift_reg <= (others => '0');
            rx_done <= '0';
            rx_data <= (others => '0');
            parity_bit <= '0';
        else
            rx_done <= '0';
            parity_bit <= '0';
            case state is
                when IDLE =>
                    if rx_serial = '0' then
                        baud_cnt <= 0;
                        state <= START;
                    end if;

                when START =>
                    if baud_cnt = SAMPLE_TIME - 1 then
                        baud_cnt <= 0;
                        if rx_serial = '0' then
                            bit_idx <= 0;
                            state <= DATA;
                            shift_reg <= (others => '0');
                        else
                            state <= IDLE;
                        end if;
                    else
                        baud_cnt <= baud_cnt + 1;
                    end if;

                when DATA =>
                    if baud_cnt = BAUD_DIV - 1 then
                        baud_cnt <= 0;
                        shift_reg(bit_idx) <= rx_serial;
                        if bit_idx = 7 then
                            if USE_PARITY then
                                state <= PARITY;
                            else
                                state <= STOP;
                            end if;
                        else
                            bit_idx <= bit_idx + 1;
                        end if;
                    else
                        baud_cnt <= baud_cnt + 1;
                    end if;
            end case;
        end if;
    end if;
end process;

```

```

        when PARITY =>
            if baud_cnt = BAUD_DIV - 1 then
                baud_cnt <= 0;
                parity_bit <= rx_serial;
                if EVEN_PARITY then
                    if parity_bit /= not
xor_reduce(shift_reg) then
                        state <= IDLE;
                    end if;
                else
                    if parity_bit /=
xor_reduce(shift_reg) then
                        state <= IDLE;
                    end if;
                end if;
                state <= STOP;
            else
                baud_cnt <= baud_cnt + 1;
            end if;

        when STOP =>
            if baud_cnt = BAUD_DIV - 1 then
                baud_cnt <= 0;
                if rx_serial = '1' then
                    rx_data <= shift_reg;
                    rx_done <= '1';
                    state <= IDLE;
                else
                    state <= IDLE;
                end if;
            else
                baud_cnt <= baud_cnt + 1;
            end if;

        when others =>
            state <= IDLE;
        end case;

    end if;
end if;
end process;

```

end Behavioral;

Файл uart_tx.vhd

```

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

```

```

entity uart_tx is
    generic (
        CLK_FREQ      : integer;
        BAUD_RATE      : integer;
        USE_PARITY      : boolean;
        EVEN_PARITY     : boolean
    );
    port (
        clk            : in  std_logic;
        rst            : in  std_logic;

        tx_serial      : out std_logic;
        tx_start        : in  std_logic;
        tx_data         : in  std_logic_vector(7 downto 0);
        tx_busy         : out std_logic
    );
end uart_tx;

architecture Behavioral of uart_tx is

    constant BAUD_DIV : integer := CLK_FREQ / BAUD_RATE;
    type tx_state_t is (IDLE, START, DATA, PARITY, STOP);
    signal state       : tx_state_t := IDLE;
    signal baud_cnt    : integer range 0 to BAUD_DIV := 0;
    signal bit_idx     : integer range 0 to 7 := 0;
    signal parity_bit  : std_logic;

begin

    process (clk)
        variable p : std_logic;
    begin
        if rising_edge(clk) then
            if rst = '1' then
                state <= IDLE;
                baud_cnt <= 0;
                bit_idx <= 0;
                tx_serial <= '1';
                tx_busy <= '0';
                parity_bit <= '0';
            else
                case state is
                    when IDLE =>
                        tx_serial <= '1';
                        tx_busy <= '0';
                        parity_bit <= '0';
                        if tx_start = '1' then
                            p := xor_reduce(tx_data);
                            if EVEN_PARITY then
                                parity_bit <= p;
                            else
                                parity_bit <= not p;
                            end if;
                        end if;
                    end case;
                end if;
            end if;
        end process;
    end architecture;

```

```

        parity_bit <= not p;
    end if;
    baud_cnt <= 0;
    bit_idx <= 0;
    tx_busy <= '1';
    state <= START;
end if;

when START =>
    tx_serial <= '0';
    if baud_cnt = BAUD_DIV - 1 then
        baud_cnt <= 0;
        tx_serial <= tx_data(0);
        state <= DATA;
    else
        baud_cnt <= baud_cnt + 1;
    end if;

when DATA =>
    if baud_cnt = BAUD_DIV - 1 then
        baud_cnt <= 0;
        if bit_idx = 7 then
            if USE_PARITY then
                tx_serial <= parity_bit;
                state <= PARITY;
            else
                tx_serial <= '1';
                state <= STOP;
            end if;
        else
            bit_idx <= bit_idx + 1;
            tx_serial <= tx_data(bit_idx + 1);
        end if;
    else
        baud_cnt <= baud_cnt + 1;
    end if;

when PARITY =>
    if baud_cnt = BAUD_DIV - 1 then
        baud_cnt <= 0;
        tx_serial <= '1';
        state <= STOP;
    else
        baud_cnt <= baud_cnt + 1;
    end if;

when STOP =>
    if baud_cnt = BAUD_DIV - 1 then
        baud_cnt <= 0;
        tx_busy <= '0';
        state <= IDLE;
    end if;

```

```

        else
            baud_cnt <= baud_cnt + 1;
        end if;

        when others =>
            state <= IDLE;
        end case;

    end if;
end if;
end process;

end Behavioral;

```

Файл control.vhd

```

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

entity control is
    generic (
        CLK_FREQ : integer;
        I2C_RATE : integer
    );
    port (
        clk          : in    std_logic;
        rst          : in    std_logic;

        con_rx_data  : in    std_logic_vector(7 downto 0);
        con_rx_done  : in    std_logic;

        con_tx_busy  : in    std_logic;
        con_tx_data  : out   std_logic_vector(7 downto 0);
        con_tx_start : out   std_logic;

        scl          : inout std_logic;
        sda          : inout std_logic;

        rtc_sclk     : out   std_logic;
        rtc_ce       : out   std_logic;
        rtc_io       : inout std_logic
    );
end control;

```

architecture Behavioral of control is

```

    signal con_start_send : std_logic;

    signal con_temp_req   : std_logic;
    signal con_pres_req   : std_logic;
    signal con_hum_req    : std_logic;

```

```

signal con_all_req      : std_logic;

signal output_tx_data   : std_logic_vector(7 downto 0);
signal output_tx_start  : std_logic;

signal sensor_tx_data   : std_logic_vector(7 downto 0);
signal sensor_tx_start  : std_logic;

signal rst_combined     : std_logic;

begin

button_handler_inst : entity work.button_handler
    port map (
        clk          => clk,
        rst          => rst,

        bh_rx_data    => con_rx_data,
        bh_rx_done     => con_rx_done,

        bh_start_send => con_start_send,
        bh_temp_req    => con_temp_req,
        bh_pres_req    => con_pres_req,
        bh_hum_req     => con_hum_req,
        bh_all_req     => con_all_req
    );

output_text_inst : entity work.output_text
    port map (
        clk          => clk,
        rst          => rst,

        ot_start_send => con_start_send,

        ot_tx_busy    => con_tx_busy,
        ot_tx_data     => output_tx_data,
        ot_tx_start    => output_tx_start
    );

rst_combined <= rst or con_start_send;

sensor_controller_inst : entity work.sensor_controller
    generic map (
        CLK_FREQ => CLK_FREQ,
        I2C_RATE => I2C_RATE
    )
    port map (
        clk          => clk,
        rst          => rst_combined,

        temp_req     => con_temp_req,

```

```

        pres_req    => con_pres_req,
        hum_req     => con_hum_req,
        all_req     => con_all_req,

        sc_tx_busy  => con_tx_busy,
        sc_tx_data  => sensor_tx_data,
        sc_tx_start => sensor_tx_start,

        scl         => scl,
        sda         => sda,

        rtc_sclk    => rtc_sclk,
        rtc_ce      => rtc_ce,
        rtc_io      => rtc_io
    );

process (clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            con_tx_data <= (others => '0');
            con_tx_start <= '0';
        else
            if sensor_tx_start = '1' then
                con_tx_data <= sensor_tx_data;
                con_tx_start <= '1';
            elsif output_tx_start = '1' then
                con_tx_data <= output_tx_data;
                con_tx_start <= '1';
            else
                con_tx_start <= '0';
            end if;
        end if;
    end if;
end process;

end Behavioral;

```

Файл button_handler.vhd

```

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

entity button_handler is
    port (
        clk          : in  std_logic;
        rst          : in  std_logic;

        bh_rx_data   : in  std_logic_vector(7 downto 0);
        bh_rx_done   : in  std_logic;
    );
end entity button_handler;

```

```

        bh_start_send : out std_logic;
        bh_temp_req    : out std_logic;
        bh_pres_req    : out std_logic;
        bh_hum_req     : out std_logic;
        bh_all_req     : out std_logic
    );
end button_handler;

architecture Behavioral of button_handler is

begin

    process (clk)
        variable cmd : character;
    begin
        if rising_edge(clk) then
            if rst = '1' then
                bh_start_send <= '0';
                bh_temp_req <= '0';
                bh_pres_req <= '0';
                bh_hum_req <= '0';
                bh_all_req <= '0';
            else
                bh_start_send <= '0';
                bh_temp_req <= '0';
                bh_pres_req <= '0';
                bh_hum_req <= '0';
                bh_all_req <= '0';

                if bh_rx_done = '1' then
                    cmd :=
character'val(to_integer(unsigned(bh_rx_data)));

                    case cmd is
                        when '1' =>
                            bh_temp_req <= '1';

                        when '2' =>
                            bh_pres_req <= '1';

                        when '3' =>
                            bh_hum_req <= '1';

                        when '4' =>
                            bh_all_req <= '1';

                        when 'r' | 'R' =>
                            bh_start_send <= '1';
                            bh_temp_req <= '0';
                            bh_pres_req <= '0';
                            bh_hum_req <= '0';

```

```

        bh_all_req <= '0';

        when others =>
            NULL;
        end case;

    end if;
end if;
end if;
end process;

end Behavioral;

```

Файл output_text.vhd

```

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

entity output_text is
    port (
        clk          : in  std_logic;
        rst          : in  std_logic;

        ot_start_send : in  std_logic;

        ot_tx_busy    : in  std_logic;
        ot_tx_data     : out std_logic_vector(7 downto 0);
        ot_tx_start    : out std_logic
    );
end output_text;

architecture Behavioral of output_text is

    constant MENU_TEXT      : string := "--- Sensor Dashboard ---" &
    CR & LF & "Use buttons: 1 = Temperature, 2 = Pressure, 3 = Humidity,
    4 = All, R = Reset" & CR & LF & CR & LF;
    constant TABLE_HEADER : string := "| No. | Time and Date      |
    Temperature (C) | Pressure (Pa) | Humidity (%) |" & CR & LF & "+----
    -+-----+-----+-----+-----+-----+
    --+" & CR & LF;

    type state_t is (IDLE, PRINT_MENU, PRINT_HEADER, WAIT_BUSY);
    signal state          : state_t := IDLE;
    signal next_state_after_wait : state_t := IDLE;
    signal text_index     : natural range 0 to
    TABLE_HEADER'length;

begin

    process(clk)
    begin

```

```

if rising_edge(clk) then
    if rst = '1' then
        state <= PRINT_MENU;
        text_index <= 1;
        ot_tx_start <= '0';
    else
        ot_tx_start <= '0';

        case state is
            when IDLE =>
                if ot_start_send = '1' then
                    state <= PRINT_MENU;
                    text_index <= 1;
                end if;

            when PRINT_MENU =>
                if ot_tx_busy = '0' then
                    if text_index <= MENU_TEXT'length then
                        ot_tx_data <=
std_logic_vector(to_unsigned(character'pos(MENU_TEXT(text_index)),
8));

                        ot_tx_start <= '1';
                        text_index <= text_index + 1;
                        next_state_after_wait <= PRINT_MENU;
                        state <= WAIT_BUSY;
                    else
                        state <= PRINT_HEADER;
                        text_index <= 1;
                    end if;
                end if;

            when PRINT_HEADER =>
                if ot_tx_busy = '0' then
                    if text_index <= TABLE_HEADER'length
then
                        ot_tx_data <=
std_logic_vector(to_unsigned(character'pos(TABLE_HEADER(text_index))
, 8));

                        ot_tx_start <= '1';
                        text_index <= text_index + 1;
                        next_state_after_wait <=
PRINT_HEADER;

                        state <= WAIT_BUSY;
                    else
                        state <= IDLE;
                    end if;
                end if;

            when WAIT_BUSY =>
                if ot_tx_busy = '1' then
                    state <= next_state_after_wait;

```

```

        end if;
    end case;

    end if;
end if;
end process;

end Behavioral;

```

Файл sensor_controller.vhd

```

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

entity sensor_controller is
    generic (
        CLK_FREQ : integer;
        I2C_RATE : integer
    );
    port (
        clk          : in     std_logic;
        rst          : in     std_logic;

        temp_req     : in     std_logic;
        pres_req     : in     std_logic;
        hum_req      : in     std_logic;
        all_req      : in     std_logic;

        sc_tx_busy   : in     std_logic;
        sc_tx_data   : out    std_logic_vector(7 downto 0);
        sc_tx_start  : out    std_logic;

        scl          : inout  std_logic;
        sda          : inout  std_logic;

        rtc_sclk     : out    std_logic;
        rtc_ce       : out    std_logic;
        rtc_io       : inout  std_logic
    );
end sensor_controller;

architecture Behavioral of sensor_controller is

    signal rtc_req   : std_logic := '0';
    signal rtc_ready : std_logic;
    signal r_sec, r_min, r_hour, r_day, r_mon, r_year :
std_logic_vector(7 downto 0);
    signal p_sec, p_min, p_hour, p_day, p_mon, p_year :
std_logic_vector(7 downto 0);

    signal bmp_start : std_logic := '0';

```

```

signal bmp_ready : std_logic := '0';
signal bmp_cmd   : std_logic_vector (7 downto 0);
signal bmp_res   : std_logic_vector (15 downto 0);

signal aht_start : std_logic := '0';
signal aht_ready : std_logic := '0';
signal aht_hum   : std_logic_vector (19 downto 0);
signal aht_temp  : std_logic_vector (19 downto 0);

signal latched_bmp_pres : std_logic_vector (15 downto 0);
signal latched_bmp_temp : std_logic_vector (15 downto 0);
signal latched_aht_temp : std_logic_vector (19 downto 0);
signal latched_aht_hum  : std_logic_vector (19 downto 0);

signal row_cnt      : integer range 0 to 999 := 1;
signal req_mask     : std_logic_vector(2 downto 0);
signal delay_cnt    : integer range 0 to 100000 := 0;
signal calc_int     : integer range 0 to 999;
signal math_32      : unsigned (31 downto 0);
signal v_temp_int   : integer;
signal space_cnt    : integer range 0 to 20;
signal rst_n        : std_logic;
signal line_idx     : integer range 0 to 80 := 0;

signal sep_idx      : integer range 0 to 6 := 1;
signal col_idx      : integer range 0 to 3 := 1;
signal dot_idx      : integer range 0 to 3 := 1;
signal spc_idx      : integer range 0 to 3 := 1;

signal aht_i2c_ena   : std_logic;
signal aht_i2c_addr  : std_logic_vector(6 downto 0);
signal aht_i2c_rw    : std_logic;
signal aht_i2c_data_wr : std_logic_vector(7 downto 0);
signal aht_i2c_data_rd : std_logic_vector(7 downto 0);
signal aht_i2c_busy  : std_logic;

signal bmp_i2c_ena   : std_logic;
signal bmp_i2c_addr  : std_logic_vector(6 downto 0);
signal bmp_i2c_rw    : std_logic;
signal bmp_i2c_data_wr : std_logic_vector(7 downto 0);
signal bmp_i2c_data_rd : std_logic_vector(7 downto 0);
signal bmp_i2c_busy  : std_logic;

type state_t is (
    S_IDLE,

    S_REQ_RTC, S_WAIT_RTC,

    S_MEAS_AHT, S_WAIT_AHT,

    S_MEAS_BMP_TEMP_CMD, S_MEAS_BMP_TEMP_START, S_WAIT_BMP_TEMP,

```

```

S_BMP_DELAY,
S_MEAS_BMP_PRES_CMD, S_MEAS_BMP_PRES_START, S_WAIT_BMP_PRES,

S_ROW_100, S_ROW_10, S_ROW_1,

S_PR_HOUR_H, S_PR_HOUR_L,
S_PR_MIN_H, S_PR_MIN_L,
S_PR_SEC_H, S_PR_SEC_L,
S_PR_DAY_H, S_PR_DAY_L,
S_PR_MON_H, S_PR_MON_L,
S_PR_YEAR_H, S_PR_YEAR_L,

S_CHECK_TEMP,
S_TEMP_CALC,
S_TEMP_SIGN,
S_TEMP_10, S_TEMP_1,
S_TEMP_C,

S_CHECK_PRES,
S_PRES_H4, S_PRES_H3, S_PRES_H2, S_PRES_H1,
S_TEMP_H4, S_TEMP_H3, S_TEMP_H2, S_TEMP_H1,

S_CHECK_HUM,
S_HUM_CALC,
S_HUM_10, S_HUM_1,
S_HUM_PERC,

S_CR, S_LF,

S_PRINT_LINE, S_PRINT_LINE_WAIT,

S_SEP0, S_SPC0,

S_WAIT_TX_BUSY,

S_SEP1, S_SEP2, S_SEP3,

S_COL,

S_SPC,

S_DOT,

S_ALIGNMENT_EMPTY, S_ALIGNMENT_EMPTY_WAIT
);

signal state                : state_t := S_IDLE;
signal next_state_after_wait : state_t := S_IDLE;

function to_dec_char(val : integer) return std_logic_vector is
begin

```

```

        return std_logic_vector(to_unsigned(48 + val, 8));
    end function;

    function to_hex(val : std_logic_vector(3 downto 0)) return
std_logic_vector is
    begin
        if unsigned(val) < 10 then
            return std_logic_vector(to_unsigned(48 +
to_integer(unsigned(val)), 8));
        else
            return std_logic_vector(to_unsigned(55 +
to_integer(unsigned(val)), 8));
        end if;
    end function;

begin

    rst_n <= not rst;

    i2c_bus_arbiter_inst : entity work.i2c_bus_arbiter
generic map (
    CLK_FREQ => CLK_FREQ,
    I2C_RATE => I2C_RATE
)
port map (
    clk          => clk,
    rst_n        => rst_n,
    scl          => scl,
    sda          => sda,

    dev1_ena     => aht_i2c_ena,
    dev1_addr    => aht_i2c_addr,
    dev1_rw      => aht_i2c_rw,
    dev1_data_wr => aht_i2c_data_wr,
    dev1_data_rd => aht_i2c_data_rd,
    dev1_busy    => aht_i2c_busy,
    dev1_ack_err => open,

    dev2_ena     => bmp_i2c_ena,
    dev2_addr    => bmp_i2c_addr,
    dev2_rw      => bmp_i2c_rw,
    dev2_data_wr => bmp_i2c_data_wr,
    dev2_data_rd => bmp_i2c_data_rd,
    dev2_busy    => bmp_i2c_busy,
    dev2_ack_err => open
);

    ds1302_driver_inst : entity work.ds1302_driver
generic map (
    CLK_FREQ => CLK_FREQ,
    DS_RATE  => I2C_RATE

```

```

)
port map (
    clk      => clk,
    rst_n    => rst_n,

    read_req => rtc_req,
    ready    => rtc_ready,

    o_sec    => r_sec,
    o_min    => r_min,
    o_hour   => r_hour,
    o_day    => r_day,
    o_mon     => r_mon,
    o_year   => r_year,

    rtc_sclk => rtc_sclk,
    rtc_ce   => rtc_ce,
    rtc_io   => rtc_io
);

aht21_driver_inst : entity work.aht21_driver
generic map (
    CLK_FREQ => CLK_FREQ,
    I2C_RATE => I2C_RATE
)
port map (
    clk      => clk,
    rst_n    => rst_n,

    i2c_ena   => aht_i2c_ena,
    i2c_addr  => aht_i2c_addr,
    i2c_rw    => aht_i2c_rw,
    i2c_data_wr => aht_i2c_data_wr,
    i2c_data_rd => aht_i2c_data_rd,
    i2c_busy  => aht_i2c_busy,

    start_meas => aht_start,
    hum_data   => aht_hum,
    temp_data  => aht_temp,
    data_ready => aht_ready
);

bmp180_driver_inst : entity work.bmp180_driver
generic map (
    CLK_FREQ => CLK_FREQ,
    I2C_RATE => I2C_RATE
)
port map (
    clk      => clk,
    rst_n    => rst_n,

```

```

i2c_ena      => bmp_i2c_ena,
i2c_addr     => bmp_i2c_addr,
i2c_rw       => bmp_i2c_rw,
i2c_data_wr  => bmp_i2c_data_wr,
i2c_data_rd  => bmp_i2c_data_rd,
i2c_busy     => bmp_i2c_busy,

start_meas   => bmp_start,
meas_cmd     => bmp_cmd,
data_out     => bmp_res,
data_ready   => bmp_ready
);

process (clk)
begin
    if rising_edge(clk) then
        if rst = '1' then
            state <= S_IDLE;
            sc_tx_start <= '0';
            bmp_start <= '0';
            aht_start <= '0';
            row_cnt <= 1;
            rtc_req <= '0';
            delay_cnt <= 0;

            sep_idx <= 1;
            col_idx <= 1;
            dot_idx <= 1;
            spc_idx <= 1;
        else
            sc_tx_start <= '0';
            bmp_start <= '0';
            aht_start <= '0';
            rtc_req <= '0';

            case state is
                when S_IDLE =>
                    req_mask <= "000";
                    if all_req = '1' then
                        req_mask <= "111";
                        state <= S_REQ_RTC;
                    elsif temp_req = '1' then
                        req_mask <= "001";
                        state <= S_REQ_RTC;
                    elsif pres_req = '1' then
                        req_mask <= "010";
                        state <= S_REQ_RTC;
                    elsif hum_req = '1' then
                        req_mask <= "100";
                        state <= S_REQ_RTC;
                    end if;
            end case;
        end if;
    end if;
end process;

```

```

when S_REQ_RTC =>
    rtc_req <= '1';
    state <= S_WAIT_RTC;

when S_WAIT_RTC =>
    if rtc_ready = '1' then
        p_sec <= r_sec;
        p_min <= r_min;
        p_hour <= r_hour;
        p_day <= r_day;
        p_mon <= r_mon;
        p_year <= r_year;
        if req_mask(0) = '1' or req_mask(2) =
'1' then

            state <= S_MEAS_AHT;
        else
            state <= S_MEAS_BMP_TEMP_CMD;
        end if;
    end if;

when S_MEAS_AHT =>
    aht_start <= '1';
    state <= S_WAIT_AHT;

when S_WAIT_AHT =>
    if aht_ready = '1' then
        latched_aht_temp <= aht_temp;
        latched_aht_hum <= aht_hum;
        state <= S_MEAS_BMP_TEMP_CMD;
    end if;

when S_MEAS_BMP_TEMP_CMD =>
    if req_mask(1) = '1' then
        bmp_cmd <= x"2E";
        state <= S_MEAS_BMP_TEMP_START;
    else
        state <= S_SEP0;
    end if;

when S_MEAS_BMP_TEMP_START =>
    bmp_start <= '1';
    state <= S_WAIT_BMP_TEMP;

when S_WAIT_BMP_TEMP =>
    if bmp_ready = '1' then
        latched_bmp_temp <= bmp_res;
        delay_cnt <= 0;
        state <= S_BMP_DELAY;
    end if;

```

```

when S_BMP_DELAY =>
    if delay_cnt < 50000 then
        delay_cnt <= delay_cnt + 1;
    else
        state <= S_MEAS_BMP_PRES_CMD;
    end if;

when S_MEAS_BMP_PRES_CMD =>
    bmp_cmd <= x"34";
    state <= S_MEAS_BMP_PRES_START;

when S_MEAS_BMP_PRES_START =>
    bmp_start <= '1';
    state <= S_WAIT_BMP_PRES;

when S_WAIT_BMP_PRES =>
    if bmp_ready = '1' then
        latched_bmp_pres <= bmp_res;
        state <= S_SEP0;
    end if;

when S_SEP0 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"7C";
        sc_tx_start <= '1';
        next_state_after_wait <= S_SPC0;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_SPC0 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"20";
        sc_tx_start <= '1';
        next_state_after_wait <= S_ROW_100;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_ROW_100 =>
    calc_int <= row_cnt;
    if sc_tx_busy = '0' then
        sc_tx_data <= to_dec_char((row_cnt/100)
rem 10);

        sc_tx_start <= '1';
        next_state_after_wait <= S_ROW_10;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_ROW_10 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_dec_char((calc_int/10)
rem 10);

```

```

        sc_tx_start <= '1';
        next_state_after_wait <= S_ROW_1;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_ROW_1 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_dec_char(calc_int rem
10);

        sc_tx_start <= '1';
        next_state_after_wait <= S_SEP1;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_HOUR_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_hour(7 downto
4));

        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_HOUR_L;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_HOUR_L =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_hour(3 downto
0));

        sc_tx_start <= '1';
        next_state_after_wait <= S_COL;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_MIN_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_min(7 downto 4));
        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_MIN_L;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_MIN_L =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_min(3 downto 0));
        sc_tx_start <= '1';
        next_state_after_wait <= S_COL;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_SEC_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_sec(7 downto 4));

```

```

        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_SEC_L;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_SEC_L =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_sec(3 downto 0));
        sc_tx_start <= '1';
        next_state_after_wait <= S_SPC;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_DAY_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_day(7 downto 4));
        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_DAY_L;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_DAY_L =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_day(3 downto 0));
        sc_tx_start <= '1';
        next_state_after_wait <= S_DOT;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_MON_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_mon(7 downto 4));
        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_MON_L;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_MON_L =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_mon(3 downto 0));
        sc_tx_start <= '1';
        next_state_after_wait <= S_DOT;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PR_YEAR_H =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(p_year(7 downto
4));

        sc_tx_start <= '1';
        next_state_after_wait <= S_PR_YEAR_L;

```

```

        state <= S_WAIT_TX_BUSY;
    end if;

    when S_PR_YEAR_L =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(p_year(3 downto
0));

            sc_tx_start <= '1';
            next_state_after_wait <= S_SEP1;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_CHECK_TEMP =>
        if req_mask(0) = '1' then
            state <= S_TEMP_CALC;
        else
            space_cnt <= 15;
            state <= S_ALIGNMENT_EMPTY;
        end if;

    when S_TEMP_CALC =>
        math_32 <= resize(unsigned(latched_aht_temp)
* 200, 32);

        state <= S_TEMP_SIGN;

    when S_TEMP_SIGN =>
        v_temp_int <= to_integer(math_32(27 downto
20));

        if v_temp_int < 50 then
            calc_int <= 50 - v_temp_int;
        else
            calc_int <= v_temp_int - 50;
        end if;
        if sc_tx_busy = '0' then
            if v_temp_int < 50 then
                sc_tx_data <= x"2D";
            else
                sc_tx_data <= x"2B";
            end if;
            sc_tx_start <= '1';
            next_state_after_wait <= S_TEMP_10;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_TEMP_10 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_dec_char((calc_int/10)
rem 10);

            sc_tx_start <= '1';
            next_state_after_wait <= S_TEMP_1;
            state <= S_WAIT_TX_BUSY;

```

```

        end if;

when S_TEMP_1 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_dec_char(calc_int rem
10);

        sc_tx_start <= '1';
        next_state_after_wait <= S_TEMP_C;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_TEMP_C =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"43";
        sc_tx_start <= '1';
        space_cnt <= 11;
        next_state_after_wait <=
S_ALIGNMENT_EMPTY;

        state <= S_WAIT_TX_BUSY;
    end if;

when S_CHECK_PRES =>
    if req_mask(1) = '1' then
        state <= S_PRES_H4;
    else
        space_cnt <= 13;
        state <= S_ALIGNMENT_EMPTY;
    end if;

when S_PRES_H4 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(latched_bmp_pres(15
downto 12));

        sc_tx_start <= '1';
        next_state_after_wait <= S_PRES_H3;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PRES_H3 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(latched_bmp_pres(11
downto 8));

        sc_tx_start <= '1';
        next_state_after_wait <= S_PRES_H2;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_PRES_H2 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_hex(latched_bmp_pres(7
downto 4));

```

```

        sc_tx_start <= '1';
        next_state_after_wait <= S_PRES_H1;
        state <= S_WAIT_TX_BUSY;
    end if;

    when S_PRES_H1 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(latched_bmp_pres(3
downto 0));

            sc_tx_start <= '1';
            next_state_after_wait <= S_SPC;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_TEMP_H4 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(latched_bmp_temp(15
downto 12));

            sc_tx_start <= '1';
            next_state_after_wait <= S_TEMP_H3;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_TEMP_H3 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(latched_bmp_temp(11
downto 8));

            sc_tx_start <= '1';
            next_state_after_wait <= S_TEMP_H2;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_TEMP_H2 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(latched_bmp_temp(7
downto 4));

            sc_tx_start <= '1';
            next_state_after_wait <= S_TEMP_H1;
            state <= S_WAIT_TX_BUSY;
        end if;

    when S_TEMP_H1 =>
        if sc_tx_busy = '0' then
            sc_tx_data <= to_hex(latched_bmp_temp(3
downto 0));

            sc_tx_start <= '1';
            space_cnt <= 4;
            next_state_after_wait <=
S_ALIGNMENT_EMPTY;

            state <= S_WAIT_TX_BUSY;
        end if;

```

```

when S_CHECK_HUM =>
    if req_mask(2) = '1' then
        state <= S_HUM_CALC;
    else
        space_cnt <= 12;
        state <= S_ALIGNMENT_EMPTY;
    end if;

when S_HUM_CALC =>
    math_32 <= resize(unsigned(latched_aht_hum)
* 100, 32);

    state <= S_HUM_10;

when S_HUM_10 =>
    calc_int <= to_integer(math_32(26 downto
20));

    if sc_tx_busy = '0' then
        sc_tx_data <=
to_dec_char((to_integer(math_32(26 downto 20))/10) rem 10);
        sc_tx_start <= '1';
        next_state_after_wait <= S_HUM_1;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_HUM_1 =>
    if sc_tx_busy = '0' then
        sc_tx_data <= to_dec_char(calc_int rem
10);

        sc_tx_start <= '1';
        next_state_after_wait <= S_HUM_PERC;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_HUM_PERC =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"25";
        sc_tx_start <= '1';
        space_cnt <= 9;
        next_state_after_wait <=
S_ALIGNMENT_EMPTY;

        state <= S_WAIT_TX_BUSY;
    end if;

when S_CR =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"0D";
        sc_tx_start <= '1';
        next_state_after_wait <= S_LF;
        state <= S_WAIT_TX_BUSY;
    end if;

```

```

when S_LF =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"0A";
        sc_tx_start <= '1';
        line_idx <= 0;
        next_state_after_wait <= S_PRINT_LINE;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_ALIGNMENT_EMPTY =>
    if space_cnt > 0 then
        if sc_tx_busy = '0' then
            sc_tx_data <= x"20";
            sc_tx_start <= '1';
            state <= S_ALIGNMENT_EMPTY_WAIT;
        end if;
    else
        state <= S_SEP1;
    end if;

when S_ALIGNMENT_EMPTY_WAIT =>
    if sc_tx_busy = '1' then
        space_cnt <= space_cnt - 1;
        state <= S_ALIGNMENT_EMPTY;
    end if;

when S_SPC =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"20";
        sc_tx_start <= '1';
        if spc_idx = 1 then
            next_state_after_wait <= S_PR_DAY_H;
        elsif spc_idx = 2 then
            next_state_after_wait <= S_TEMP_H4;
        end if;
        spc_idx <= spc_idx + 1;
        state <= S_WAIT_TX_BUSY;
    end if;

when S_COL =>
    if sc_tx_busy = '0' then
        sc_tx_data <= x"3A";
        sc_tx_start <= '1';
        if col_idx = 1 then
            next_state_after_wait <= S_PR_MIN_H;
        elsif col_idx = 2 then
            next_state_after_wait <= S_PR_SEC_H;
        end if;
        col_idx <= col_idx + 1;
        state <= S_WAIT_TX_BUSY;
    end if;

```

```

end if;

when S_DOT =>
  if sc_tx_busy = '0' then
    sc_tx_data <= x"2E";
    sc_tx_start <= '1';
    if dot_idx = 1 then
      next_state_after_wait <= S_PR_MON_H;
    elsif dot_idx = 2 then
      next_state_after_wait <=
S_PR_YEAR_H;

    end if;
    dot_idx <= dot_idx + 1;
    state <= S_WAIT_TX_BUSY;
  end if;

when S_SEP1 =>
  if sc_tx_busy = '0' then
    sc_tx_data <= x"20";
    sc_tx_start <= '1';
    next_state_after_wait <= S_SEP2;
    state <= S_WAIT_TX_BUSY;
  end if;

when S_SEP2 =>
  if sc_tx_busy = '0' then
    sc_tx_data <= x"7C";
    sc_tx_start <= '1';
    next_state_after_wait <= S_SEP3;
    state <= S_WAIT_TX_BUSY;
  end if;

when S_SEP3 =>
  if sc_tx_busy = '0' then
    sc_tx_data <= x"20";
    sc_tx_start <= '1';
    if sep_idx = 1 then
      next_state_after_wait <=
S_PR_HOUR_H;

    elsif sep_idx = 2 then
      next_state_after_wait <=
S_CHECK_TEMP;

    elsif sep_idx = 3 then
      next_state_after_wait <=
S_CHECK_PRES;

    elsif sep_idx = 4 then
      next_state_after_wait <=
S_CHECK_HUM;

    elsif sep_idx = 5 then
      next_state_after_wait <= S_CR;
    end if;

```

```

        sep_idx <= sep_idx + 1;
        state <= S_WAIT_TX_BUSY;
    end if;

    when S_WAIT_TX_BUSY =>
        if sc_tx_busy = '1' then
            state <= next_state_after_wait;
        end if;

    when S_PRINT_LINE =>
        if sc_tx_busy = '0' then
            if line_idx = 0 or line_idx = 6 or
line_idx = 26 or line_idx = 44 or line_idx = 60 or line_idx = 75
then
                sc_tx_data <= x"2B";
            elsif line_idx = 76 then
                sc_tx_data <= x"0D";
            elsif line_idx = 77 then
                sc_tx_data <= x"0A";
            else
                sc_tx_data <= x"2D";
            end if;
            sc_tx_start <= '1';
            state <= S_PRINT_LINE_WAIT;
        end if;

    when S_PRINT_LINE_WAIT =>
        if sc_tx_busy = '1' then
            if line_idx < 77 then
                line_idx <= line_idx + 1;
                state <= S_PRINT_LINE;
            else
                if row_cnt < 999 then
                    row_cnt <= row_cnt + 1;
                else
                    row_cnt <= 1;
                end if;
                sep_idx <= 1;
                col_idx <= 1;
                dot_idx <= 1;
                spc_idx <= 1;
                state <= S_IDLE;
            end if;
        end if;

    when others =>
        state <= S_IDLE;
    end case;
end if;
end if;
end process;

```

```

end Behavioral;

    Файл i2c_bus_arbiter.vhd

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

entity i2c_bus_arbiter is
    generic (
        CLK_FREQ : integer;
        I2C_RATE : integer
    );
    port (
        clk          : in     std_logic;
        rst_n        : in     std_logic;
        scl          : inout  std_logic;
        sda          : inout  std_logic;

        dev1_ena     : in     std_logic;
        dev1_addr    : in     std_logic_vector(6 downto 0);
        dev1_rw      : in     std_logic;
        dev1_data_wr  : in     std_logic_vector(7 downto 0);
        dev1_data_rd  : out    std_logic_vector(7 downto 0);
        dev1_busy    : out    std_logic;
        dev1_ack_err  : out    std_logic;

        dev2_ena     : in     std_logic;
        dev2_addr    : in     std_logic_vector(6 downto 0);
        dev2_rw      : in     std_logic;
        dev2_data_wr  : in     std_logic_vector(7 downto 0);
        dev2_data_rd  : out    std_logic_vector(7 downto 0);
        dev2_busy    : out    std_logic;
        dev2_ack_err  : out    std_logic
    );
end i2c_bus_arbiter;

architecture Behavioral of i2c_bus_arbiter is

    type state_t is (IDLE, DEV1_ACTIVE, DEV2_ACTIVE);
    signal state : state_t := IDLE;

    signal master_ena      : std_logic;
    signal master_addr     : std_logic_vector(6 downto 0);
    signal master_rw       : std_logic;
    signal master_data_wr  : std_logic_vector(7 downto 0);
    signal master_data_rd  : std_logic_vector(7 downto 0);
    signal master_busy     : std_logic;
    signal master_ack_err  : std_logic;

begin

```

```

i2c_master_inst : entity work.i2c_master
generic map(
    CLK_FREQ => CLK_FREQ,
    I2C_RATE => I2C_RATE
)
port map(
    clk      => clk,
    rst_n    => rst_n,
    scl      => scl,
    sda      => sda,

    ena      => master_ena,
    addr     => master_addr,
    rw       => master_rw,
    data_wr  => master_data_wr,
    data_rd  => master_data_rd,
    busy     => master_busy,
    ack_error => master_ack_err
);

process (clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            state <= IDLE;
        else
            case state is
                when IDLE =>
                    if dev1_ena = '1' then
                        state <= DEV1_ACTIVE;
                    elsif dev2_ena = '1' then
                        state <= DEV2_ACTIVE;
                    end if;

                    when DEV1_ACTIVE =>
                        if (dev1_ena = '0' and master_busy = '0')
then
                            state <= IDLE;
                        end if;

                    when DEV2_ACTIVE =>
                        if (dev2_ena = '0' and master_busy = '0')
then
                            state <= IDLE;
                        end if;

                    when others =>
                        state <= IDLE;
                    end case;
                end if;
            end if;
        end if;
    end if;
end if;

```

```

end process;

master_ena <= dev1_ena when state = DEV1_ACTIVE else
               dev2_ena when state = DEV2_ACTIVE else '0';

master_addr <= dev1_addr when state = DEV1_ACTIVE else
               dev2_addr when state = DEV2_ACTIVE else (others
=> '0');

master_rw <= dev1_rw when state = DEV1_ACTIVE else
             dev2_rw when state = DEV2_ACTIVE else '0';

master_data_wr <= dev1_data_wr when state = DEV1_ACTIVE else
                 dev2_data_wr when state = DEV2_ACTIVE else
(others => '0');

dev1_data_rd <= master_data_rd;
dev2_data_rd <= master_data_rd;
dev1_ack_err <= master_ack_err;
dev2_ack_err <= master_ack_err;

dev1_busy <= master_busy when state = DEV1_ACTIVE else '0'
              when state = IDLE else '1';

dev2_busy <= master_busy when state = DEV2_ACTIVE else '0'
              when state = IDLE else '1';

end Behavioral;

```

Файл i2c_master.vhd

```

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

entity i2c_master is
  generic (
    CLK_FREQ : integer;
    I2C_RATE : integer
  );
  port (
    clk      : in      std_logic;
    rst_n    : in      std_logic;
    scl      : inout   std_logic;
    sda      : inout   std_logic;

    ena      : in      std_logic;
    addr     : 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);
    busy     : out     std_logic;
  );
end entity i2c_master;

```

```

        ack_error : buffer std_logic
    );
end i2c_master;

architecture Behavioral of i2c_master is

    constant divider : integer := (CLK_FREQ/I2C_RATE)/4;

    type state_t is (READY, START, COMMAND, SLV_ACK1, WR, RD,
SLV_ACK2, MSTR_ACK, STOP);
    signal state      : state_t;
    signal data_clk   : std_logic;
    signal data_clk_prev : std_logic;
    signal scl_clk    : std_logic;
    signal scl_ena    : std_logic := '0';
    signal sda_int    : std_logic := '1';
    signal sda_ena_n  : std_logic;
    signal addr_rw    : std_logic_vector(7 downto 0);
    signal data_tx     : std_logic_vector(7 downto 0);
    signal data_rx     : std_logic_vector(7 downto 0);
    signal bit_cnt     : integer range 0 to 7 := 7;
    signal stretch    : std_logic := '0';

begin

    process (clk)
        variable count : integer range 0 to divider*4;
    begin
        if rising_edge(clk) then
            if rst_n = '0' then
                stretch <= '0';
                count := 0;
            else
                data_clk_prev <= data_clk;

                if count = divider*4-1 then
                    count := 0;
                elsif stretch = '0' then
                    count := count + 1;
                end if;

                case count is
                    when 0 to divider-1 =>
                        scl_clk <= '0';
                        data_clk <= '0';

                    when divider to divider*2-1 =>
                        scl_clk <= '0';
                        data_clk <= '1';

                    when divider*2 to divider*3-1 =>

```

```

        scl_clk <= '1';
        if scl = '0' then
            stretch <= '1';
        else
            stretch <= '0';
        end if;
        data_clk <= '1';

        when others =>
            scl_clk <= '1';
            data_clk <= '0';
        end case;
    end if;
end if;
end process;

process (clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            state <= READY;
            busy <= '1';
            scl_ena <= '0';
            sda_int <= '1';
            ack_error <= '0';
            bit_cnt <= 7;
            data_rd <= "00000000";
        else
            if (data_clk = '1' and data_clk_prev = '0') then
                case state is
                    when READY =>
                        if ena = '1' then
                            busy <= '1';
                            addr_rw <= addr & rw;
                            data_tx <= data_wr;
                            state <= START;
                        else
                            busy <= '0';
                            state <= READY;
                        end if;
                    when START =>
                        busy <= '1';
                        sda_int <= addr_rw(bit_cnt);
                        state <= COMMAND;
                    when COMMAND =>
                        if bit_cnt = 0 then
                            sda_int <= '1';
                            bit_cnt <= 7;
                            state <= SLV_ACK1;
                        end if;
                    end case;
                end if;
            end if;
        end if;
    end if;
end process;

```

```

else
    bit_cnt <= bit_cnt - 1;
    sda_int <= addr_rw(bit_cnt - 1);
    state <= COMMAND;
end if;

when SLV_ACK1 =>
    if addr_rw(0) = '0' then
        sda_int <= data_tx(bit_cnt);
        state <= WR;
    else
        sda_int <= '1';
        state <= RD;
    end if;

when WR =>
    busy <= '1';
    if bit_cnt = 0 then
        sda_int <= '1';
        bit_cnt <= 7;
        state <= SLV_ACK2;
    else
        bit_cnt <= bit_cnt - 1;
        sda_int <= data_tx(bit_cnt - 1);
        state <= WR;
    end if;

when RD =>
    busy <= '1';
    if bit_cnt = 0 then
        if (ena = '1' and addr_rw = addr &
rw) then

            sda_int <= '0';
        else
            sda_int <= '1';
        end if;
        bit_cnt <= 7;
        data_rd <= data_rx;
        state <= MSTR_ACK;
    else
        bit_cnt <= bit_cnt - 1;
        state <= RD;
    end if;

when SLV_ACK2 =>
    if ena = '1' then
        busy <= '0';
        addr_rw <= addr & rw;
        data_tx <= data_wr;
        if (addr_rw = addr & rw) then
            sda_int <= data_wr(bit_cnt);

```

```

        state <= WR;
    else
        state <= START;
    end if;
else
    state <= STOP;
end if;

when MSTR_ACK =>
    if ena = '1' then
        busy <= '0';
        addr_rw <= addr & rw;
        data_tx <= data_wr;
        if (addr_rw = addr & rw) then
            sda_int <= '1';
            state <= RD;
        else
            state <= START;
        end if;
    else
        state <= STOP;
    end if;

when STOP =>
    busy <= '0';
    state <= READY;

when others =>
    null;
end case;

elsif (data_clk = '0' and data_clk_prev = '1') then
    case state is
        when START =>
            if scl_ena = '0' then
                scl_ena <= '1';
                ack_error <= '0';
            end if;

            when SLV_ACK1 =>
                if (sda /= '0' or ack_error = '1') then
                    ack_error <= '1';
                end if;

            when RD =>
                data_rx(bit_cnt) <= sda;

            when SLV_ACK2 =>
                if (sda /= '0' or ack_error = '1') then
                    ack_error <= '1';
                end if;
    end case;
end if;

```

```

        when STOP =>
            scl_ena <= '0';

        when others =>
            null;
    end case;
end if;
end if;
end if;
end process;

with state select
    sda_ena_n <= data_clk_prev when START,
                not data_clk_prev when STOP,
                sda_int when others;

scl <= '0' when (scl_ena = '1' and scl_clk = '0') else 'Z';
sda <= '0' when sda_ena_n = '0' else 'Z';

end Behavioral;

```

Файл ds1302_driver.vhd

```

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

entity ds1302_driver is
    generic (
        CLK_FREQ : integer;
        DS_RATE   : integer
    );
    port (
        clk      : in    std_logic;
        rst_n     : in    std_logic;

        read_req  : in    std_logic;
        ready     : out   std_logic;

        o_sec     : out   std_logic_vector(7 downto 0);
        o_min     : out   std_logic_vector(7 downto 0);
        o_hour    : out   std_logic_vector(7 downto 0);
        o_day     : out   std_logic_vector(7 downto 0);
        o_mon     : out   std_logic_vector(7 downto 0);
        o_year    : out   std_logic_vector(7 downto 0);

        rtc_sclk  : out   std_logic;
        rtc_ce    : out   std_logic;
        rtc_io    : inout std_logic
    );
end ds1302_driver;

```

architecture Behavioral of ds1302_driver is

```

    constant TICK_MAX : integer := CLK_FREQ/DS_RATE;
    signal tick_cnt    : integer range 0 to TICK_MAX := 0;
    signal sclk_edge   : std_logic := '0';

    type state_t is (
        IDLE,
        CMD_START,
        CMD_BITS,
        READ_BITS,
        STOP
    );
    signal state : state_t := IDLE;

    signal sclk_reg : std_logic := '0';
    signal ce_reg   : std_logic := '0';
    signal io_dir   : std_logic := '0';
    signal io_out   : std_logic := '0';

    signal bit_idx  : integer range 0 to 7 := 0;
    signal byte_idx : integer range 0 to 7 := 0;

    type reg_array is array (0 to 7) of std_logic_vector(7 downto
0);
    signal time_regs : reg_array;

    constant CMD_BURST_READ : std_logic_vector(7 downto 0) := x"BF";

begin

    rtc_sclk <= sclk_reg;
    rtc_ce <= ce_reg;
    rtc_io <= io_out when io_dir = '1' else 'Z';

    process (clk)
    begin
        if rising_edge(clk) then
            sclk_edge <= '0';
            if state /= IDLE then
                if tick_cnt = TICK_MAX - 1 then
                    tick_cnt <= 0;
                    sclk_edge <= '1';
                else
                    tick_cnt <= tick_cnt + 1;
                end if;
            else
                tick_cnt <= 0;
            end if;
        end if;
    end if;

```

```

end process;

process (clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            state <= IDLE;
            ready <= '0';
            sclk_reg <= '0';
            ce_reg <= '0';
            io_dir <= '0';
            o_sec <= (others => '0');
            o_min <= (others => '0');
            o_hour <= (others => '0');
            o_day <= (others => '0');
            o_mon <= (others => '0');
            o_year <= (others => '0');
        else
            case state is
                when IDLE =>
                    ready <= '0';
                    sclk_reg <= '0';
                    ce_reg <= '0';
                    io_dir <= '0';
                    if read_req = '1' then
                        state <= CMD_START;
                    end if;

                when CMD_START =>
                    ce_reg <= '1';
                    io_dir <= '1';
                    bit_idx <= 0;
                    io_out <= CMD_BURST_READ(0);
                    state <= CMD_BITS;
                    sclk_reg <= '0';

                when CMD_BITS =>
                    if sclk_edge = '1' then
                        if sclk_reg = '0' then
                            sclk_reg <= '1';
                        else
                            sclk_reg <= '0';
                            if bit_idx = 7 then
                                io_dir <= '0';
                                state <= READ_BITS;
                                bit_idx <= 0;
                                byte_idx <= 0;
                            else
                                bit_idx <= bit_idx + 1;
                                io_out <= CMD_BURST_READ(bit_idx
+ 1);

```

```

        end if;
    end if;
end if;

when READ_BITS =>
    if sclk_edge = '1' then
        if sclk_reg = '0' then
            sclk_reg <= '1';
            time_regs(byte_idx)(bit_idx) <=
rtc_io;

            else
                sclk_reg <= '0';
                if bit_idx = 7 then
                    bit_idx <= 0;
                    if byte_idx = 7 then
                        state <= STOP;
                    else
                        byte_idx <= byte_idx + 1;
                    end if;
                else
                    bit_idx <= bit_idx + 1;
                end if;
            end if;
        end if;

    when STOP =>
        ce_reg <= '0';
        sclk_reg <= '0';
        ready <= '1';
        o_sec  <= time_regs(0);
        o_min  <= time_regs(1);
        o_hour <= time_regs(2);
        o_day  <= time_regs(3);
        o_mon  <= time_regs(4);
        o_year <= time_regs(6);
        state <= IDLE;

        when others =>
            state <= IDLE;
        end case;
    end if;
end if;
end process;

```

end Behavioral;

Файл aht21_driver.vhd

```

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

```

```

entity aht21_driver is
  generic (
    CLK_FREQ : integer;
    I2C_RATE : integer
  );
  port (
    clk          : in  std_logic;
    rst_n        : in  std_logic;

    i2c_ena       : out std_logic;
    i2c_addr      : out std_logic_vector(6 downto 0);
    i2c_rw        : out std_logic;
    i2c_data_wr   : out std_logic_vector(7 downto 0);
    i2c_data_rd   : in  std_logic_vector(7 downto 0);
    i2c_busy      : in  std_logic;

    start_meas    : in  std_logic;
    hum_data      : out std_logic_vector(19 downto 0);
    temp_data     : out std_logic_vector(19 downto 0);
    data_ready    : out std_logic
  );
end aht21_driver;

architecture Behavioral of aht21_driver is

  constant MEAS_DELAY : integer := CLK_FREQ/12;
  constant AHT21_ADDR : std_logic_vector(6 downto 0) := "0111000";

  type state_t is (IDLE, CMD_START, CMD_33, CMD_00, CMD_STOP,
WAIT_MEAS, READ_START, READ_BYTES, READ_STOP);
  signal state : state_t := IDLE;

  signal busy_prev    : std_logic;
  signal raw_hum_reg  : std_logic_vector(19 downto 0);
  signal raw_temp_reg : std_logic_vector(19 downto 0);
  signal busy_cnt     : integer range 0 to 15 := 0;
  signal wait_cnt     : integer := 0;

begin

  i2c_addr <= AHT21_ADDR;
  hum_data <= raw_hum_reg;
  temp_data <= raw_temp_reg;

  process (clk)
  begin
    if rising_edge(clk) then
      if rst_n = '0' then
        i2c_ena <= '0';
        i2c_rw <= '0';
        i2c_data_wr <= (others => '0');

```

```

    busy_cnt <= 0;
    wait_cnt <= 0;
    data_ready <= '0';
    state <= IDLE;
    raw_hum_reg <= (others => '0');
    raw_temp_reg <= (others => '0');
else
    data_ready <= '0';
    case state is
        when IDLE =>
            if start_meas = '1' then
                busy_cnt <= 0;
                i2c_data_wr <= x"AC";
                i2c_rw <= '0';
                state <= CMD_START;
            end if;

        when CMD_START =>
            i2c_ena <= '1';

            if i2c_busy = '1' then
                i2c_data_wr <= x"33";
            end if;

            busy_prev <= i2c_busy;
            if (busy_prev = '0' and i2c_busy = '1') then
                busy_cnt <= busy_cnt + 1;
            end if;
            if (busy_cnt = 1 and i2c_busy = '0') then
                busy_cnt <= 0;
                state <= CMD_33;
            end if;

        when CMD_33 =>
            if i2c_busy = '1' then
                i2c_data_wr <= x"00";
            end if;

            busy_prev <= i2c_busy;
            if (busy_prev = '0' and i2c_busy = '1') then
                busy_cnt <= busy_cnt + 1;
            end if;
            if (busy_cnt = 1 and i2c_busy = '0') then
                busy_cnt <= 0;
                state <= CMD_00;
            end if;

        when CMD_00 =>
            busy_prev <= i2c_busy;
            if (busy_prev = '0' and i2c_busy = '1') then
                busy_cnt <= busy_cnt + 1;
            end if;
    end case;
end if;

```

```

        end if;
        if (busy_cnt = 1 and i2c_busy = '0') then
            busy_cnt <= 0;
            state <= CMD_STOP;
        end if;

when CMD_STOP =>
    i2c_ena <= '0';
    wait_cnt <= 0;
    state <= WAIT_MEAS;

when WAIT_MEAS =>
    if wait_cnt < MEAS_DELAY then
        wait_cnt <= wait_cnt + 1;
    else
        busy_cnt <= 0;
        i2c_rw <= '1';
        state <= READ_START;
    end if;

when READ_START =>
    i2c_ena <= '1';
    state <= READ_BYTES;
    busy_cnt <= 0;

when READ_BYTES =>
    busy_prev <= i2c_busy;
    if (busy_prev = '0' and i2c_busy = '1') then
        busy_cnt <= busy_cnt + 1;
    end if;

    if busy_cnt >= 7 then
        i2c_ena <= '0';
    end if;

    if (busy_cnt > 0 and i2c_busy = '0') then
        case busy_cnt is
            when 1 =>
                null;

                when 2 =>
                    raw_hum_reg(19 downto 12) <=
i2c_data_rd;

                when 3 =>
                    raw_hum_reg(11 downto 4) <=
i2c_data_rd;

                when 4 =>
                    raw_hum_reg(3 downto 0) <=
i2c_data_rd(7 downto 4);
        end case;
    end if;
end if;

```

```

raw_temp_reg(19 downto 16) <=
i2c_data_rd(3 downto 0);

when 5 =>
    raw_temp_reg(15 downto 8) <=
i2c_data_rd;

when 6 =>
    raw_temp_reg(7 downto 0) <=
i2c_data_rd;

when 7 =>
    null;
    state <= READ_STOP;

when others =>
    null;
end case;
end if;

when READ_STOP =>
    i2c_ena <= '0';
    data_ready <= '1';
    state <= IDLE;

when others =>
    state <= IDLE;
end case;
end if;
end if;
end process;

end Behavioral;

```

Файл bmp180_driver.vhd

```

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

entity bmp180_driver is
    generic (
        CLK_FREQ : integer;
        I2C_RATE : integer
    );
    port (
        clk          : in  std_logic;
        rst_n        : in  std_logic;

        i2c_ena      : out std_logic;
        i2c_addr     : out std_logic_vector(6 downto 0);
        i2c_rw       : out std_logic;
    );
end entity;

```

```

        i2c_data_wr : out std_logic_vector(7 downto 0);
        i2c_data_rd : in  std_logic_vector(7 downto 0);
        i2c_busy    : in  std_logic;

        start_meas  : in  std_logic;
        meas_cmd    : in  std_logic_vector(7 downto 0);
        data_out     : out std_logic_vector(15 downto 0);
        data_ready   : out std_logic
    );
end bmp180_driver;

architecture Behavioral of bmp180_driver is

    constant MEAS_DELAY : integer := CLK_FREQ/33;
    constant BMP180_ADDR : std_logic_vector(6 downto 0) :=
"1110111";

    type state_t is (IDLE, WRITE_CMD, WRITE_VAL, WRITE_STOP,
WAIT_MEAS, READ_REQ, READ_MSB, READ_LSB, READ_STOP);
    signal state : state_t := IDLE;

    signal busy_prev : std_logic;
    signal result_reg : std_logic_vector(15 downto 0);
    signal busy_cnt   : integer range 0 to 5 := 0;
    signal wait_cnt   : integer := 0;

begin

    i2c_addr <= BMP180_ADDR;
    data_out <= result_reg;

    process (clk)
    begin
        if rising_edge(clk) then
            if rst_n = '0' then
                i2c_ena <= '0';
                i2c_rw <= '0';
                i2c_data_wr <= (others => '0');
                busy_cnt <= 0;
                wait_cnt <= 0;
                data_ready <= '0';
                state <= IDLE;
                result_reg <= (others => '0');
            else
                data_ready <= '0';
                case state is
                    when IDLE =>
                        if start_meas = '1' then
                            busy_cnt <= 0;
                            i2c_data_wr <= x"F4";
                            i2c_rw <= '0';

```

```

        state <= WRITE_CMD;
    end if;

when WRITE_CMD =>
    i2c_ena <= '1';

    if i2c_busy = '1' then
        i2c_data_wr <= meas_cmd;
    end if;

    busy_prev <= i2c_busy;
    if (busy_prev = '0' and i2c_busy = '1') then
        busy_cnt <= busy_cnt + 1;
    end if;
    if (busy_cnt = 1 and i2c_busy = '0') then
        busy_cnt <= 0;
        state <= WRITE_VAL;
    end if;

when WRITE_VAL =>
    busy_prev <= i2c_busy;
    if (busy_prev = '0' and i2c_busy = '1') then
        busy_cnt <= busy_cnt + 1;
    end if;
    if (busy_cnt = 1 and i2c_busy = '0') then
        busy_cnt <= 0;
        state <= WRITE_STOP;
    end if;

when WRITE_STOP =>
    i2c_ena <= '0';
    wait_cnt <= 0;
    state <= WAIT_MEAS;

when WAIT_MEAS =>
    if wait_cnt < MEAS_DELAY then
        wait_cnt <= wait_cnt + 1;
    else
        state <= READ_REQ;
        busy_cnt <= 0;
        i2c_data_wr <= x"F6";
        i2c_rw <= '0';
    end if;

when READ_REQ =>
    i2c_ena <= '1';

    if i2c_busy = '1' then
        i2c_rw <= '1';
    end if;

```

```

        busy_prev <= i2c_busy;
        if (busy_prev = '0' and i2c_busy = '1') then
            busy_cnt <= busy_cnt + 1;
        end if;
        if (busy_cnt = 1 and i2c_busy = '0') then
            busy_cnt <= 0;
            state <= READ_MSB;
        end if;

    when READ_MSB =>
        i2c_rw <= '1';
        busy_prev <= i2c_busy;
        if (busy_prev = '0' and i2c_busy = '1') then
            busy_cnt <= busy_cnt + 1;
        end if;
        if (busy_cnt = 1 and i2c_busy = '0') then
            result_reg(15 downto 8) <= i2c_data_rd;
            busy_cnt <= 2;
            state <= READ_LSB;
        end if;

    when READ_LSB =>
        busy_prev <= i2c_busy;
        if (busy_prev = '0' and i2c_busy = '1') then
            busy_cnt <= busy_cnt + 1;
        end if;

        if busy_cnt >= 3 then
            i2c_ena <= '0';
        end if;

        if (busy_cnt >= 3 and i2c_busy = '0') then
            result_reg(7 downto 0) <= i2c_data_rd;
            state <= READ_STOP;
        end if;

    when READ_STOP =>
        i2c_ena <= '0';
        data_ready <= '1';
        state <= IDLE;

    when others =>
        state <= IDLE;
    end case;
end if;
end if;
end process;

end Behavioral;

```

Файл ucf_file.ucf

					ПГУ 1.090301.35.001 ПЗ	Лист
						131
Изм.	Лист	№ докум.	Подпись	Дата		

```

NET "rx_serial" LOC = "C11" | IOSTANDARD = LVCMOS33;
NET "tx_serial" LOC = "D12" | IOSTANDARD = LVCMOS33;

NET "scl" LOC = "M5" | IOSTANDARD = I2C;
NET "sda" LOC = "P6" | IOSTANDARD = I2C;

NET "rtc_sclk" LOC = "E13" | IOSTANDARD = LVCMOS33;
NET "rtc_ce" LOC = "C13" | IOSTANDARD = LVCMOS33;
NET "rtc_io" LOC = "D14" | IOSTANDARD = LVCMOS33;

NET "button_rst" LOC = "C3" | IOSTANDARD = LVCMOS33;

NET "clk" LOC = "T8" | TNM_NET = "clk" | IOSTANDARD = LVCMOS33;
TIMESPEC "TS_clk" = PERIOD "clk" 50000 kHz;

```

					ПГУ 1.090301.35.001 ПЗ	Лист
						132
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В.2 – Листинг ПО для ПК

Файл Diplom_PC.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <windows.h>
#include <conio.h>
#include <iomanip>
#include <ctime>
#include <cctype>
#include <stdexcept>
#include "sqlite3.h"

using namespace std;

const int16_t AC1 = 7962;
const int16_t AC2 = -1225;
const int16_t AC3 = -14581;
const uint16_t AC4 = 34038;
const uint16_t AC5 = 25211;
const uint16_t AC6 = 15212;
const int16_t B1 = 6515;
const int16_t B2 = 50;
const int16_t MB = -32768;
const int16_t MC = -11786;
const int16_t MD = 2691;

double calculate_bmp180_pressure(uint32_t UP, uint32_t UT, uint8_t
oss = 0) {
    int32_t X1 = ((UT - (int32_t)AC6) * (int32_t)AC5) >> 15;
    int32_t X2 = ((int32_t)MC * 2048) / (X1 + (int32_t)MD);
    int32_t B5 = X1 + X2;

    int32_t B6 = B5 - 4000;
    X1 = ((int32_t)B2 * ((B6 * B6) >> 12)) >> 11;
    X2 = ((int32_t)AC2 * B6) >> 11;
    int32_t X3 = X1 + X2;
    int32_t B3 = (((((int32_t)AC1 * 4) + X3) * (1 << oss)) + 2) / 4;

    X1 = ((int32_t)AC3 * B6) >> 13;
    X2 = ((int32_t)B1 * ((B6 * B6) >> 12)) >> 16;
    X3 = ((X1 + X2) + 2) >> 2;
    uint32_t B4 = ((uint32_t)AC4 * (uint32_t)(X3 + 32768)) >> 15;
    uint32_t B7 = ((uint32_t)UP - (uint32_t)B3) * (50000UL >> oss);

    int32_t p;
    if (B7 < 0x80000000) {
        p = (B7 * 2) / B4;
```

```

    }
    else {
        p = (B7 / B4) * 2;
    }

    X1 = (p >> 8) * (p >> 8);
    X1 = (X1 * 3038) >> 16;
    X2 = (-7357 * p) >> 16;
    p = p + ((X1 + X2 + 3791) >> 4);

    return (double)p;
}

string trim(const string& s) {
    auto start = s.find_first_not_of(" \t\r\n");
    auto end = s.find_last_not_of(" \t\r\n");
    if (start == string::npos) return "";
    return s.substr(start, end - start + 1);
}

vector<string> split(const string& s, char delimiter) {
    vector<string> tokens;
    string token;
    istringstream tokenStream(s);
    while (getline(tokenStream, token, delimiter)) {
        tokens.push_back(trim(token));
    }
    return tokens;
}

double extract_number(const string& str) {
    string num_str = "";
    for (char c : str) {
        if (isdigit(c) || c == '.' || c == '-' || c == '+') {
            num_str += c;
        }
    }
    if (num_str.empty() || num_str == "+" || num_str == "-") return 0.0;
    try { return stod(num_str); }
    catch (...) { return 0.0; }
}

string generate_db_filename() {
    time_t t = time(nullptr);
    tm* now = localtime(&t);
    char buffer[128];
    sprintf(buffer, "sensors_data_%04d%02d%02d_%02d%02d%02d.db",
        now->tm_year + 1900, now->tm_mon + 1, now->tm_mday,
        now->tm_hour, now->tm_min, now->tm_sec);
    return string(buffer);
}

```

```

}

bool read_input_with_esc(string& out_str) {
    out_str = "";
    while (true) {
        if (_kbhit()) {
            unsigned char ch = _getch();

            if (ch == 0 || (ch == 224 && _kbhit())) {
                _getch();
                continue;
            }

            if (ch == 27) {
                return false;
            }
            else if (ch == '\r') {
                cout << "\n";
                return true;
            }
            else if (ch == '\b') {
                if (!out_str.empty()) {
                    out_str.pop_back();
                    cout << "\b\b";
                }
            }
            else if (ch >= 32) {
                out_str += ch;
                cout << ch;
            }
        }
        Sleep(10);
    }
}

bool is_valid_number(const string& str) {
    if (str.empty()) return false;
    for (char c : str) {
        if (!isdigit((unsigned char)c)) return false;
    }
    return true;
}

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    HANDLE hSerial = INVALID_HANDLE_VALUE;
    string portNum;
    int userBaudRate;
    cout << "Нажмите ESC для завершения работы\n";
}

```

```

while (true) {
    cout << "Введите номер COM-порта: ";
    if (!read_input_with_esc(portNum)) {
        cout << "\nВыход из программы...\n";
        return 0;
    }

    if (!is_valid_number(portNum)) {
        cout << "-> Ошибка: Введены недопустимые символы.
Пожалуйста, введите только цифры.\n\n";
        continue;
    }

    cout << "Введите скорость UART: ";
    string baudStr;
    if (!read_input_with_esc(baudStr)) {
        cout << "\nВыход из программы...\n";
        return 0;
    }

    if (!is_valid_number(baudStr)) {
        cout << "-> Ошибка: Введены недопустимые символы.
Пожалуйста, введите только цифры.\n\n";
        continue;
    }

    try {
        userBaudRate = stoi(baudStr);
    }
    catch (const out_of_range&) {
        cout << "-> Ошибка: Введено слишком большое значение для
скорости.\n\n";
        continue;
    }
    catch (const invalid_argument&) {
        cout << "-> Ошибка: Некорректный формат скорости.\n\n";
        continue;
    }

    string portName = "\\.\COM" + portNum;

    hSerial = CreateFileA(portName.c_str(), GENERIC_READ |
GENERIC_WRITE, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
    if (hSerial == INVALID_HANDLE_VALUE) {
        cout << "-> Ошибка: Не удалось подключиться к COM" <<
portNum << ". Возможно, он отключен или занят.\n\n";
        continue;
    }

    SetupComm(hSerial, 16384, 16384);

```

```

DCB dcb = { 0 };
dcb.DCBlength = sizeof(dcb);
GetCommState(hSerial, &dcb);

dcb.BaudRate = userBaudRate;
dcb.ByteSize = 8;
dcb.StopBits = ONESTOPBIT;
dcb.Parity = NOPARITY;

if (!SetCommState(hSerial, &dcb)) {
    cout << "-> Ошибка: Не удалось применить настройки.
Возможно, скорость " << userBaudRate << " не поддерживается.\n\n";
    CloseHandle(hSerial);
    continue;
}

COMMTIMEOUTS timeouts = { 0 };
timeouts.ReadIntervalTimeout = MAXDWORD;
timeouts.ReadTotalTimeoutConstant = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;
SetCommTimeouts(hSerial, &timeouts);

break;
}

string db_filename = generate_db_filename();
sqlite3* db;
if (sqlite3_open(db_filename.c_str(), &db) != SQLITE_OK) {
    cerr << "Ошибка открытия БД SQLite!\n";
    CloseHandle(hSerial);
    return 1;
}

sqlite3_exec(db, "PRAGMA journal_mode = WAL; PRAGMA synchronous
= NORMAL;", nullptr, nullptr, nullptr);
sqlite3_busy_timeout(db, 5000);

const char* sql_create =
    "CREATE TABLE IF NOT EXISTS SensorLog ("
    "RequestNum TEXT, "
    "DateTime TEXT, "
    "\"Temperature(C)\" REAL, "
    "\"Pressure(Pa)\" REAL, "
    "\"Humidity(%)\" REAL);";
sqlite3_exec(db, sql_create, nullptr, nullptr, nullptr);

cout << "\nУспешное подключение к COM" << portNum << "
(Скорость: " << userBaudRate << ")\n";
cout << "База данных успешно создана: " << db_filename << "\n";
cout << "Нажмите R для начала работы\n\n";

```

```

char buffer[4096];
DWORD bytesRead, bytesWritten;
string line_buffer = "";

while (true) {
    bool activity = false;

    if (_kbhit()) {
        char ch = _getch();
        if (ch == 27) break;
        WriteFile(hSerial, &ch, 1, &bytesWritten, NULL);
        activity = true;
    }

    if (ReadFile(hSerial, buffer, sizeof(buffer) - 1,
&bytesRead, NULL) && bytesRead > 0) {
        activity = true;
        buffer[bytesRead] = '\\0';
        line_buffer += buffer;

        size_t pos;
        while ((pos = line_buffer.find('\\n')) != string::npos) {
            string line = line_buffer.substr(0, pos);
            line_buffer.erase(0, pos + 1);
            if (!line.empty() && line.back() == '\\r')
line.pop_back();

            auto parts = split(line, '|');
            bool is_data_row = false;

            if (parts.size() >= 2 && parts[1].length() > 0 &&
isdigit((unsigned char)parts[1][0])) {
                is_data_row = true;

                string num = parts.size() > 1 ? parts[1] : "";
                string date = parts.size() > 2 ? parts[2] : "";
                string temp = parts.size() > 3 ? parts[3] : "";
                string press = parts.size() > 4 ? parts[4] : "";
                string hum = parts.size() > 5 ? parts[5] : "";

                double calc_P = 0.0;
                string press_out = press;

                stringstream hex_ss(press);
                string hex1, hex2;
                if (hex_ss >> hex1 >> hex2) {
                    try {
                        uint32_t raw_p = stoul(hex1, nullptr,
16);

```

```

        uint32_t raw_t = stoul(hex2, nullptr,
16);

        calc_P =
calculate_bmp180_pressure(raw_p, raw_t, 0);
        press_out = to_string((int)calc_P) + "
Па";

    }
    catch (...) {}
}

double clean_temp = extract_number(temp);
double clean_hum = extract_number(hum);

cout << "| " << left
    << setw(3) << num << " | "
    << setw(17) << date << " | "
    << setw(15) << temp << " | "
    << setw(13) << press_out << " | "
    << setw(12) << hum << " |\n";

    string sql = "INSERT INTO SensorLog (RequestNum,
DateTime, \"Temperature(C)\", \"Pressure(Pa)\", \"Humidity(%)\" )
VALUES ('"

        + num + "', '" + date + "', "
        + to_string(clean_temp) + ", "
        + to_string(calc_P) + ", "
        + to_string(clean_hum) + ");";

    char* errMsg = nullptr;
    if (sqlite3_exec(db, sql.c_str(), nullptr,
nullptr, &errMsg) != SQLITE_OK) {
        sqlite3_free(errMsg);
    }
}

    if (!is_data_row) {
        cout << line << "\n";
    }
}

    if (!activity) {
        Sleep(1);
    }
}

    CloseHandle(hSerial);
    sqlite3_close(db);
    return 0;
}

```

Приложение Г
(обязательное)

Система опроса датчиков по интерфейсу I2C с передачей данных по UART и
занесением в базу данных SQLite

Презентация

					ПГУ 1.090301.35.001 ПЗ	Лист
						140
Изм.	Лист	№ докум.	Подпись	Дата		

Система опроса датчиков по интерфейсу I2C с передачей данных по UART и занесением в базу данных SQLite

Автор работы:

студент гр. 22ВВП2 Городничев Максим Игоревич

Научный руководитель:

к.т.н., доцент кафедры ВТ Юрова Ольга Викторовна

Цель и задачи работы

Цель работы: разработать программно-аппаратный комплекс для опроса внешних цифровых датчиков по интерфейсу I2C с последующей передачей данных на ПК по интерфейсу UART с целью дальнейшего хранения и использования.

Задачи:

- Разработать стенд
- Разработать программное обеспечение для ПЛИС
- Разработать программное обеспечение для ПК

Актуальность и аналоги

Актуальность: бюджетная и гибкая в настройке система, реализующая базовые принципы DAQ-систем

Аналоги:

- Промышленные DAQ-системы
- Осциллографы с функцией декодирования протоколов
- Логические анализаторы
- DIY-решения на базе микроконтроллеров

Инструменты разработки

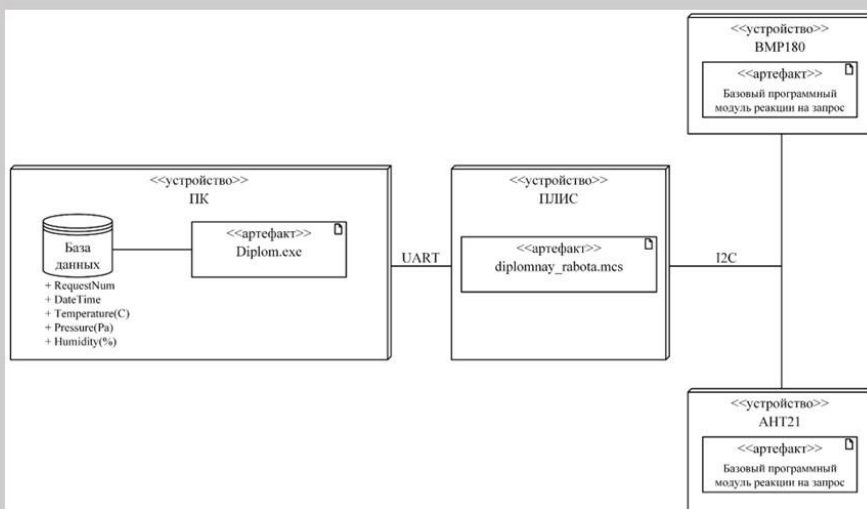
- Отладочная плата ALINX AX309
- ПЛИС Spartan-6 XC6SLX16
- Цифровые датчики АНТ21 и BMP180
- Среда разработки Xilinx ISE 14.7 и язык программирования VHDL
- Среда разработки Visual Studio 2022 и язык программирования C++
- СУБД SQLite

					ПГУ 1.090301.35.001 ПЗ	Лист
						142
Изм.	Лист	№ докум.	Подпись	Дата		

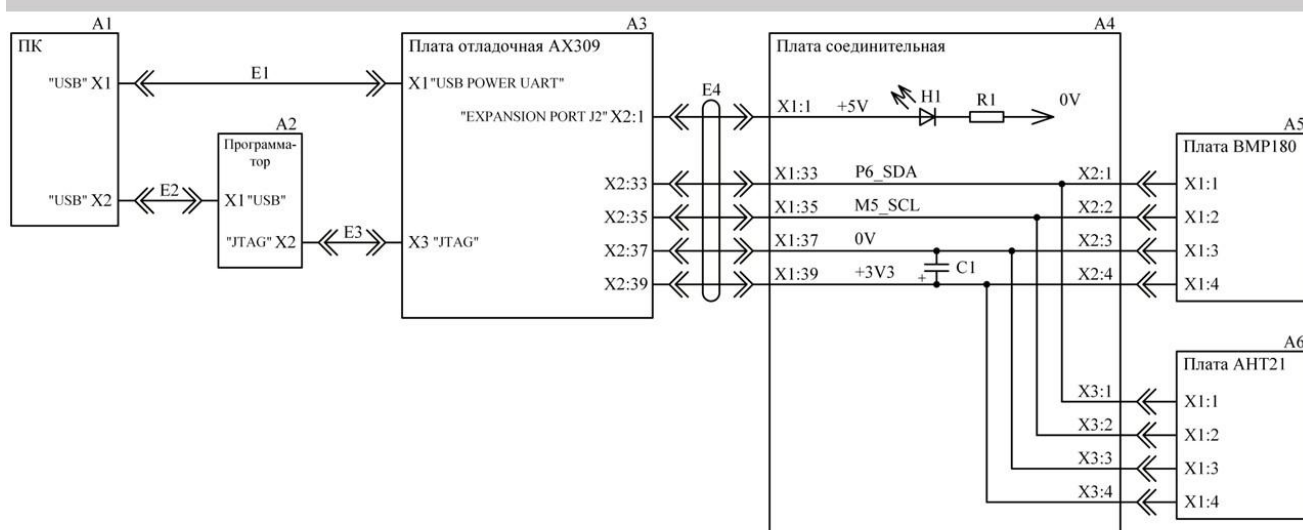
Архитектура

Модульная двухуровневая архитектура:

- **Нижний уровень:** сбор данных с датчиков по интерфейсу I2C и подготовка их к отправке на ПК
- **Связующее звено:** интерфейс UART
- **Верхний уровень:** настройка подключения, получение данных, математическая обработка и сохранение данных в БД



Электрическая схема соединений стенда



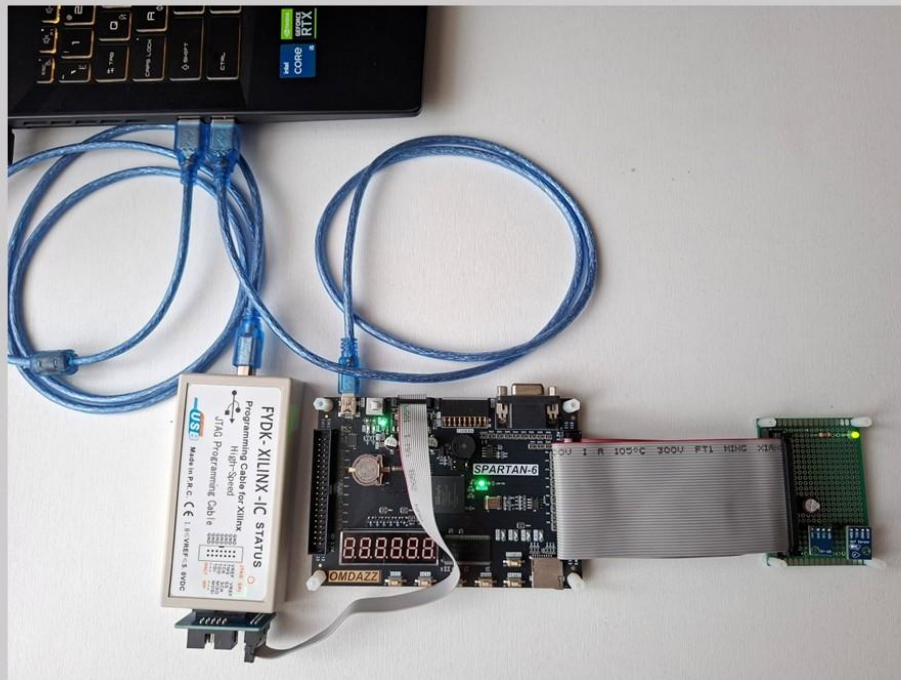
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.35.001 ПЗ

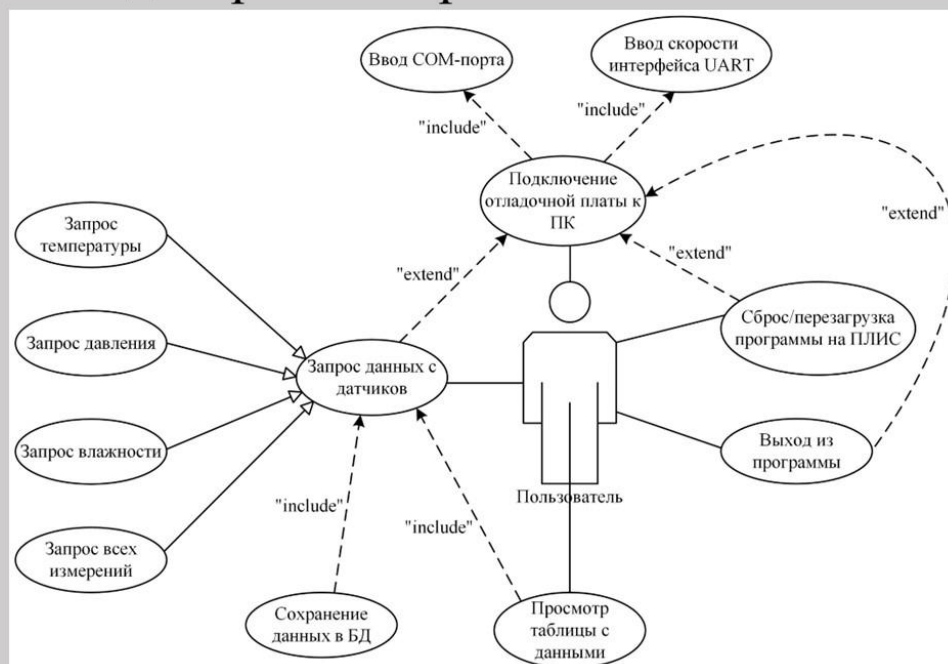
Лист

143

Внешний вид стенда



UML-диаграмма вариантов использования



Пользователь	ПК	ПЛИС	Датчики
запуска программу			
отображение координат с запросом ввести COM-порт			
ввод COM-порта			
запрос ввести скорость интерфейса UART			
ввод скорости UART			
сообщение об успешном подключении и об успешном создании БД	создание БД		
			БД
Исюр			
сброс/перезагрузка программы на ПЛИС (R)	отправка сигнала R		
отображение инструкций по управлению и данных таблицы	отправка инструкций по управлению и данных таблицы		
запрос температуры (1)	отправка сигнала "1"	отправка запроса датчику на получение данных	
отображение таблицы	формирование строки таблицы с данными	отправка данных	
	сохранение данных в БД		
запрос данных (2)	отправка сигнала "2"	отправка запроса датчику на получение данных	
отображение таблицы	формирование строки таблицы с данными	отправка данных	
	сохранение данных в БД		
запрос клавиатуры (3)	отправка сигнала "3"	отправка запроса датчику на получение данных	
отображение таблицы	формирование строки таблицы с данными	отправка данных	
	сохранение данных в БД		
запрос касс изображений (4)	отправка сигнала "4"	отправка запроса датчику на получение данных	
отображение таблицы	формирование строки таблицы с данными	отправка данных	
	сохранение данных в БД		
завершение работы программы (ESC)	сохранение и закрытие файла		
сохранение результатов			

```

Консоль отладки Microsoft V1 x + +
Введите номер COM-порта: -5
-> Ошибка: Введены недопустимые символы. Пожалуйста, введите только цифры.

Введите номер COM-порта: 5
Введите скорость UART: -115200
-> Ошибка: Введены недопустимые символы. Пожалуйста, введите только цифры.

Введите номер COM-порта: 5
Введите скорость UART: 9999999999999999999
-> Ошибка: Введено слишком большое значение для скорости.

Введите номер COM-порта: 7
Введите скорость UART: 115200
-> Ошибка: Не удалось подключиться к COM7. Возможно, он отключен или занят.

Введите номер COM-порта: 5
Введите скорость UART: 115200

Успешное подключение к COM5 (Скорость: 115200)
База данных успешно создана: sensors_data_20260506_175016.db
Нажмите R для начала работы

--- Sensor Dashboard ---
Use buttons: 1 = Temperature, 2 = Pressure, 3 = Humidity, 4 = All, R = Reset

| No. | Time and Date | Temperature (C) | Pressure (Pa) | Humidity (%) |
|-----|-----|-----|-----|-----|
| 001 | 17:52:16 06.05.26 | +22C |  |  |
| 002 | 17:52:34 06.05.26 |  | 98737 Па |  |
| 003 | 17:52:42 06.05.26 |  |  | 39% |
| 004 | 17:52:48 06.05.26 | +22C | 98726 Па | 39% |

C:\Users\Максим\source\repos\Diplom_PC\x64\Debug\Diplom.exe (процесс 15424) завершил работу с кодом 0 (0x0)
Нажмите любую клавишу, чтобы закрыть это окно.

```

DBeaver for SQLite - C:\Users\Maximov\source\repos\Diplom_PC\sensors_data_20260506_175016.db

Файл Редактирование Вид Инструменты Справка

Новая База Данных Открыть База Данных Сохранить Изменения Отменить Изменения Undo Open Project Save Project Прикрепить БД

Database Structure Browse Data Edit Pragma's Execute SQL

Таблица: SensorLog

RequestNum	DateTime	Temperature(C)	Pressure(Pa)	Humidity(%)
Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1 001	17:52:16 06.05.26	22.0	0.0	0.0
2 002	17:52:34 06.05.26	0.0	98737.0	0.0
3 003	17:52:42 06.05.26	0.0	0.0	39.0
4 004	17:52:48 06.05.26	22.0	98726.0	39.0

Редактирование анализ БД

Режим: Текст

1 1

Editing row=1, column=0
Type: Text / Numeric; Size: 1 character(s)

Схема БД

Имя	Тип	Схема
Таблицы (1)		
SensorLog		CREATE TABLE SensorLog (RequestNum
RequestNum...	TEXT	"RequestNum" TEXT
DateTime	TEXT	"DateTime" TEXT
Temperature...	REAL	"Temperature(C)" REAL
Pressure(Pa)	REAL	"Pressure(Pa)" REAL
Humidity(%)	REAL	"Humidity(%) " REAL
Индексы (0)		
Представления (0)		
Триггеры (0)		

Журнал SQL Графики Схема БД Удаленный сервер

Спасибо за внимание!

					ПГУ 1.090301.35.001 ПЗ
Изм.	Лист	№ докум.	Подпись	Дата	

Лист
146