

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 контроля климата в теплице»

Студент

Тумасов Вадим Валерьевич
(подпись, дата)

Руководитель

Токарев Андрей Николаевич
(подпись, дата)

Нормоконтролёр

Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 г. №

11)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
20.06.2024 № 31)

Секретарь ГЭК

Е.И. Гурин
(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2024

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«19» 04 2024 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Тумасов Вадим Валерьевич гр.20ВВП1 факультета вычислительной техники

2. Тема работы: «Система контроля климата в теплице»

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Токарев А.Н

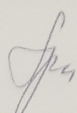
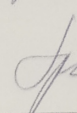
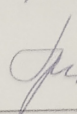
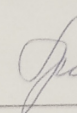
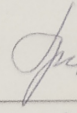
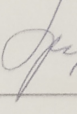
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать систему для осуществления контроля за показателями датчиков окружающей среды, отправки данных клиенту и осуществлять прием и обработку управляющих кадров от клиента. Также должен быть реализован графический интерфейс для отображения показателей состояния окружающей среды и возможности конфигурации исполнительных устройств теплицы.

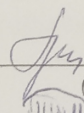
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки программно-аппаратного комплекса. Также необходимо изучить технологий для разработки комплекса, такие как *Java Swing, Arduino IDE*. Кроме этого необходимо провести анализ элементной базы и выбрать подходящую под проектирование аппаратной составляющей проекта.

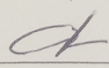
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2024- 30.04.2024	
2.	Анализ требований	10	01.05.2024- 02.05.2024	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2024- 07.05.2024	
4.	Описание алгоритмов	40	08.05.2024- 11.05.2024	
5.	Разработка программно-аппаратной части проекта	150	12.05.2024- 29.05.2024	
6.	Оформление пояснительной записки	24	30.05.2024- 31.05.2024	

Дата выдачи задания "19" апреля 2024 г.

Руководитель бакалаврской работы  (подпись, дата) Токарев А.Н

Задание к исполнению принял студент  (подпись, дата) Тумасов В.В

Работу к защите допустить
Декан факультета  (подпись, дата) Фионова Л.Р.
19.06.2024

Реферат

Бакалаврская работа содержит 120 листов, 27 рисунков, 7 таблиц, 7 источников и 3 приложения

СИСТЕМА КОНТРОЛЯ КЛИМАТА В ТЕПЛИЦЕ, ARDUINO IDE, NETBEANS, JAVA, ARDUINO MEGA 2560, JAVA SWING

Цель работы – разработать алгоритмы и практически реализовать макет автоматизированной теплицы.

Разработанный комплекс должен позволять отслеживать показатели состояния окружающей среды с использованием сервера, отправлять считанные данные клиенту и осуществлять прием и обработку управляющих кадров от клиента. Также должен быть реализован графический интерфейс на стороне клиента для отображения состояния окружающей среды и конфигурирования исполнительных устройств теплицы.

Результатом разработки является проект, удовлетворяющий поставленной задаче. А именно, разработан, реализован и протестирован комплекс для управления климатом в теплице. Разработка проводилась с использованием следующий технологий: Arduino IDE для разработки ПО сервера, отладочная плата Arduino Mega 2560, язык программирования JAVA, среда разработки Netbeans, библиотека для работы с графическим интерфейсом Java Swing.

					ПГУ 1.090301.25.001 ПЗ			
Изм.	Лист.	№ докум.	Подпись	Дата	«Система контроля климата в теплице»			
Разраб.		Тумасов В.В.		17.06.24				
Провер.		Токарев А.Н.		17.06.24				
Н. Контр.		Бычков А.С.		17.06.24				
Утвер.		Митрохин М.А.		17.06.24				
					Лит.	Лист.	Листов	
						2	120	

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Анализ предметной области.....	6
1.1 Анализ актуальности темы.....	6
1.2 Анализ готовых решений “умных” теплиц	7
1.3 Собственные автоматизированные теплицы и готовые решения.....	10
1.4 Постановка задачи.....	13
1.5 Выводы	16
2 Разработка комплекса автоматизированной теплицы.....	18
2.1 Выбор средств автоматизации.....	18
2.2 Выбор микроконтроллера и отладочной платы.....	21
2.3 Выбор датчиков, исполнительных устройств, устройства вывода.....	25
2.4 Анализ технологий для разработки ПО для Arduino Mega 2560	30
2.5 Выбор языка программирования для ПО клиента для ПК	32
2.6 Выбор среды разработки для ПО клиента для ПК	34
2.7 Выводы	35
3 Описание разработанного комплекса автоматизированной теплицы	37
3.1 Структура разработанного макета автоматизированной теплицы	37
3.2 Описание протокола взаимодействия клиента и сервера	39
3.3 Описание разработанного ПО.....	41
3.3.1 Клиент-серверная архитектура.....	41
3.3.2 Описание серверного ПО	41
3.3.3 Архитектура клиентского приложения	42

3.3.4 Процедура настройки интерфейса UART	45
3.3.5 Процедура отображения статуса теплицы	51
3.3.6 Процедура управления теплицей	54
3.4 Выводы	58
4 Безопасность жизнедеятельности.....	59
4.1 Требования к помещению и рабочему месту	59
4.2 Требования к освещению	59
4.3 Требования к режиму труда и отдыха.....	60
4.4 Рекомендации по использованию программного обеспечения	61
Заключение	62
Список используемых источников.....	64
Приложение А – UML-диаграммы.....	65
Приложение Б – Листинг программы	70
Приложение В – Презентация.....	108

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

ПО – программное обеспечение

ПК – персональный компьютер

Wi-Fi – (от англ. Wireless Fidelity) – беспроводная связь

UART – (от англ. Universal Asynchronous Receiver/Transmitter) – универсальный асинхронный приемопередатчик

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

I/O – (от англ. Input/Output) – ввод/вывод

IDE – (от англ. Integrated Development Environment) – интегрированная среда разработки

GPIO – (от англ. General-Purpose Input/Output) – универсальный ввод/вывод

RAM – (от англ. Random Access Memory) – память с произвольным доступом

Введение

В условиях современного мира, где вопросы продовольственной безопасности и рационального использования природных ресурсов становятся все более актуальными, сельское хозяйство играет ключевую роль. Традиционные методы ведения сельского хозяйства зачастую не обеспечивают необходимую эффективность и устойчивость. В этом контексте автоматизация и применение современных технологий представляют собой перспективное направление для решения стоящих перед человечеством задач.

Одной из инновационных технологий в сельском хозяйстве является концепция умной теплицы. Умная теплица — это высокотехнологичное сооружение, оснащенное системой автоматического контроля и управления климатическими условиями, освещением, поливом и другими факторами, влияющими на рост растений. Использование сенсоров, интернета вещей (IoT), систем машинного обучения и других современных технологий позволяет значительно повысить урожайность и качество продукции, а также снизить затраты на ресурсы и минимизировать воздействие на окружающую среду.

Цель данной дипломной работы — исследовать принципы функционирования автоматизированной умной теплицы, рассмотреть основные компоненты системы, алгоритмы управления, разработать алгоритмы и практически реализовать макет автоматизированной теплицы. Особое внимание уделено разработке и внедрению автоматизированной системы мониторинга и управления параметрами микроклимата, а также анализу экономической эффективности и экологических преимуществ использования умных теплиц.

В рамках исследования планируется рассмотреть современные решения в области умных теплиц, провести анализ существующих разработок, а также предложить собственные подходы к улучшению функционирования таких систем. Ожидается, что результаты данного исследования будут способствовать развитию инновационных технологий в сельском хозяйстве и окажут положительное влияние на устойчивое развитие аграрного сектора.

1 Анализ предметной области

В начале разработки программно-аппаратного комплекса для автоматизированной умной теплицы первостепенное значение имеет анализ конкретной области, в которой он будет применяться. Важно определить, кто будет конечным пользователем продукта, чтобы точно соответствовать их потребностям и ожиданиям. Основными пользователями умной теплицы могут быть аграрии, фермеры, исследователи и владельцы крупных сельскохозяйственных комплексов. Кроме того важное значение имеет нужная степень автоматизации для конкретного пользователя, так как это будет оказывать непосредственное влияние на сложность установки и количество нуждающихся в ней компонентов. Рассмотрение данных вопросов поможет в выявлении потенциальных проблем, которые могут возникнуть в процессе разработки проекта.

1.1 Анализ актуальности темы

Разработка автоматизированной умной теплицы является одной из самых актуальных задач современного сельского хозяйства. Рассмотрим основные аспекты, подчеркивающие актуальность данной темы:

- рациональное использование ресурсов. Современные теплицы позволяют значительно сократить использование воды, удобрений и энергии, что является критически важным в условиях нарастающего дефицита природных ресурсов. Автоматизация процессов позволяет точнее дозировать необходимые вещества и минимизировать потери, что способствует устойчивому развитию сельского хозяйства и снижению экологического следа;

- технологический прогресс. Развитие технологий, таких как интернет вещей (IoT), большие данные и искусственный интеллект (AI), открывает новые возможности для управления и оптимизации аграрных процессов. Интеграция этих технологий в тепличные хозяйства позволяет создать высокоэффективные и

саморегулирующиеся системы, способные адаптироваться к изменяющимся условиям окружающей среды;

– экономическая эффективность. Автоматизированные умные теплицы могут существенно снизить операционные затраты за счет сокращения ручного труда и оптимизации использования ресурсов. Несмотря на высокие первоначальные вложения, долгосрочные экономические выгоды включают увеличение урожайности, улучшение качества продукции и снижение затрат на ресурсы, что делает такие системы привлекательными для инвесторов и фермеров.

Таким образом, разработка автоматизированной умной теплицы представляет собой актуальное и перспективное направление, способное внести значительный вклад в решение глобальных продовольственных и экологических проблем. Внедрение таких систем способствует повышению эффективности и устойчивости сельского хозяйства, что делает эту тему крайне важной и востребованной в современных условиях.

1.2 Анализ готовых решений “умных” теплиц

Существует множество автоматизированных решений для теплиц, каждое из которых имеет свои особенности и преимущества. Вот несколько популярных решений, которые можно сравнить:

GreenIQ - компания-лидер в сегменте интеллектуальных технологий полива разрабатывает и производит инновационный контроллер для умного сада SMART GARDEN HUB, предназначенный для управления системой автоматического полива с учетом погодных данных (как на текущий момент, так и прогнозируемых), что позволяет экономить до 50% потребления воды на орошаемом участке. Компания GreenIQ уделяет большое внимание интегрируемости своей продукции с системами, веб-сервисами и мобильными приложениями для умного дома. Не случайно слоган компании в переводе на русский язык звучит так: «Ваш дом становится умнее». Компания GreenIQ имеет представительства более чем в десяти странах по всему миру. В России

официальным дистрибьютором продукции GreenIQ является компания Полив-Стандарт. Имеет функцию автоматизации, а также возможность управления через мобильное приложение.

Преимущества контроллера Smart Garden Hub:

- управление автоматическим поливом с помощью мобильного приложения GreenIQ;
- управление освещением с помощью мобильного приложения GreenIQ;
- экономия до 50% воды при орошении на участке.

Схема системы автоматизации на основе контроллера GreenIQ представлена на рисунке 1.

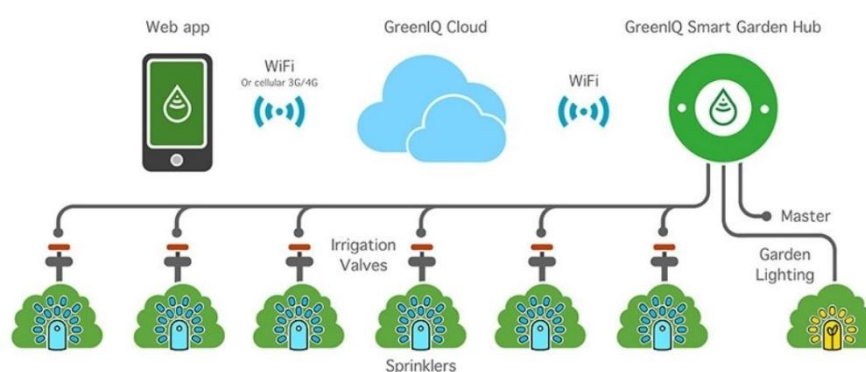


Рисунок 1 – Схема системы автоматизации на основе GreenIQ

Growlink - это система управления теплицей, которая включает в себя мониторинг параметров среды, автоматизацию полива, управление освещением и вентиляцией с помощью своего собственного контроллера. Он предназначен для обеспечения расширенных возможностей управления и автоматизации процесса сельского хозяйства в закрытых помещениях, включая гидропонику, вертикальное земледелие и теплицы. Внешний вид одной из моделей контроллера Growlink представлен на рисунке 2.



Рисунок 2 – Контроллер Growlink

Преимущества контроллера Growlink:

- удаленный мониторинг. Следите за показателями своей теплицы из любой точки мира с помощью мобильного приложения Growlink;
- управление ирригацией. Настраивайте графики полива, управляйте насосами;
- управление питательными веществами. Отслеживайте и корректируйте уровни питательных веществ в режиме реального времени, чтобы максимизировать здоровье растений и урожайность.

Autogrow предлагает полный спектр решений для автоматизации тепличного хозяйства, включая системы управления поливом, удобрениями, освещением и вентиляцией. Они также предоставляют аналитику и отчетность для оптимизации производства. Данный функционал достигается с использованием контроллера, произведенного данной компанией, под названием Multigrow. Multigrow - это универсальный контроллер нескольких зон выращивания. Он управляет до 8 различными зонами выращивания. Имеется возможность управления климатом и графиками орошения. Внешний вид одной из моделей контроллера Multigrow представлен на рисунке 3.



Рисунок 3 – Контроллер Multigrow

Преимущества контроллера Multigrow:

- легкость в установке. Multigrow легко устанавливается, используя сеть RS485 от контроллера и подключая ее к сенсорам окружающей среды;
- защита и безопасность. Все настройки и данные ежедневно копируются на защищенные серверы, что позволяет быстро восстановить их в случае сбоя, снизить риск потери данных и сохранить все настройки конфигурации;
- инструменты поддержки. Техническая команда Multigrow может войти в вашу систему для установки обновления для программного обеспечения и анализа ошибок в процессе работы контроллера, уменьшая возможные затраты на починку компонентов.

Анализ существующих аналогов позволяет нам сделать следующий вывод: большинство таких систем имеют весьма схожий функционал и возможности, что делает каждого из данных вариантов рассматриваемыми к покупке. Решающая роль в приобретении того или иного продукта отводится лишь финансовой стороне вопроса.

1.3 Собственные автоматизированные теплицы и готовые решения

Сравнение собственных автоматизированных теплиц и готовых решений можно провести по нескольким ключевым параметрам: затраты, гибкость, контроль, внедрение и поддержка.

Ниже представлены основные преимущества и недостатки собственных автоматизированных теплиц.

Преимущества:

- индивидуальный подход. Возможность создания системы, полностью адаптированной под конкретные условия и потребности, учет специфики местного климата и выращиваемых культур;
- гибкость. Возможность модернизации и доработки в процессе эксплуатации, легкость адаптации под новые технологии и требования;
- контроль качества. Полный контроль над выбором материалов и компонентов, возможность использования более качественных или специализированных компонентов;
- экономия на масштабировании. При увеличении масштабов возможно использовать наработанные решения и опыт, что может снизить затраты.

Недостатки:

- высокие стартовые затраты. Значительные расходы на проектирование, закупку материалов и оборудования, высокие затраты на настройку и тестирование системы;
- сложность разработки. Необходимость наличия специализированных знаний и опыта, время на разработку и настройку системы;
- риски и ошибки. Возможность допущения ошибок в проектировании и настройке, дополнительные затраты и временные потери при устранении проблем;
- время на реализацию. Длительный период разработки и внедрения системы.

Ниже представлены основные преимущества и недостатки готовых решений.

Преимущества:

- быстрое внедрение. Возможность быстрого запуска системы в эксплуатацию, снижение времени до начала использования;
- снижение рисков. Предварительно протестированные и настроенные системы, уменьшение вероятности ошибок и неисправностей;
- техническая поддержка. Поддержка и обслуживание от поставщика, помощь в решении возникающих проблем;
- обновления и модернизация. Поставщики часто предлагают обновления и улучшения, обеспечение актуальности системы без значительных затрат.

Недостатки:

- меньшая гибкость. Ограниченные возможности для изменения и настройки под специфические нужды, меньше возможностей для адаптации под уникальные условия;
- зависимость от поставщика. Необходимость обращаться к поставщику для модернизации или ремонта, потенциальные проблемы при изменении поставщика;
- стоимость. Возможная высокая стоимость готовых решений, зависимость от ценовой политики поставщика;
- стандартизированные решения. Могут не учитывать все уникальные потребности конкретного пользователя, возможны компромиссы в функциональности или эффективности.

Выбор между собственными автоматизированными теплицами и готовыми решениями зависит от конкретных условий и потребностей. Если важны индивидуализация и контроль, собственные разработки предпочтительнее, несмотря на высокие стартовые затраты и сложность. Готовые решения лучше подходят для быстрого внедрения и снижения рисков, но могут быть менее гибкими и дороже в долгосрочной перспективе. При выборе готовых решений большую роль имеет адаптируемость их функционала под конкретную область применения. В ходе анализа было установлено, что готовые решения предоставляют ограниченный функционал, который недостаточен для эффективного применения в различных условиях.

Исходя из результатов анализа, было принято решение о создании собственной “умной” теплицы. Основная цель проекта заключается в предоставлении конечному пользователю оптимального функционала и нужной степени автоматизации системы для обеспечения требуемого результата.

1.4 Постановка задачи

Автоматизированная теплица должна выполнять ряд функций, направленных на поддержание оптимальных условий для роста растений.

Основной функционал можно разделить на следующие категории:

- мониторинг окружающей среды;
- управление микроклиматом;
- управление поливом;
- управление освещением;
- уведомления и обратная связь;
- интерфейс пользователя.

Мониторинг окружающей среды.

Для поддержания оптимальных условий необходимо постоянно отслеживать ключевые параметры окружающей среды:

- температура воздуха. Использование датчиков температуры (например, DHT11 или DHT22) для контроля температуры внутри теплицы;
- влажность воздуха. Мониторинг влажности воздуха с помощью датчиков влажности (DHT11 или DHT22)[1];
- влажность почвы. Измерение влажности почвы при помощи емкостных датчиков влажности почвы;
- уровень освещенности. Использование фоторезисторов или других датчиков освещенности для измерения уровня света в теплице.

Управление микроклиматом. На основе данных, полученных от датчиков, система должна автоматически управлять устройствами, которые влияют на микроклимат.

Управление поливом. Автоматизация системы полива включает автоматическое включение насосов или клапанов для полива, когда влажность почвы опускается ниже заданного уровня.

Управление освещением. Автоматическое управление уровнем освещенности внутри теплицы включает возможность включения и выключения ламп или фитоламп при недостаточном естественном освещении.

Уведомления и обратная связь. Для оперативного реагирования на изменения в теплице система должна сохранять данные мониторинга для последующего анализа и принятия решений.

Интерфейс пользователя.

Для удобного управления и мониторинга система должна включать:

- пользовательский интерфейс. Веб-интерфейс или мобильное приложение для мониторинга параметров и управления устройствами;
- отображение данных в реальном времени. Графики и таблицы, отображающие текущие и исторические данные о состоянии теплицы.

Диаграмма вариантов использования программного-аппаратного комплекса автоматизированной теплицы представлена на рисунке 4.



Рисунок 4 – Диаграмма вариантов использования программно-аппаратного комплекса автоматизированной теплицы

Для достижения данного результата необходимо следовать следующим этапам выполнения проекта:

- разработать схему теплицы;
 - 1) определить размеры макета;
 - 2) спланировать расположение датчиков и исполнительных устройств.
- выбрать и приобрести компоненты;
 - 1) платформа Arduino (например, Arduino Uno);
 - 2) датчики (температуры, влажности воздуха, влажности почвы, освещенности);
 - 3) исполнительные устройства (реле, насосы для полива, вентиляторы, лампы);
 - 4) компоненты для соединения (провода, коннекторы, макетные платы).
- разработать систему сбора данных;
 - 1) написать программу для Arduino, которая будет считывать данные с датчиков;

2) организовать хранение данных (например, на SD-карте) или передачу данных на компьютер или сервер для дальнейшего анализа.

- разработать систему управления;

1) написать алгоритмы управления исполнительными устройствами на основе показаний датчиков;

2) реализовать контроль температуры (включение/выключение вентиляторов или нагревателей);

3) реализовать автоматический полив на основе данных влажности почвы.

- создать интерфейс для мониторинга и управления;

1) разработать пользовательский интерфейс (например, на основе веб-интерфейса или клиентского приложения) для мониторинга состояния теплицы и управления системой;

2) обеспечить возможность ручного управления устройствами.

- провести тестирование и отладку системы;

1) проверить работу всех датчиков и исполнительных устройств;

2) отладить программу, устранив возможные ошибки и оптимизировав алгоритмы управления;

3) провести тестирование макета в условиях, близких к реальным.

- документировать проект: описать архитектуру системы, используемые компоненты и их взаимосвязи.

1.5 Выводы

В разделе проанализирована область автоматизированных теплиц. Рассмотрены основные аспекты, подчёркивающие актуальность темы автоматизированных теплиц: рациональное использование ресурсов, технологический прогресс, экономическая эффективность. Рассмотрены распространённые готовые решения “умных” теплиц: GreenIQ, Growlink, Autogrow. На основе произведённого анализа сделан вывод, что готовые решения лучше подходят для быстрого внедрения и снижения рисков, но могут быть менее гибкими и дороже в долгосрочной перспективе. Если важны индивидуализация и

контроль, собственные разработки предпочтительнее, несмотря на высокие стартовые затраты и сложность. Поскольку важны индивидуализация разработки под различные условия применения, учитывая возможные суровые условия теплиц, было принято решение о создании собственной «умной» теплицы.

Были определены категории основного функционала разрабатываемого программно-аппаратного комплекса автоматизированной теплицы:

- мониторинг окружающей среды;
- управление поливом;
- управление дозатором удобрений;
- управление вентиляцией;
- уведомления и обратная связь;
- интерфейс пользователя.

2 Разработка комплекса автоматизированной теплицы

2.1 Выбор средств автоматизации

При разработке собственной автоматизированной теплицы можно использовать различные средства автоматизации, такие как микроконтроллеры, программируемые логические контроллеры (PLC), одноплатные компьютеры (SBC) и специализированные контроллеры для управления теплицами.

Микроконтроллеры.

Преимущества:

- экономичность. Микроконтроллеры, такие как Arduino, ESP8266, STM32, имеют низкую стоимость, что делает их доступными для небольших и средних проектов[2];
- низкое энергопотребление. Микроконтроллеры потребляют значительно меньше энергии по сравнению с одноплатными компьютерами, что важно для автономных систем;
- гибкость. Микроконтроллеры легко программируются и могут быть настроены для выполнения специфических задач. Они поддерживают широкий спектр датчиков и исполнительных устройств;
- простота использования. Доступность большого количества библиотек, документации и учебных материалов упрощает разработку и настройку системы;
- надежность. Микроконтроллеры могут работать в тяжелых условиях (высокая влажность, большие колебания температур), что важно для теплиц.

Недостатки:

- ограниченная вычислительная мощность. Микроконтроллеры не подходят для задач, требующих высокой производительности (например, сложные вычисления или обработка больших объемов данных в реальном времени);
- ограниченные возможности сети. Микроконтроллеры могут иметь ограничения по подключению к сети или интернету, хотя некоторые модели (например, ESP8266) предлагают встроенную поддержку Wi-Fi.

Программируемые логические контроллеры (PLC).

Преимущества:

- надежность. PLC известны своей высокой надежностью и долговечностью в промышленной среде;
- широкие возможности ввода/вывода. PLC обычно поддерживают множество каналов ввода/вывода, что позволяет управлять большим количеством датчиков и исполнительных устройств;
- соответствие стандартам. PLC соответствуют промышленным стандартам и имеют сертификаты, что важно для крупных коммерческих проектов.

Недостатки:

- высокая стоимость. PLC значительно дороже микроконтроллеров, что может быть экономически невыгодно для небольших теплиц;
- сложность программирования. Программирование PLC требует специальных знаний и навыков, что может быть затруднительно для новичков;
- ограниченная гибкость. PLC обычно менее гибки в плане расширения и модернизации по сравнению с микроконтроллерами.

Одноплатные компьютеры (SBC).

Преимущества:

- высокая вычислительная мощность. Одноплатные компьютеры, такие как Raspberry Pi, обладают высокой производительностью и могут выполнять сложные вычислительные задачи;
- многозадачность. Возможность выполнения нескольких задач одновременно благодаря полноценной операционной системе;
- широкие возможности подключения. Поддержка различных интерфейсов и сетевых подключений.

Недостатки:

- высокое энергопотребление. Одноплатные компьютеры потребляют больше энергии, что может быть критично для автономных систем;

- сложность настройки. Настройка и управление одноплатными компьютерами сложнее и требует больше времени по сравнению с микроконтроллерами;

- уязвимость к внешним условиям. Одноплатные компьютеры менее устойчивы к суровым условиям (влажность, температура) по сравнению с микроконтроллерами.

Специализированные контроллеры для теплиц.

Преимущества:

- оптимизация под конкретные задачи. Эти контроллеры разработаны специально для управления теплицами и обладают всеми необходимыми функциями;

- простота использования. Они часто имеют интуитивно понятный интерфейс и предустановленные программы, что упрощает их использование.

Недостатки:

- высокая стоимость. Специализированные контроллеры могут быть дорогими;

- ограниченная гибкость. Возможности изменения и настройки таких систем могут быть ограничены.

Выбор микроконтроллеров при разработке собственной автоматизированной теплицы обоснован следующими факторами:

- экономичность. Микроконтроллеры предлагают оптимальное соотношение цены и функциональности, что особенно важно для небольших и средних теплиц;

- гибкость и адаптивность. Возможность легкой адаптации под конкретные нужды и возможность модернизации системы;

- простота разработки и использования. Большое количество доступных ресурсов, библиотек и примеров, что упрощает процесс разработки и внедрения;

- энергоэффективность. Низкое энергопотребление микроконтроллеров делает их подходящими для автономных и энергоэффективных систем;

– надежность. Высокая устойчивость к суровым условиям эксплуатации, что важно для применения в теплицах.

Эти факторы делают микроконтроллеры идеальным выбором для автоматизации теплицы, предлагая баланс между стоимостью, гибкостью и функциональностью.

2.2 Выбор микроконтроллера и отладочной платы

В мире микроконтроллеров существует широкий ассортимент моделей, предназначенных для различных задач и соответствующих требованиям пользователей. Они производятся разными компаниями и имеют различные характеристики. Поэтому важно выбрать микроконтроллер, который подходит для нашего проекта. Мы сделали выбор из различных моделей, представленных разными производителями в том же сегменте рынка, чтобы определить подходящий микроконтроллер для реализации нашего проекта.

Рассмотрены следующие микроконтроллеры:

ATmega. Рассмотрен микроконтроллер ATmega2560[5]. Описание архитектуры ATmega2560: этот микроконтроллер использует модифицированную архитектуру Гарварда с 8-битным процессором RISC (Reduced Instruction Set Computing), что обеспечивает высокую производительность при низком энергопотреблении.

– память;

- 1) программная память. 16 КБ флэш-памяти для хранения кода программы;
- 2) оперативная память (SRAM). 512 байт для временного хранения данных;
- 3) EEPROM. 512 байт для хранения данных, которые должны сохраняться при выключении питания.

– тактовая частота: поддерживает частоту до 16 МГц, что обеспечивает достаточно высокую скорость обработки данных для большинства приложений;

– интерфейсы;

- 1) USB. Один из ключевых аспектов ATmega16U2 — встроенный интерфейс USB 2.0 с поддержкой полного и низкого скоростей. Это позволяет использовать

микроконтроллер для создания USB-устройств, таких как клавиатуры, мыши, и другие периферийные устройства;

2) USART. Два универсальных синхронных/асинхронных приёмопередатчика для последовательной связи;

3) SPI и I2C. Поддержка этих интерфейсов для связи с другими микроконтроллерами и периферийными устройствами.

- цифровые и аналоговые входы/выходы;

1) 22 цифровых I/O пина. Используются для подключения различных датчиков и исполнительных механизмов;

2) 4 аналоговых входа. Используются для измерения аналоговых сигналов.

- дополнительные функции.

1) встроенный регулятор напряжения. Поддерживает питание от 2.7 В до 5.5 В;

2) поддержка программирования через интерфейс ISP (In-System Programming);

3) поддержка различных режимов сна для снижения энергопотребления в неактивных состояниях.

Характеристики микроконтроллера ATmega2560:

- рабочее напряжение: 5В;
- входное напряжение (рекомендуемое): 7-12В;
- входное напряжение (предельное): 6-20В;
- цифровые Входы/Выходы: 54(14 из которых могут работать также как выходы ШИМ);

- аналоговые входы: 16;

- постоянный ток через вход/выход: 40 mA;

- постоянный ток для вывода: 3.3 В;

- флеш-память: 256 КВ(из которых 8 КВ используются для загрузчика);

- ОЗУ: 8 КВ;

- энергонезависимая память: 4 КВ;

- тактовая частота: 16 МГц.

Самой распространённой отладочной платой для ATmega2560 является Arduino Mega 2560. Arduino Mega 2560 — мощная плата на базе микроконтроллера ATmega2560, предназначенная для сложных проектов, требующих множества входов/выходов[6]. Она имеет 54 цифровых входа/выхода (из них 15 могут использоваться как ШИМ - выходы), 16 аналоговых входов, 4 UART (последовательных порта), кристалл с частотой 16 МГц, разъем USB, разъем питания, разъем ICSP и кнопку сброса. Плата поддерживается Arduino IDE, что делает её удобной для программирования и прототипирования. Благодаря большому количеству пинов и расширенным возможностям, Arduino Mega 2560 идеально подходит для проектов, требующих большого количества датчиков, моторов и других компонентов. На рисунке 5 изображена отладочная плата Arduino Mega 2560 с ATmega2560.

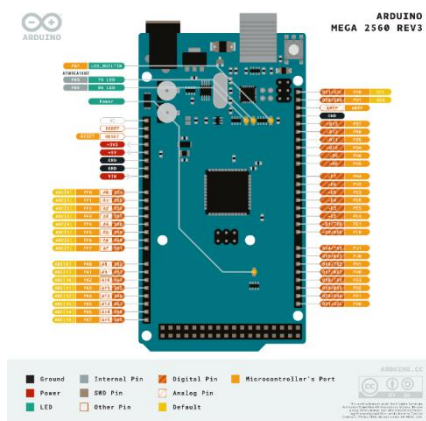


Рисунок 5 – Отладочная плата Arduino Mega 2560 с ATmega2560

STM32F4xx - это серия микроконтроллеров, разработанных и производимых компанией STMicroelectronics. Они являются частью семейства STM32, которое базируется на архитектуре ядра ARM Cortex-M. Общие характеристики серии микроконтроллеров STM32F4xx представлены в таблице 2.

Характеристики серии микроконтроллеров STM32F4xx:

- ядро ARM: ARM Cortex-M4;
- максимальная частота ядра, МГц: 180;
- GPIO (Цифровые входы/выходы): до 168;

- аналоговые входы (ADC): до 24 (12-бит);
- аналоговые выходы (DAC): до 2 (12-бит);
- объем Flash памяти (кБайт): 64-2056;
- объем RAM (кБайт): до 384;
- напряжение питания, В: 1.8-3.6;
- особенности: двойной Quad- SPI , интерфейс SDRAM , Ethernet MAC , интерфейс камеры.

Одной из распространённых отладочных плат для STM32F4xx является STM32F4 Discovery. STM32F4 Discovery — это отладочная плата, созданная компанией STMicroelectronics для разработки приложений на основе микроконтроллеров серии STM32F4. Эта плата оснащена микроконтроллером STM32F407VGT6, который работает на частоте 168 МГц и обладает 1 МБ флэш-памяти и 192 КБ оперативной памяти. Использование STM32F4xx микроконтроллеров позволяет создавать мощные и эффективные решения для широкого спектра приложений, включая автоматизацию, медицинскую технику, промышленные устройства, робототехнику и многое другое. На плате также присутствуют различные периферийные устройства, такие как акселерометр, микрофон, аудио ЦАП и множество GPIO пинов. На рисунке 6 изображена отладочная плата STM32F4 Discovery с STM32F4xx.



Рисунок 6 – Отладочная плата STM32F4 Discovery с STM32F4xx

MSP430F5xx/F6xx — это высокопроизводительное семейство 16-битных микроконтроллеров, разработанных для приложений с низким

энергопотреблением. Эти микроконтроллеры оснащены ядром MSP430 с тактовой частотой до 25 МГц, до 256 КБ флэш-памяти и до 18 КБ RAM. Они поддерживают множество периферийных устройств, таких как многоканальные 12-битные АЦП, ЦАП, таймеры с поддержкой ШИМ, UART, SPI, I2C и USB. Семейство MSP430F5xx/F6xx идеально подходит для энергоэффективных приложений, требующих высокой производительности и гибкости. Общие характеристики серии микроконтроллеров MSP430F5xx/F6xx представлены в таблице 3.

Характеристики серии микроконтроллеров MSP430F5xx/F6xx:

- максимальная частота ядра, МГц: 180;
- GPIO (Цифровые входы/выходы): До 25;
- аналоговые входы (ADC): до 16 (12-бит);
- аналоговые выходы (DAC): до 2 (12-бит);
- объем Flash памяти (кБайт): до 256;
- объем RAM (кБайт): до 18;
- напряжение питания, В: 1.8-3.6;
- особенности: низкое энергопотребление, широкий набор периферии.

Для выполнения проекта была выбрана модель ATmega2560 исходя из ее технических характеристик. Была выбрана плата ARDUINO MEGA 2560 REV3.

2.3 Выбор датчиков, исполнительных устройств, устройства вывода

В теплице необходимо измерять как минимум температуру, влажность воздуха и почвы. Для этого было решено использовать электронные датчики способные на измерение необходимых параметров. Сравнение характеристик датчиков температуры, из которых происходил выбор, представлено в таблице 1.

Таблица 1 - Сравнение датчиков температуры

Наименование характеристики	DHT11	DHT22	DHT21
--------------------------------	-------	-------	-------

Напряжение питания, В	3-5	3-5	3-5
Температура, мин °С	0 ±2	-40 ±0,5	-40 ±0,5
Температура, макс °С	50 ±2	125 ±0,5	80 ±0,5
Влажность, мин %	20	0	0
Влажность, макс %	80	100	100

Для измерения температуры выбран датчик DTH11, который является более бюджетным вариантом для использования в небольших системах. Данный датчик изображен на рисунке 7.

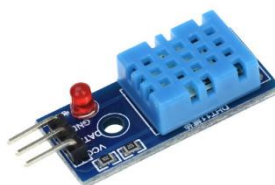


Рисунок 7 – Датчик DTH11

Кроме того, важно следить за влажностью почвы, чтобы определить, когда нужно поливать. Для этого нужно выбрать подходящий гигрометр, описание которого приведено в таблице 2.

Таблица 2. Сравнительная характеристика датчиков

Наименование характеристики	FC-28	YL-69	ICB510-01
Напряжение питания, В	3.3, 5	3.3 - 5	5 В, 12-24 В
Выход	цифровой и аналоговый	цифровой и аналоговый	аналоговый
Потребляемый ток, мА	35	35	50
Размер, мм	30×60	20×60	50×208

Был выбран датчик YL-69 из-за его компактных размеров, возможности работы с аналоговыми и цифровыми входами контроллера, а также из-за доступной цены. Этот датчик изображен на рисунке 8.

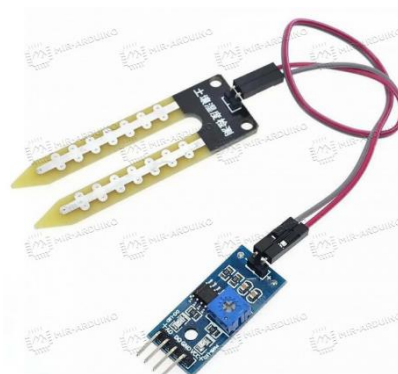


Рисунок 8 – Датчик YL-69

В данном проекте требуется обеспечение автоматического дозирования удобрений растений. Для этого будет использоваться сервопривод. В таблице 3 приведено сравнение характеристик трех моделей сервоприводов.

Таблица 3. Сравнительные характеристики сервоприводов

Наименование характеристики	SG-90	Turnigy TSS-10MG	ASMC-04B
Напряжение питания	5В	5В	12 В
Угол поворота ротора, градусы	360	360	300
Крутящий момент	1.6 кг/см	2.2 кг/см	90 кг/см
Внутренний интерфейс	аналоговый	цифровой	аналоговый
Направление	против часовой стрелки	против часовой стрелки	против часовой стрелки
Материал шестерней	нейлон	металл	металл
Материал корпуса	пластик	пластик	металл

Наименование характеристики	SG-90	Turnigy TSS-10MG	ASMC-04B
Длина проводов	20 см	20 см	30 см
Габариты	23.2 × 12.5 × 22 мм	23.1 x12.0 x24.9 мм	60*95*102 мм
Вес	9 г	15 г	530 г

Для данного проекта был выбран сервопривод SG-90. Он легкий, работает от напряжения 5 Вольт, как и большинство компонентов проекта, обладает достаточным крутящим моментом и компактными размерами. Этого будет достаточно для выполнения требуемых задач. Выбранный сервопривод показан на рисунке 9.



Рисунок 9 – Сервопривод SG-90

Для отображения показателей с датчиков требуется использования дисплея. В таблице 4 приведено сравнение трех моделей дисплеев.

Таблица 4. Сравнительные характеристики дисплеев

Наименование характеристики	1602 LCD	2004 LCD	OLED 128x64
Разрешение	16x2 символов	20x4 символов	128x64 пикселей
Технология	ЖК	ЖК	OLED
Интерфейс	Параллельный/I2C	Параллельный/I2C	SPI/I2C
Потребление энергии	Низкое	Низкое	Низкое (при черном фоне)
Стоимость	Низкая	Средняя	Высокая
Цвет	Монохромный	Монохромный	Монохромный/цветной

Для данного проекта было решено выбрать дисплей 1602 LCD, так как его функционала вполне хватает для требуемых задач. Так же одним из решающих

факторов при выборе является самая низкая стоимость из списка сравниваемых аналогов. Выбранный дисплей изображен на рисунке 10.

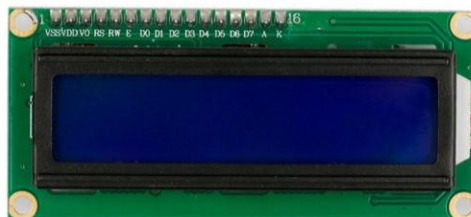


Рисунок 10 – Дисплей 1602 LCD

Для обеспечения функционирования системы закрытия люка необходимо использование шагового двигателя. В таблице 5 приведено сравнения трех моделей двигателей.

Таблица 5. Сравнительные характеристики шаговых двигателей

Наименование характеристики	28BYJ-48	NEMA 17	42BYGHW609
Тип	Униполярный	Биполярный	Биполярный
Количество шагов	32 (64/2048 с редуктором)	200	200
Угол шага	5.625°/шаг (с редуктором)	1.8°/шаг	1.8°/шаг
Напряжение питания	5V	3V-12V	12V
Стоимость	Низкая	Средняя	Средняя
Сложность управления	Низкая	Средняя	Средняя

В рамках данного проекта было решено использовать модель 28BYJ-48 из-за низкой сложности управления и низкой стоимости. Данная модель представлена на рисунке 11.

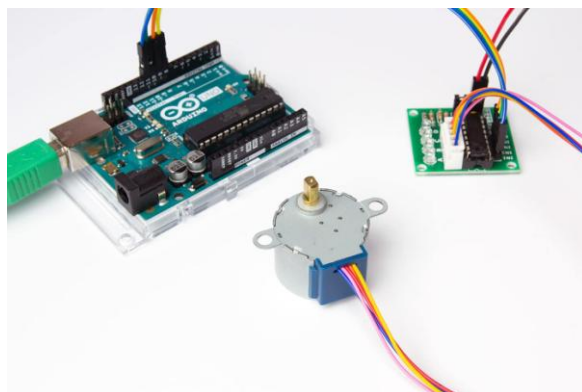


Рисунок 11 – Шаговый двигатель 28BYJ-48

2.4 Анализ технологий для разработки ПО для Arduino Mega 2560

В первую очередь нам необходимо проанализировать программные средства, которые мы будем использовать для настройки собранного макета. В процесс настройки будет входить: разработка системы сбора данных, создание системы управления исполнительными устройствами на основе показаний датчиков.

Для настройки и программирования Arduino Mega 2560 можно использовать следующие средства:

PlatformIO - это мощная среда для разработки встроенного ПО, которая поддерживает множество плат, включая Arduino Mega 2560. Она интегрируется с различными редакторами кода, такими как Visual Studio Code и Atom.

Основные особенности:

- многофункциональность;
- управление библиотеками;
- расширенные инструменты сборки;
- система проектов;
- интеграция с системами контроля версий;
- поддержка тестирования и отладки.

Eclipse с плагином Sloeber: Eclipse — это популярная IDE для разработки программного обеспечения, которую можно использовать с плагином Sloeber для работы с платами Arduino.

Основные особенности:

- интеграция с Eclipse;
- поддержка множества плат;
- мощный редактор кода;
- управление проектами;
- инструменты отладки и мониторинга.

Atmel Studio (Microchip Studio): это официальная IDE от производителя микроконтроллеров Atmel (ныне часть Microchip). Она поддерживает работу с микроконтроллерами AVR, которые используются в платах Arduino.

Основные особенности:

- профессиональная среда разработки;
- поддержка микроконтроллеров AVR и SAM;
- инструменты отладки;
- интеграция с платформой Arduino;
- средства анализа кода.

Arduino IDE : это официальная среда разработки, созданная специально для работы с платами Arduino. Она предоставляет удобный и простой интерфейс для написания, компиляции и загрузки программ на платы Arduino.

Основные особенности:

- интуитивно понятный интерфейс;
- поддержка множества плат;
- библиотеки;
- развитое сообщество разработчиков;
- примеры кода;
- компиляция и загрузка кода;
- монитор порта.

Вывод: исходя из проведенного анализа можно сделать вывод о том, что Arduino IDE является отличным инструментом для работы с текущей компонентой базой.

2.5 Выбор языка программирования для ПО клиента для ПК

В данном разделе будет проведено сравнения Java, C++ и Python в качестве языков программирования для создания нужного клиентского ПО.

Java.

Плюсы:

- кроссплатформенность. Java-приложения могут работать на различных операционных системах без изменения кода, благодаря JVM (Java Virtual Machine);
- богатая экосистема. Существует множество библиотек и фреймворков для создания графических интерфейсов (например, JavaFX, Swing), а также для сетевого взаимодействия и работы с базами данных;
- безопасность и управление памятью. Автоматическая сборка мусора снижает риск утечек памяти;
- хорошая производительность. Хотя Java медленнее, чем C++, но она все равно достаточно быстрая для большинства приложений.

Минусы:

- сложность. Входной порог для изучения Java может быть выше, чем для Python;
- потребление памяти. Java-приложения могут потреблять больше памяти, чем аналогичные приложения на C++ или Python;
- медленное время запуска. Приложения на Java могут запускаться медленнее из-за необходимости загрузки JVM.

C++.

Плюсы:

- высокая производительность. C++ компилируется в машинный код, что обеспечивает высокую скорость выполнения;
- низкоуровневый контроль. Прямой доступ к памяти и аппаратным ресурсам позволяет оптимизировать производительность и использовать специфические функции оборудования;
- богатый набор библиотек. Существует множество библиотек для создания графических интерфейсов (например, Qt, wxWidgets) и других задач;
- оптимизация. Возможность тонкой настройки и оптимизации кода под конкретные задачи.

Минусы:

- сложность разработки. C++ сложнее в изучении и использовании, чем Java и Python, особенно из-за управления памятью вручную;
- безопасность. Высокий риск ошибок, связанных с управлением памятью (утечки памяти, ошибки сегментации);
- меньшая кроссплатформенность. Требуется компиляция под каждую целевую платформу, что может усложнять разработку кроссплатформенных приложений.

Python.

Плюсы:

- простота и читабельность. Python имеет простой синтаксис, что делает его доступным для начинающих разработчиков;
- богатая экосистема библиотек. Существует множество библиотек для создания графических интерфейсов (например, Tkinter, PyQt, Kivy), а также для сетевого взаимодействия и работы с базами данных;
- быстрое прототипирование. Высокая скорость разработки благодаря динамической типизации и интерпретации;
- кроссплатформенность. Python-приложения могут работать на различных операционных системах с минимальными изменениями.

Минусы:

- производительность. Python медленнее, чем Java и C++, что может быть критично для приложений, требующих высокой производительности;
- управление памятью. Хотя Python имеет автоматическую сборку мусора, его интерпретатор может потреблять много памяти;
- распределение приложений. Создание исполняемых файлов и их распространение может быть сложнее по сравнению с Java и C++.

В таблице 6 представлено сравнение плюсов и минусов рассмотренных языков программирования.

Таблица 6. Сравнение языков программирования

Язык программирования	Плюсы	Минусы
Java	<ul style="list-style-type: none"> - Кроссплатформенность - Богатая экосистема - Безопасность и управление памятью - Хорошая производительность 	<ul style="list-style-type: none"> - Сложность - Потребление памяти - Медленное время запуска
C++	<ul style="list-style-type: none"> - Высокая производительность - Низкоуровневый контроль - Богатый набор библиотек - Оптимизация 	<ul style="list-style-type: none"> - Сложность разработки - Безопасность - Меньшая кроссплатформенность
Python	<ul style="list-style-type: none"> - Простота и читабельность - Богатая экосистема библиотек - Быстрое прототипирование - Кроссплатформенность 	<ul style="list-style-type: none"> - Производительность - Управление памятью - Распределение приложений

Был выбран язык Java для создания клиентского приложения за достаточную производительность и защиту от утечек памяти.

2.6 Выбор среды разработки для ПО клиента для ПК

Выбор нужной среды разработки напрямую зависит от индивидуальных предпочтений разработчика, от типа проекта и нужной функциональности. Каждая из упомянутых ниже сред имеет свои минусы и плюса и может быть

оптимальной в определенных сценариях разработки. Выбор будет происходить из трех IDE: Eclipse, IntelliJ IDEA и Netbeans.

В таблице 7 представлено сравнение плюсов и минусов рассмотренных сред разработки.

Таблица 7. Сравнение сред разработки

IDE	Плюсы	Минусы
Eclipse	<ul style="list-style-type: none">- Мощные возможности кастомизации- Широкая поддержка языков программирования- Большая и активная комьюнити- Поддержка для разработки корпоративного ПО	<ul style="list-style-type: none">- Требовательность к ресурсам- Сложность в освоении- Сложности с обновлениями
IntelliJ IDEA	<ul style="list-style-type: none">- Продвинутая поддержка рефакторинга и анализа кода- Интеллектуальное автодополнение- Поддержка множества языков и технологий- Интуитивно понятный интерфейс- Плагины	<ul style="list-style-type: none">- Платность- Требовательность к ресурсам
Netbeans	<ul style="list-style-type: none">- Бесплатность- Встроенная поддержка для Java EE и web-разработки- Хорошая поддержка GUI-разработки- Простота в использовании	<ul style="list-style-type: none">- Медлительность- Меньшая функциональность по сравнению с IntelliJ- Ограниченный каталог плагинов

Исходя из приведенной выше информации Netbeans была выбрана в качестве используемой среды разработки из-за ее бесплатного распространения и простоты в использовании.

2.7 Выводы

В данном разделе был проведён анализ аппаратных средств, выбраны языки программирования, среды разработки. Были рассмотрены средства автоматизации

– микроконтроллеры, программируемые логические контроллеры, одноплатные компьютеры и специализированные контроллеры для управления теплицами. Были выбраны микроконтроллеры, исходя из оптимальной соотношения цены и функциональности, возможности адаптации под конкретные нужды, большого количества библиотек и примеров, низкого энергопотребления, высокой надёжности в суровых условиях. Были рассмотрены микроконтроллеры различных производителей: ATmega, STM, MSP430. Исходя из технических характеристик был выбран микроконтроллер ATmega2560 и отладочная плата ARDUINO MEGA 2560 REV3. Далее рассмотрены различные модели датчиков, исполнительных устройств и устройств вывода и были выбраны подходящие модели для датчика температуры, датчика влажности, сервопривода, шагового двигателя, дисплея. Была выбрана среда разработки ArduinoIDE для ARDUINO MEGA 2560 REV3 за интуитивно понятный интерфейс, поддержку большого количества библиотек, развитое сообщество разработчиков, большое количество примеров кода. Были рассмотрены плюсы и минусы языков программирования для ПО клиента на ПК. Был выбран язык программирования Java за достаточную производительность и защиту от утечек памяти. Была выбрана среда разработки Netbeans для разработки клиентского ПО из-за бесплатного распространения и простоты в использовании.

3 Описание разработанного комплекса автоматизированной теплицы

3.1 Структура разработанного макета автоматизированной теплицы

Для построения требуемого макета следует построить структурную схему, исходя из требуемого функционала, данная схема приведена на рисунке 12.

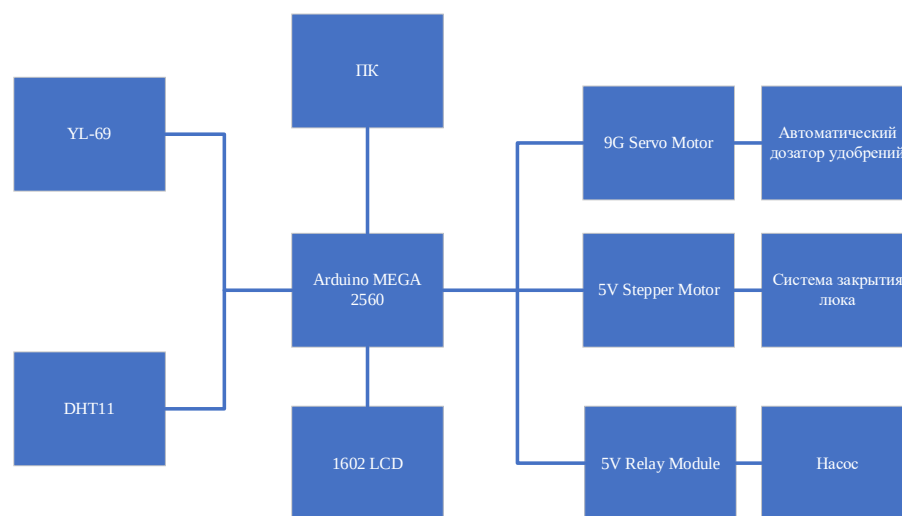


Рисунок 12 – Структурная схема макета теплицы

Ниже приведено описания компонентов данной схемы.

Компоненты структурной схемы:

- ПК (Компьютер). Связан с Arduino MEGA 2560. Используется для мониторинга и управления всей системой теплицы;
- YL-69 (Датчик влажности почвы). Измеряет уровень влажности в почве;
- DHT11 (Датчик температуры и влажности). Измеряет температуру и влажность воздуха внутри теплицы. Эти данные также поступают на Arduino MEGA 2560;
- 1602 LCD (ЖК дисплей). Подключен к Arduino MEGA 2560 для отображения данных, таких как температура, влажность, уровень освещенности и статус системы;

– 9G Servo Motor. Управляет положением механических компонентов, таких как заслонки. Обеспечивает управление автоматическим дозатором удобрений;

– 5V Stepper Motor. Обеспечивает точное движение механических компонентов, управляет системой закрытия люка для управления вентиляцией теплицы;

– 5V Relay Module. Управляет насосом для полива растений. Когда Arduino MEGA 2560 получает данные от датчика влажности почвы и определяет, что необходимо полить растения, он активирует реле, которое включает насос;

– насос. Подключен к релейному модулю. Отвечает за полив растений в теплице.

На основе вышеизложенной информации был спроектирован макет сервера программно-аппаратного комплекса автоматизированной теплицы, который представлен на рисунке 13.

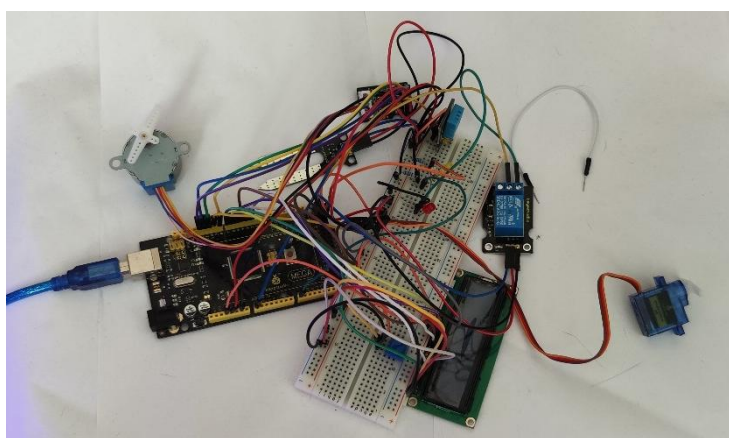


Рисунок 13 – Сервер программно-аппаратного комплекса автоматизированной теплицы

Распределение программных и аппаратных компонентов разработанного аппаратно-программного комплекса автоматизированной теплицы представлено на UML-диаграмме развёртывания на рисунке 14.

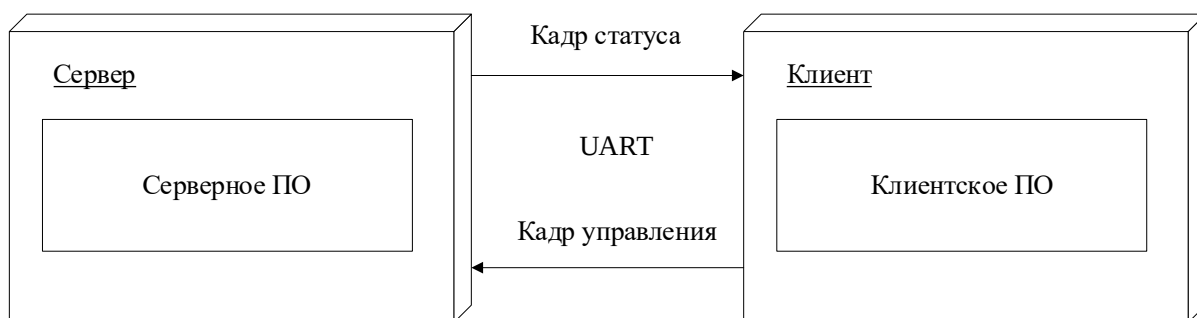


Рисунок 14 – Диаграмма развёртывания ПО аппаратно-программного комплекса автоматизированной теплицы

Сервер представляет собой Arduino MEGA 2560 и сопутствующую периферию. На сервере запускается серверное ПО. Клиент представляет собой ПК. На клиенте запускается клиентское ПО.

3.2 Описание протокола взаимодействия клиента и сервера

Сервер и клиент на рисунке 15 соединены между собой с использованием интерфейса UART.

Характеристики интерфейса UART:

- скорость интерфейса – 9600 бод/с;
- длина передаваемых данных в посылке – 8 бит;
- бит паритета – отсутствует;
- стоповый бит – 1 бит.



Рисунок 15 – Передаваемая посылка UART

Во время функционирования разработанного аппаратно-программного комплекса сервер отправляет клиенту кадры статуса в формате, представленном на рисунке 16 для обновления отображаемого статуса в клиентском ПО.

Заголовок	Температура	Влажность	Стат. исп. устройств	Контрольная сумма
1 байт	1 байт	1 байт	1 байт	1 байт

Рисунок 16 – Формат кадра статуса

Кадр имеет следующие поля:

- заголовок – поле используется для кадровой синхронизации;
- температура – значение температуры, измеренной с помощью датчика DHT11;
- влажность – значение влажности, измеренной с помощью датчика YL-69;
- стат. исп. устройств - статус исполнительных устройств (9G Servo Motor, 5V Stepper Motor, 5V Relay Module);
- контрольная сумма – контрольная сумма кадра, рассчитанная на основе алгоритма CRC8. CRC-8 использует полином $x^8 + x^5 + x^4 + 1$. В двоичном виде этот полином представляется как 10001100, что соответствует шестнадцатеричному значению 0x8C.

Для управления исполнительными устройствами теплицы клиент отправляет серверу кадры управления. Формат кадра управления представлен на рисунке 17.

Заголовок	Упр. исп. устройствами	Контрольная сумма
1 байт	1 байт	1 байт

Рисунок 17 – Формат кадра управления

Кадр имеет следующие поля:

- заголовок – поле используется для кадровой синхронизации;
- упр. исп. устройствами - устанавливаемые состояния исполнительных устройств (9G Servo Motor, 5V Stepper Motor, 5V Relay Module);

– контрольная сумма – контрольная сумма кадра, рассчитанная на основе алгоритма CRC8. CRC-8 использует полином $x^8 + x^5 + x^4 + 1$. В двоичном виде этот полином представляется как 10001100, что соответствует шестнадцатеричному значению 0x8C.

3.3 Описание разработанного ПО

3.3.1 Клиент-серверная архитектура

Клиент-серверная архитектура представляет собой модель, состоящую из двух типов узлов: клиентов и серверов. Клиенты являются программами, запущенными на устройствах пользователей, которые предоставляют интерфейс для взаимодействия с серверами. Серверы, в свою очередь, являются программами, работающими на обособленных устройствах, предоставляя при этом различные услуги клиентам, такие как веб-сервисы, базы данных.

Взаимодействие между клиентом и сервером осуществляется по определенному протоколу, который устанавливает некие правила их взаимодействия. Клиент посылает запросы на сервер, а сервер, в свою очередь, отвечает, предоставляя при этом необходимую информацию.

Данная архитектура имеет свои плюсы, такие как централизованное управление данными, находящимися на сервере, возможность совместной работы и обмена информацией между пользователями. Но в данной архитектуре существуют и минусы: необходимость поддержания и обновления серверной инфраструктуры, риск потери данных в случае сбоя сервера, а также необходимость в обеспечении безопасности данных на сервере.

3.3.2 Описание серверного ПО

Разработанное серверное ПО функционирует на основе алгоритма, представленного на рисунке 18.

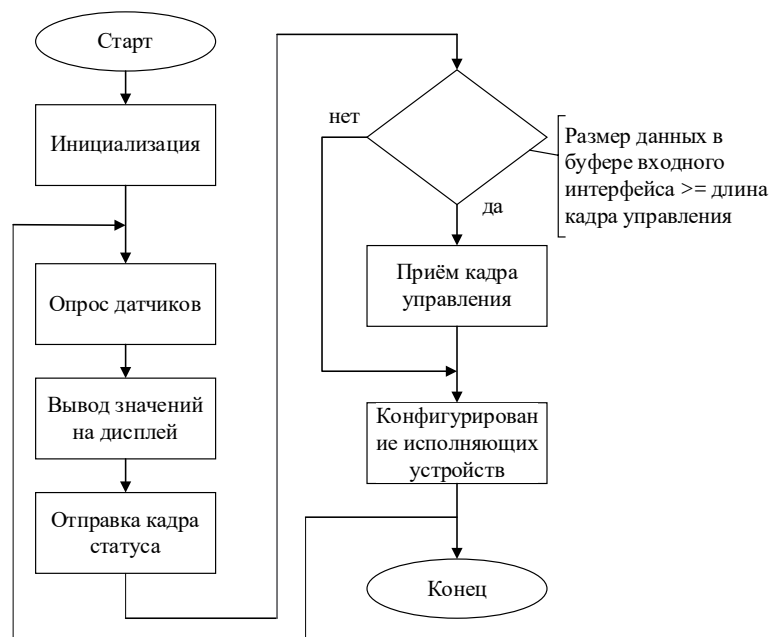


Рисунок 18 – Схема алгоритма работы серверного ПО

Вначале производится инициализация объектов используемых библиотек DHT, LiquidCrystal, Servo, Stepper и других используемых переменных. Далее производится опрос применяемых датчиков: DHT11 (температуры и влажности), YL-69 (Датчик влажности почвы) и запись значений в переменные. Далее полученные значения выводятся дисплей 1602 LCD. Далее отправляется кадр статуса с текущим состоянием датчиков и исполнительных устройств. Далее в случае наличия в буфере приёмника количества данных, равных размеру управляющего кадра, данные считываются из буфера приёмника. Если заголовок кадра равен требуемому значению и вычисленная контрольная сумма совпала с требуемой, кадр считывается и состояния исполнительных устройств устанавливаются в соответствии со значением поля «Упр. исп. устройствами» кадра управления.

3.3.3 Архитектура клиентского приложения

Иерархию и взаимодействие между классами описывает UML-диаграмма классов. Для разработанного клиентского ПО UML-диаграмма представлена на рисунке 19.

- отправляет запросы на передачу кадров потоку, созданному на основе класса SerialProcess, используя циклический опрос с интервалом 100мс с использованием таймера;

- создаёт объекты классов Menu, MenuEvents, SettingsForm, SettingsEvents, StatusControlForm, StatusControlEvents, SerialProcess.

Menu(класс):

- реализует графический функционал меню;
- вызывает функции обратного вызова (callback) объекта класса Main.

MenuEvents(интерфейс): представляет интерфейс для событий в меню.

SettingsForm(класс):

- реализует графический функционал формы для настройки интерфейса UART;

- вызывает callback-функцию объекта класса Menu и передаёт настройки интерфейса.

SettingsEvents(интерфейс): представляет интерфейс для событий формы настройки интерфейса UART.

StatusControlForm(класс):

- реализует графический функционал формы для отображения параметров теплицы: температура и влажность, а также состояний исполнительных устройств на основе данных кадра статуса, получаемого по интерфейсу UART;

- реализует графический функционал для управления исполнительными устройствами теплицы: 9G Servo Motor, 5V Stepper Motor, 5V Relay Module. С помощью формы можно выбрать будет то или иное исполнительное устройство включено или выключено. Графический функционал позволяет выбрать режим работы: автоматический или ручной. В автоматическом режиме анализируется состояние параметров и теплицы и при увеличении значений параметров заданных ограничений производится запуск или остановка соответствующего исполнительного устройства. Далее данные передаются для формирования кадра управления.

StatusControlEvents(интерфейс): представляет интерфейс для событий формы, отображающей статус теплицы и позволяющий управлять исполнительными устройствами.

SerialProcess(класс):

- реализует функционал потока, обеспечивающего взаимодействие по интерфейсу UART с Arduino MEGA 2560. При этом реализована обработка получаемого кадра статуса и формирование отправляемого кадра управления;

- реализует необходимый интерфейс для анализа состояния обработчика кадров;

- используются потокозащищённые типы данных AtomicBool и т.д. для обеспечения наибольшей защиты от доступа нескольких потоков к данным.

GardenStatus(класс):

- хранит состояние теплицы;

- предоставляет интерфейс для установки текущих параметров теплицы и состояний исполнительных устройств;

- предоставляет интерфейс для получения текущих параметров теплицы и состояний исполнительных устройств;

- используются потокозащищённые типы данных AtomicBool и т.д. для обеспечения наибольшей защиты от доступа нескольких потоков к данным.

SerialPortConfig(класс):

- хранит конфигурацию интерфейса UART;

- имеет интерфейс для установки параметров интерфейса UART;

- имеет интерфейс для считывания параметров интерфейса UART.

3.3.4 Процедура настройки интерфейса UART

В процедуре настройки интерфейса UART задействованы: пользователь, объект класса Menu, объект класса Main, объект класса SettingsForm, объект класса SerialProcess. На рисунке 20 представлена диаграмма последовательности для процедуры настройки интерфейса UART.

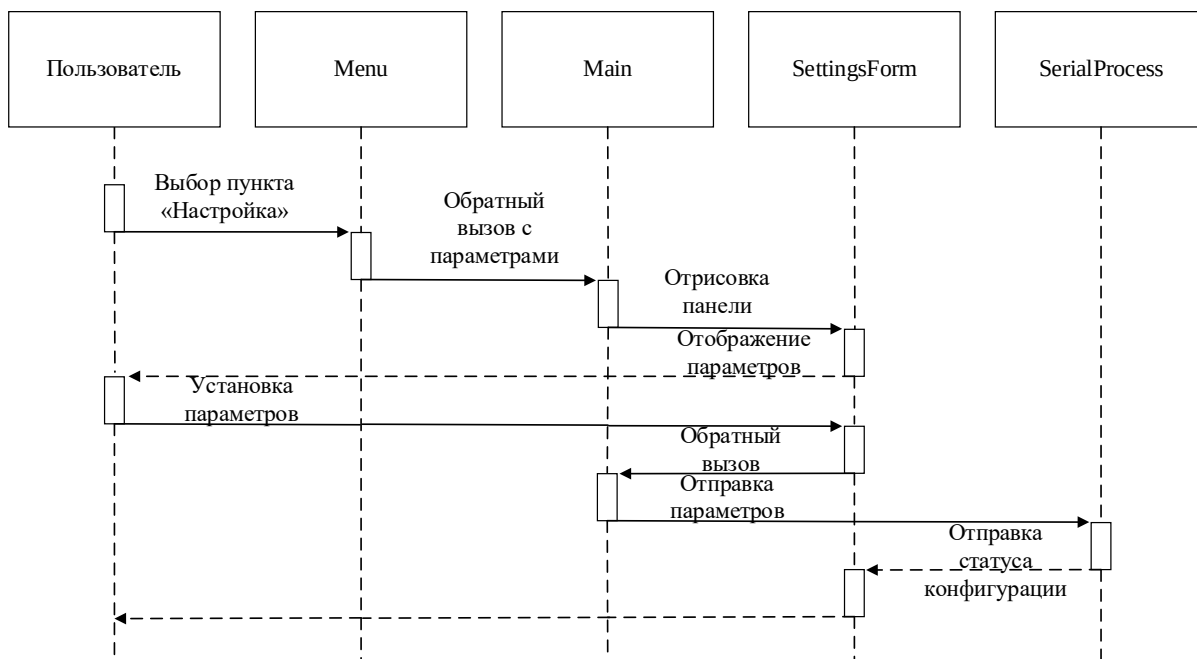


Рисунок 20 – Диаграмма последовательности процедуры настройки интерфейса UART

Вначале пользователь выбирает пункт меню настройка в левой панели главного окна, как на рисунке 21.

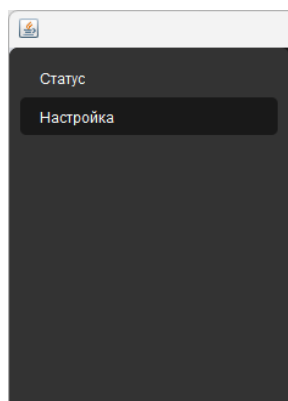


Рисунок 21 – Выбор пункта меню «Настройка»

Событие нажатия клавиши приводит к перерисовке кнопки меню с использованием метода:

```

private void setSelected(ButtonMenu menu) {
    for (Component com : panelMenu.getComponents()) {
        if (com instanceof ButtonMenu) {
            ButtonMenu b = (ButtonMenu) com;

```

```

        b.setSelected(false);
    }
}
menu.setSelected(true);
}

```

При этом производится вызов callback-метода `SelectedItem` интерфейса `MenuEvents[3]`. При этом объект интерфейса `MenuEvents` определён в классе `Main`.

```

MenuEvents event = new MenuEvents() {
    @Override
    public void SelectedMenuItem(String elementName) {
        switch (elementName)
        {
            case "Статус":
                showForm(statusControlForm);
                break;
            case "Статистика":
                //showForm(new StatusControlForm());
                break;
            case "Настройка":
                String[] UARTNames =
serialProcess.GetUARTList().toArray(new String[0]);
                settingsForm.SetUARTList(UARTNames);
                //Запрос списка интерфейсов UART
                showForm(settingsForm);
                break;
        }
    }
};

```

В методе `SelectedItem` производится отрисовка формы `SettingsForm`. Далее пользователь осуществляет установку параметров интерфейса `UART`. Пример формы настройки интерфейса `UART` представлен на рисунке 22.

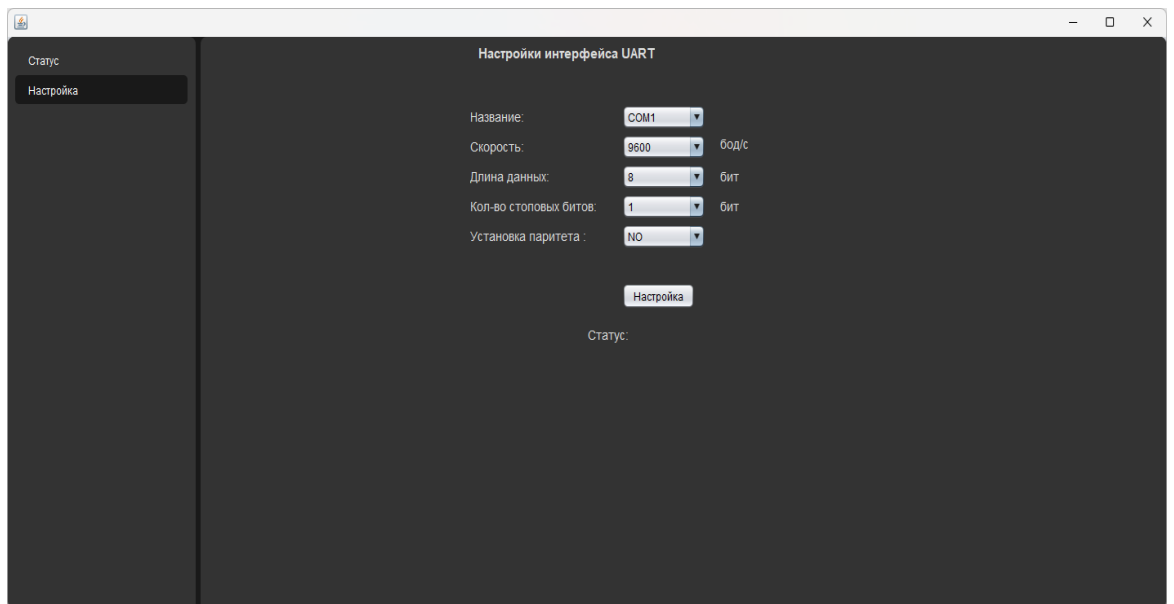


Рисунок 22 – Пример формы настройки интерфейса UART

При нажатии на клавишу «Настройка» производится считывание указанных параметров интерфейса в представленном методе:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {

    String nameUART = null;
    int baudRateValue = 0;
    int lengthDataValue = 0;
    int stopBitsValue = 0;
    int parityValue = 0;

    // Получение выбранного элемента
    nameUART = (String) nameBox.getSelectedItem();

    baudRateValue =
Integer.valueOf((String)baudRateBox.getSelectedItem());

    lengthDataValue =
Integer.valueOf((String)lengthDataBox.getSelectedItem());

    stopBitsValue =
Integer.valueOf((String)stopBitsBox.getSelectedItem());
```

```

        if ((String)parityBox.getSelectedItem() == "NO")
        {
            parityValue = 0;
        }
        else
        {
            parityValue = 1;
        }

        SerialPortConfig serialPortConfig = new SerialPortConfig();
        serialPortConfig.SetSerialPortConfig(nameUART, baudRateValue,
lengthDataValue, stopBitsValue, parityValue);

        event.SettingsUART(serialPortConfig);

    }

```

При этом параметры интерфейса UART передаются в callback-метод SettingsUART класса Main:

```

        SettingsEvents settingsEvents = new SettingsEvents() {
            @Override
            public void SettingsUART(SerialPortConfig
serialPortConfig)
            {
                serialProcess.UpdateSettingsUART(serialPortConfig);
            }
        };

```

В методе SettingsUART вызывается метод UpdateSettingsUART в котором устанавливаются устанавливаемые настройки интерфейса UART, устанавливается статус смены настройки интерфейса UART:

```

        public void UpdateSettingsUART(SerialPortConfig
serialPortConfigParam)
        {
            serialPortConfig = serialPortConfigParam;
            settingsChanged.set(true);
        }

```

Если UART-порт не открыт, то происходит открытие порта с использованием метода OpenUART класса SerialProcess:

```
public void OpenUART ()
{
    SerialPort[] ports = SerialPort.getCommPorts();
    try {
        for (SerialPort port : ports) {
            if
(port.getSystemPortName().equals(serialPortConfig.GetNameUART())) {
                sp = port;
                break;
            }
        }

        if (sp == null) {
            System.out.println("Порт не найден.");
            return;
        }

        // Открытие порта
        sp.setBaudRate(serialPortConfig.GetbaudRateValue());
        sp.setNumDataBits(serialPortConfig.GetlengthDataValue());
        sp.setNumStopBits(serialPortConfig.GetstopBitsValue());
        sp.setParity(serialPortConfig.GetparityValue());

        if (!sp.openPort()) {
            System.out.println("Не удалось открыть порт.");
            return;
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Если UART-порт уже был открыт, то вначале открытый на данный момент порт закрывается и открывается новый. В дальнейшем открытый порт будет применяться для приёма кадров статуса и отправки кадров управления.

3.3.5 Процедура отображения статуса теплицы

В процедуре отображения статуса теплицы участвуют пользователь, клиент, сервер. Клиент представляет собой ПК с клиентским ПО. Сервер представляет собой Arduino MEGA 2560 с датчиками, исполнительными устройствами и дисплеем для вывода.

Графическое описание процедуры управления теплицей представлено диаграммой деятельности на рисунке 23.

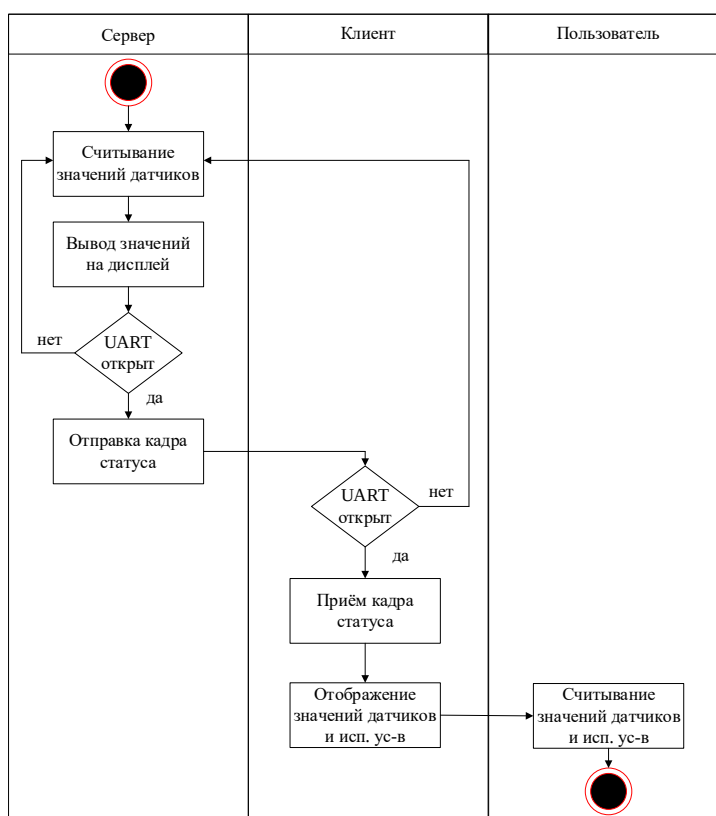


Рисунок 23 – Диаграмма деятельности процедуры отображения статуса теплицы

Перед выполнением процедуры отображения статуса теплицы необходимо успешно пройти процедуры настройки интерфейса UART на стороне сервера и клиента.

Вначале процедуры отображения статуса теплицы сервер считывает значения с датчиков, далее выводит значения на дисплей.

Далее производится формирование кадра статуса в формате, представленном на рисунке 16 при этом статус исполнительных устройств хранится в переменной.

После этого производится передача сформированного кадра статуса по интерфейсу UART. Сторона клиента принимает кадр, проверяет его целостность с использованием алгоритма CRC-8, применяя методы класса SerialProcess[4].

Фрагмент функции, осуществляющий обработку принимаемых кадров статуса по интерфейсу UART:

```
public void run()
{

    if (sp != null)
    {
        if (!statusReady.get())
        {
            //Ожидание кадра на входном интерфейсе
            if (crc_error)
            {
                sp.readBytes(buffer_rx, 1);
                crc_error = false;
            }
            //Производим поиск заголовка
            if (sp.bytesAvailable() >= 1)
            {
                sp.readBytes(buffer_rx, 1);
                if (buffer_rx[0] ==
global_variables.HEAD_STATUS)
                {
                    if (sp.bytesAvailable() >=
(global_variables.LENGTH_FRAME_STATUS - 1))
                    {
```

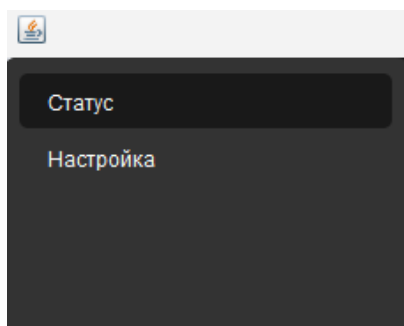



Рисунок 24 – Выбор пункта меню «Статус»

В результате Main отображается StatusControlForm. Область «Текущий статус» в StatusControlForm представлена на рисунке 25.

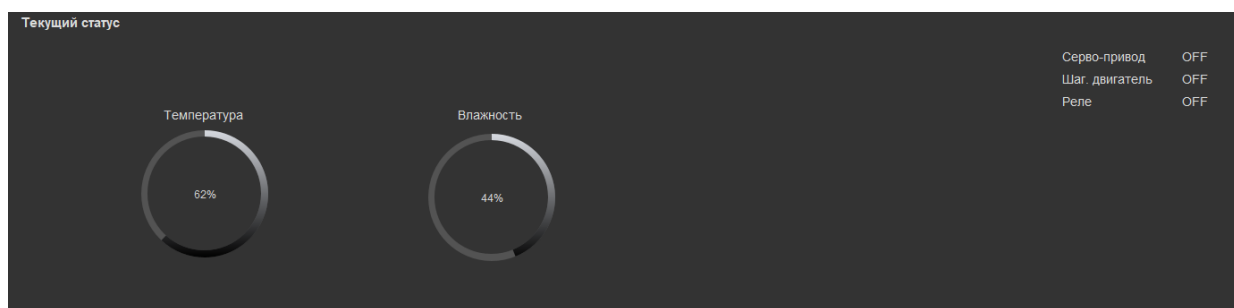


Рисунок 25 – Область «Текущий статус» в StatusControlForm

Используя область «Текущий статус», пользователь наблюдает за состоянием датчиков и исполнительных устройств.

3.3.6 Процедура управления теплицей

В процедуре управления теплицей участвуют пользователь, клиент, сервер. Клиент представляет собой ПК с клиентским ПО. Сервер представляет собой Arduino MEGA 2560 с датчиками, исполнительными устройствами и дисплеем для вывода[7].

Графическое описание процедуры управления теплицей представлено диаграммой деятельности на рисунке 26.

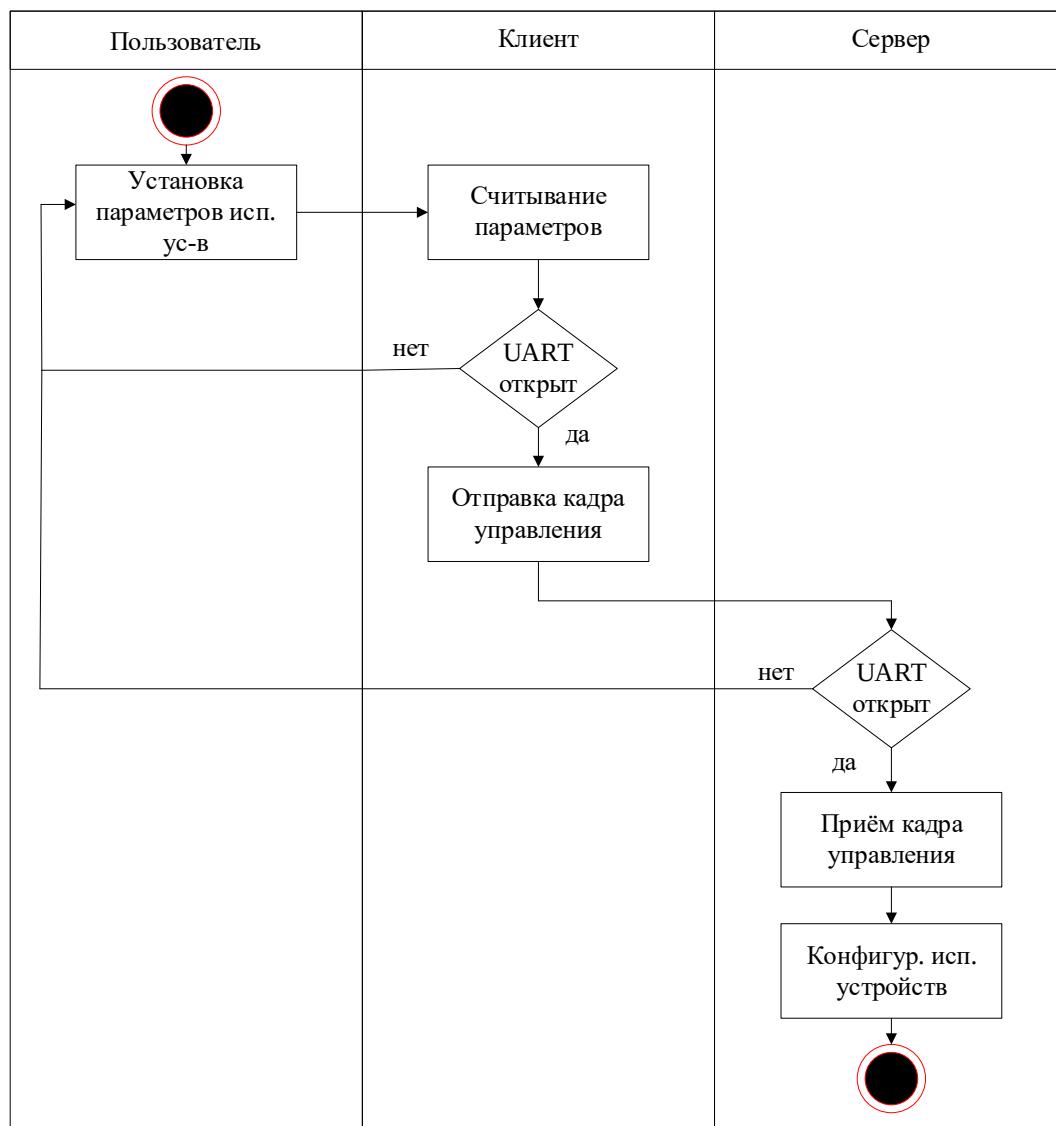


Рисунок 26 – Диаграмма деятельности процедуры управления теплицей

Далее пользователь устанавливает режим управления теплицей: ручной или автоматический, состояние исполнительных устройств (вкл/выкл) с использованием области «Управление теплицей» в StatusControlForm, представленной на рисунке 27.

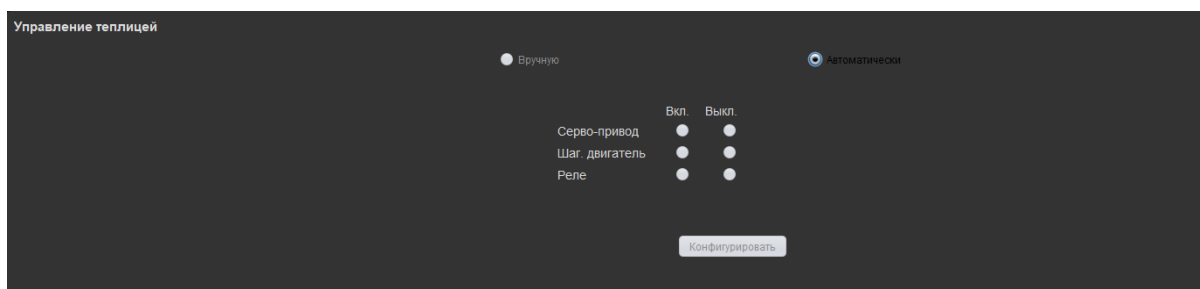


Рисунок 27 – Область «Управление теплицей» в StatusControlForm

Функционал для данной формы реализует класс StatusControlForm.

При выборе автоматического режима клиентское ПО анализирует значения параметров теплицы (температура, влажность) и делает выбор об изменении состояний исполнительных устройств с использованием метода SetAutomaticControl класса GardenStatus.

```
public void SetAutomaticControl ()
{
    if (humidityPer >= global_variables.TRESHHOLD_HUMIDITY_PER)
    {
        actuators |= global_variables.MASK_SERVO;
    }

    if (temperaturePer >=global_variables.TRESHHOLD_TEMPERATURE_PER)
    {
        actuators |= global_variables.MASK_STEPPER;
    }
}
```

При выборе ручного режима клиентское ПО возвращает выбранные состояния исполнительных устройств с помощью метода SendControl интерфейса StatusControlEvents с классом StatusControlForm. Методы интерфейса с StatusControlForm:

```
StatusConrtolEvents statusControlEvents = new
StatusConrtolEvents() {
    @Override
    public void SelectedControlMode(String elementName) {
        switch (elementName)
        {
            case "Автоматически":
                gardenStatus.SetControlMode(global_variables.AUTOMATIC_CONTROL);
                break;
            case "Вручную":
```

```

gardenStatus.SetControlMode(global_variables.MANUAL_CONTROL);
        break;
    }
}

public void SendControl(byte actuators_param)
{
    controlEn.set(true);
    actuatorsTX = actuators_param;
}
};

```

Фрагмент функции, осуществляющий формирования кадров управления, передаваемых по интерфейсу UART:

```

public void run()
{
    // Отправка кадра с выходного интерфейса
    if (controlReady.get() == false)
    {
        byte[] buffer = new
byte[global_variables.LENGTH_FRAME_CONTROL];

        buffer[0] = global_variables.HEAD_CONTROL;
        buffer[1] = actuators_value;

        byte calculatedCrc = (byte)CRC8(buffer, 2);
        buffer[2] = calculatedCrc;

        controlReady.set(true);
    }
}

}
}
catch (Exception e)
{
    System.out.println("UART exception!");
}
}

```

3.4 Выводы

В разделе представлена структура разработанного макета с указанием функционального назначения компонентов, распределение программных и аппаратных компонентов разработанного аппаратно-программного комплекса автоматизированной теплицы. Описан разработанный протокол взаимодействия клиента и сервера в составе аппаратно-программного комплекса автоматизированной теплицы с указанием настроек интерфейса взаимодействия UART, формата кадров статуса и управления, использующихся для взаимодействия клиента и сервера. Описан алгоритм работы серверного ПО. Представлена диаграмма классов клиентского приложения, описаны функциональные назначения классов. Описана процедура настройки интерфейса UART с приведением диаграммы последовательности процедуры настройки. Описана процедура отображения статуса теплицы и управления теплицей с приведением диаграмм последовательностей.

4 Безопасность жизнедеятельности

Огромную роль для разработчиков, которые длительное время проводят за монитором играют правила безопасности при работе за компьютером. Нарушение данных правил может привести к различным проблемам со здоровьем, таким как нарушения осанки, перенапряжение глаз, развитие туннельного синдрома.

В данном разделе мы рассмотрим основные принципы безопасности при работе за компьютером.

4.1 Требования к помещению и рабочему месту

При организации рабочего места для работы с ПК следует учитывать определенные требования, чтобы обеспечить комфорт, безопасность и продуктивность.

Так, например, на одного работающего должно приходиться не менее 6 квадратных метра площади и 20 кубических метров помещения. Оптимальная температура должна быть в пределах 22-24 градусов Цельсия при относительной влажности 40-60%. Должна быть обеспечена достаточная вентиляция и кондиционирование для поддержания комфортного микроклимата. Уровень шума не должен превышать 50 дБА.

Что же касается конкретно рабочего места, то размеры стола должны обеспечивать удобное размещение оборудования, документов и персональных вещей. Стул должен быть регулируемым по высоте, с поддержкой поясничного отдела позвоночника. Экран монитора должен находиться на расстоянии 50-70 см от глаз пользователя, верхняя граница экрана должна находиться на уровне глаз или чуть ниже. При необходимости должна быть подставка для ног, обеспечивающая устойчивое положение ног под углом 90 градусов.

4.2 Требования к освещению

Требования к освещению при работе за компьютером включают несколько аспектов, которые обеспечивают создание комфортных и безопасных условий для зрения и общего самочувствия пользователя.

Так, например, рабочее место должно быть размещено таким образом, чтобы естественный свет падал сбоку, предпочтительно слева-для правой и справа-для левой. Для предотвращения бликов и прямого попадания солнечных лучей на монитор следует использовать жалюзи, шторы или специальные пленки на окна.

На рабочем месте устанавливаются дополнительные источники света(настольные лампы) с регулируемой яркостью и направлением света. Рекомендуется использование ламп с цветовой температурой 3000-4000 К для создания комфортного рабочего пространства. Уровень освещенности на рабочей поверхности стола должен быть в пределах 500-1000 люкс. Настольная лампа должна располагаться таким образом, чтобы свет падал на рабочую зону без образования теней и бликов на экране монитора.

4.3 Требования к режиму труда и отдыха

Требования к режиму труда и отдыха при работе с компьютером направлены на снижение утомляемости и сохранения здоровья работников. Эти требования включают организацию рабочего времени, перерывов, а также гимнастики для глаз и тела.

Что касается продолжительности непрерывной работы для взрослых работников, то она должна составлять не более 2 часов, для школьников не более 40 минут. В течение рабочего дня необходимо делать короткие перерывы через каждые 45-60 минут работы за компьютером. Длительность перерыва должна быть не менее 5-10 минут. За 8-часовой рабочий день рекомендуется совокупная продолжительность перерывов не менее 60 минут.

Каждые 20 минут работы за компьютером рекомендуется выполнять упражнения для глаз, чтобы уменьшить зрительное напряжение. Примеры упражнений:

- фокусировка взгляда на дальних и ближних объектах;
- медленные круговые движения глазами;
- частое и мягкое моргание.

Каждые 1-2 часа работы рекомендуется выполнять короткие физические упражнения для разминки мышц и суставов.

4.4 Рекомендации по использованию программного обеспечения

Для программистов, которые хотят следить за безопасным использованием компьютера, существуют специализированные программы и инструменты. Эти инструменты помогают контролировать аспекты безопасности, продуктивности и здоровья при работе за компьютером.

Рекомендуется использовать следующее программное обеспечения:

- Workrave. Программа, напоминающая о регулярных перерывах и предлагающая упражнения для глаз и тела;
- Eyeleo. Приложение для защиты глаз, которое напоминает о перерывах и предлагает упражнения для глаз;
- Clockify. Бесплатный инструмент для трекинга времени, который помогает следить за продуктивностью и управлять задачами;
- Toggl. Трекер времени, который позволяет отслеживать время, затраченное на различные задачи и проекты.

Таким образом, использование данного ПО позволит обеспечить комфортные условия для работы за ПК.

Заключение

Использование автоматизированных теплиц способствует повысить эффективность аграрного средства и снизить уровень негативного воздействия на окружающую среду. Основные аспекты, подчёркивающие актуальность темы автоматизированных теплиц: рациональное использование ресурсов, технологический прогресс, экономическая эффективность. Рассмотрены распространённые готовые решения “умных” теплиц и произведено их сравнение с собственными разработками. Поскольку важны индивидуализация разработки под различные условия применения, учитывая возможные суровые условия теплиц, было принято решение о создании собственной «умной» теплицы.

При разработке собственной теплицы был проведён анализ аппаратных средств, выбраны языки программирования, среды разработки. В качестве средств автоматизации были выбран микроконтроллер ATmega2560 и отладочная плата ARDUINO MEGA 2560 REV3, исходя из оптимального соотношения цены и функциональности, возможности адаптации под конкретные нужды, большого количества библиотек и примеров, низкого энергопотребления, высокой надёжности в суровых условиях. Был произведён выбор датчиков, исполнительных устройств, ориентируясь на основные показатели и минимальный возможный функционал для исполнительных устройств. Для разработки клиентского ПО был выбран язык Java и среду разработки NetBeans.

На основе произведённых исследований была разработана структурная схема программно-аппаратного комплекса автоматизированной теплицы, произведена сборка аппаратной составляющей комплекса. Комплекс состоит из сервера (Arduino MEGA 2560 и сопутствующая периферия) и клиента (ПК). Был разработан протокол, регламентирующий взаимодействие между клиентом и сервером. Были описаны архитектуры и алгоритмы работы сервера и клиента с использованием диаграмм UML.

В дальнейшем планируется увеличить количество проверяемых показателей и добавить датчики, например, датчик освещённости, инфракрасный датчик и т.д. Также планируется добавить возможность управления теплицей с помощью

пульта дистанционного управления. Также планируется реализовать веб-версию клиентского приложения для увеличения количества пользователей программно-аппаратным комплексом. Также планируется проведение постоянного тестирования работы теплицы для обеспечения обнаружения и исправления возможных ошибок в случае высокой информационной нагрузки на автоматизированную теплицу.

Список используемых источников

1. Дж.Блум. Изучаем Arduino: инструменты и методы технического волшебства. -2-е изд.БХВ,2020. -185-215с.
2. А.Биняковский,В.Петин.Практическая энциклопедия Arduino. -2-е изд.ДМК Пресс,2019. -76-92с.
3. М.Лой,Р.Экстейн.Java Swing. -2-е изд.О'Рейли Медиа,2002. -225-259с.
4. Г.Шилдт.Java.Полное руководство. -12-е изд.Диалектика-Вильямс,2022. -1189-1240с.
5. М.Марголис.Arduino.Большая книга рецептов. -3-е изд.ВНВ-СПБ,2021. -51-113с.
6. В.Петин.Проекты с использованием контроллера Arduino. -4-е изд.ВНВ-СПБ,2021. -142-150с.
7. Клиент-серверная архитектура.[Электронный ресурс]/Режим доступа: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit,свободный.(дата обращения 27.04.2024).

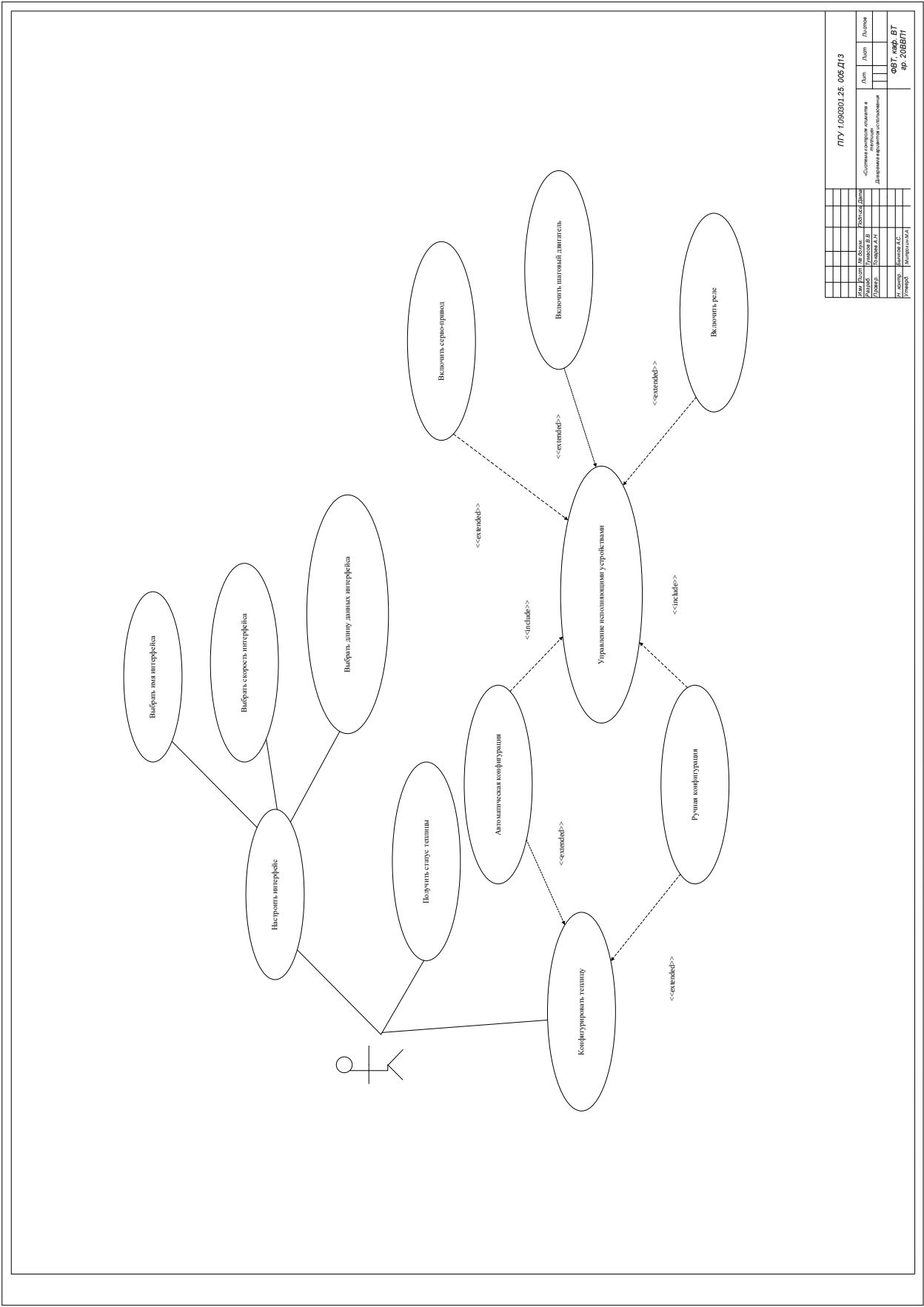
Приложение А

(Обязательное)

Система контроля климата в теплице

UML-диаграммы

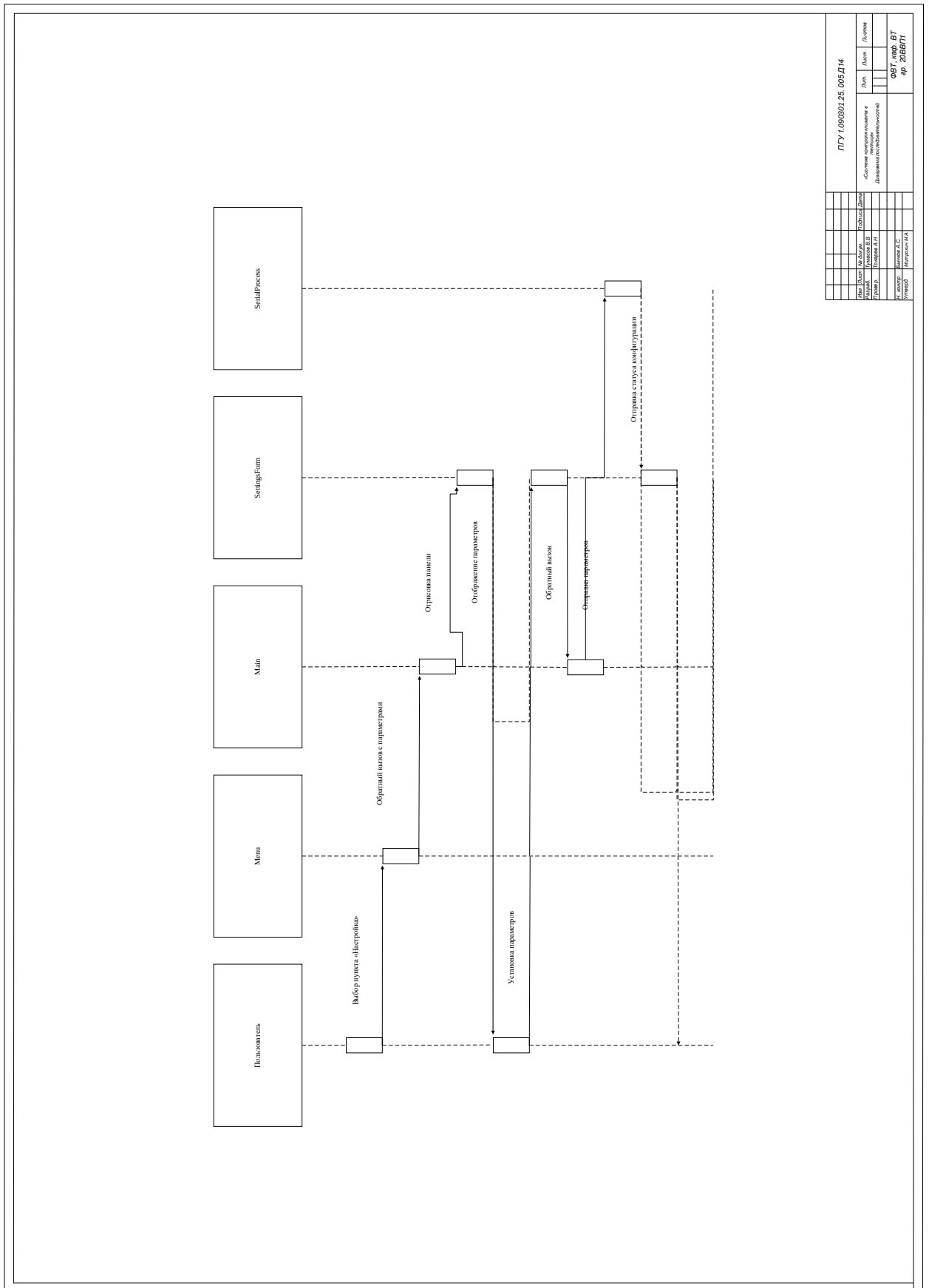
Приложение А.1 - Диаграмма вариантов использования программного-аппаратного комплекса автоматизированной теплицы



ПУ 1090301.25 004 Д13									
Имя	Роль	№ докум.	Подпись	Дата	Содержимое раздела в документе				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				
Иванов	Инженер	1090301.25	Иванов	2024.11.15	Диаграмма вариантов использования				

[illegible]

настройки интерфейса UART



```

    graph TD
      subgraph Server
        Start(( )) --> Read[Считывание значений датчиков]
        Read --> Output[Вывод значений на дисплей]
        Output --> UART_Open{UART открыт}
        UART_Open -- да --> Send[Отправка кадра статуса]
        UART_Open -- нет --> Read
      end

      subgraph Client
        UART_Open -- нет --> Read
        UART_Recv{UART открыт} -- да --> Recv[Приём кадра статуса]
        Recv --> Display[Отображение значений датчиков и исп. ус-в]
      end

      subgraph User
        Display --> ReadUser[Считывание значений датчиков и исп. ус-в]
        ReadUser --> End(( ))
      end

      Send --> UART_Recv
      UART_Recv -- нет --> Read
  
```

Приложение А.5 - Диаграмма деятельности процедуры управления теплицей

Система контроля климата в теплице

Листинг программы

Файл Auto_garden_project_v3.ino

```
#include <dht.h>
#include <LiquidCrystal.h>
#include <Servo.h>
#include <Stepper.h>

// Датчик влажности воздуха и температуры
dht DHT;

#define DHT11_PIN 22
#define SERVO1_ON 0x01
#define SERVO1_OFF 0xFE
#define SERVO2_ON 0x02
#define SERVO1_OFF 0xFD

//Фоторезистор
int potpin=8;// initialize analog pin 0, connected with photovaristor
int ledpin_pot=28;// initialize digital pin 11

//Micro Servo 9g
Servo myservo;// define servo variable name

//Stepper
#define STEPS 100
Stepper stepper(STEPS, 8, 10, 11, 12);

//Реле
int Relay = 26;

// Дисплей 1602LCD
int DI = 12;
int RW = 11;
int DB[] = {3, 4, 5, 6, 7, 8, 9, 10}; // use array to select pin for
bus
int Enable = 2;

// Инициализация интерфейса 1602LCD
const int rs = 7, en = 2, d4 = 3, d5 = 4, d6 = 5, d7 = 6;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

```

//Инициализация используемых переменных
char humidity_str[17];
char temp_str[17];

byte crc8(const byte *data, int len) {
    byte crc = 0x00; // Инициализация CRC с 0
    for (int i = 0; i < len; i++) {
        byte extract = data[i]; // Извлечение байта данных
        for (int tempI = 8; tempI > 0; --tempI) { // Перебор каждого
бита байта данных
            byte sum = (byte) ((crc ^ extract) & 0x01); // Вычисление
младшего бита суммы
            crc >>= 1; // Сдвиг CRC вправо
            if (sum != 0) { // Если сумма не равна 0, то выполняем
XOR с полиномом 0x8C
                crc ^= 0x8C;
            }
            extract >>= 1; // Сдвиг извлеченного байта вправо
        }
    }
    return crc; // Возвращение конечного значения CRC
}

void setup() {

    //Настройка UART для связи с ПК
    Serial.begin(9600);
    //Вывод сообщения о датчике влажности воздуха и температуры
    //
    Serial.println("Type,\tstatus,\tTemperature(Air)
(°C),\tHumidity(Air) (%),\tHumidity(Ground) (%)");

    //Настройка LED для индикации фоторезистора
    pinMode(ledpin_pot, OUTPUT);

    //Настройка Micro Servo 9g
    myservo.attach(9); // select servo pin

    //Настройка Stepper
    stepper.setSpeed(90);

```

```

//Настройка реле
pinMode(27, OUTPUT);          //Set Pin13 as output
digitalWrite(27, HIGH);       //Set Pin13 High
pinMode(Relay, OUTPUT);       //Set Pin8 as output

// Количество строк и столбцов на экране
lcd.begin(16, 2);

}

void loop() {
    // Обработка данных датчика влажности воздуха и температуры
    int chk_dht11;
    byte dht_temperature;
    byte dht_humidity;
    byte humidity;
    byte illumination;
    int previous_stepper_val = 0;
    int stepper_val = 0;
    int status_frame_size = 5;
    int control_frame_size = 3;
    byte header_status_frame = 0x4b;
    byte header_conrol_frame = 0x4a;
    byte actuators_status = 0x00;
    byte crc8_val = 0x00;
    byte MASK_SERVO1 = 0x01;
    byte MASK_SERVO2 = 0x02;
    byte MASK_RELAY = 0x04;

    // Кадр статуса на ПК
    byte status_frame[status_frame_size];

    // Кадр управления от ПК
    byte control_frame[control_frame_size];

    // Фаза опроса датчиков
    // READ DATA
    // Serial.print("DHT11, \t");

```

```

chk_dht11 = DHT.read11(DHT11_PIN);
switch (chk_dht11)
{
    case DHTLIB_OK:
//    Serial.print("OK,\t");
        break;
    case DHTLIB_ERROR_CHECKSUM:
//    Serial.print("Checksum error,\t");
        break;
    case DHTLIB_ERROR_TIMEOUT:
//    Serial.print("Time out error,\t");
        break;
    // case DHTLIB_ERROR_CONNECT:
    //    Serial.print("Connect error,\t");
    //    break;
    case DHTLIB_ERROR_ACK_L:
//    Serial.print("Ack Low error,\t");
        break;
    case DHTLIB_ERROR_ACK_H:
//    Serial.print("Ack High error,\t");
        break;
    default:
//    Serial.print("Unknown error,\t");
        break;
}

dht_temperature = DHT.temperature;
humidity = analogRead(0);
illumination=analogRead(potpin);
analogWrite(ledpin_pot,illumination/9);

//Вывод результата обработки значений датчика влажности воздуха и
температуры в UART
// Установка курсора на первую строку
//Вывод значения температуры
lcd.setCursor(0, 0);
lcd.print("Temp: " + String(dht_temperature) + "C");
// Установка курсора на вторую строку
lcd.setCursor(0, 1);

```

```

//Вывод значения влажности
//lcd.print("                ");
sprintf(humidity_str,"Humidity: %03d",humidity);
lcd.print(humidity_str + String("%"));

// Фаза отправки кадра статуса на ПК
status_frame[0] = header_status_frame;
status_frame[1] = dht_temperature;
status_frame[2] = humidity;
status_frame[3] = actuators_status;

// Расчет CRC8
crc8_val = CRC8(status_frame, 4);
status_frame[4] = crc8_val;

// Отправка кадра
Serial.write(status_frame, 5);

//Фаза приёма кадра управления
if (Serial.available() >= control_frame_size)
{
    Serial.readBytes(control_frame, control_frame_size);
    crc8_val = CRC8(control_frame, 2);
    if (crc8_val == control_frame[2])
    {
        actuators_status = control_frame[1];
    }
}

//Фаза управления исполнительными устройствами
//Управление Micro Servo 9g
if ((actuators_status & (~ MASK_SERVO1)) != 0)
{
    myservo.write(1);// set rotate angle of the motor
}

//Управление Stepper
if ((actuators_status & (~ MASK_SERVO2)) != 0)
{

```

```

        stepper_val = analogRead(0);
        stepper.step(stepper_val - previous_stepper_val);
        previous_stepper_val = stepper_val;
    }

    if ((actuators_status & (~ MASK_SERVO2)) != 0)
    {
        //Управление Реле
        digitalWrite(Relay, HIGH);    //Turn off relay
        delay(2000);
        digitalWrite(Relay, LOW);     //Turn on relay
    }

    // DISPLAY DATA
    //Вывод результата обработки значений датчика влажности воздуха и
температуры, датчика влажности почвы в UART
    // Serial.println(DHT.temperature,1);
    // Serial.print(",\t");
    // Serial.print(DHT.humidity,1);
    // Serial.print(",\t");
    // Serial.print(analogRead(0),1);

    // Задержка на 500 миллисекунд
    delay(500);
}

```

Файл ButtonMenu.java

```

package Components_GUI;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Cursor;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.Area;
import java.awt.geom.Ellipse2D;

```

```

import java.awt.geom.RoundRectangle2D;
import javax.swing.JButton;
import javax.swing.border.EmptyBorder;

public class ButtonMenu extends JButton {

    public Color getEffectColor() {
        return effectColor;
    }

    public void setEffectColor(Color effectColor) {
        this.effectColor = effectColor;
    }

    //    private Animator animator;
    private int targetSize;
    private float animatSize;
    private Point pressedPoint;
    private float alpha;
    private Color effectColor = new Color(173, 173, 173);

    public ButtonMenu() {
        setContentAreaFilled(false);
        setBorder(new EmptyBorder(8, 10, 8, 10));
        setHorizontalAlignment(JButton.LEFT);
        setBackground(new Color(43, 44, 75));
        setForeground(new Color(250, 250, 250));
        setCursor(new Cursor(Cursor.HAND_CURSOR));
        addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent me) {
                targetSize = Math.max(getWidth(), getHeight()) * 2;
                animatSize = 0;
                pressedPoint = me.getPoint();
                alpha = 0.5f;
            }
        });
    }
}

```

```

@Override
protected void paintComponent(Graphics grphcs) {
    int width = getWidth();
    int height = getHeight();
    Graphics2D g2 = (Graphics2D) grphcs;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setColor(getBackground());
    if (pressedPoint != null) {
        Area area = new Area(new RoundRectangle2D.Double(0, 0,
width, height, 10, 10));
        g2.setColor(effectColor);

g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_ATOP,
alpha));

        area.intersect(new Area(new
Ellipse2D.Double((pressedPoint.x - animatSize / 2), (pressedPoint.y -
animatSize / 2), animatSize, animatSize)));
        g2.fill(area);
    }
    g2.setComposite(AlphaComposite.SrcOver);
    super.paintComponent(grphcs);
}

@Override
public void paint(Graphics grphcs) {
    if (isSelected()) {
        int width = getWidth();
        int height = getHeight();
        Graphics2D g2 = (Graphics2D) grphcs.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(new Color(25, 25, 25));
        g2.fillRoundRect(0, 0, width - 1, height - 1, 10, 10);
    }
    super.paint(grphcs);
}
}

```

Файл Menu.java

```
package Components_GUI;

import java.awt.Color;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import net.miginfocom.swing.MigLayout;

public class Menu extends javax.swing.JPanel {

    private MenuEvents menuEvents;

    public Menu() {
        initComponents();
        setOpaque(false);
        ScrollBarCustom sb = new ScrollBarCustom();
        sb.setForeground(new Color(130, 130, 130, 100));
        jScrollPane1.setVerticalScrollBar(sb);
        panelMenu.setLayout(new MigLayout("wrap, fillx, inset 3",
"[fill]", "[[]0[])"));
    }

    public void initMenu(MenuEvents menuEvents) {
        this.menuEvents = menuEvents;
        addMenu("Статус");
        addMenu("Статистика");
        addMenu("Настройка");
    }

    private void addEmpty() {
        panelMenu.add(new JLabel(), "push");
    }
}
```

```

private void addMenu(String text) {
    ButtonMenu menu = new ButtonMenu();
    menu.setText(" " + text);
    panelMenu.add(menu);
    menu.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            menuEvents.SelectedMenuItem(text);
            setSelected(menu);
        }
    });
}

private void setSelected(ButtonMenu menu) {
    for (Component com : panelMenu.getComponents()) {
        if (com instanceof ButtonMenu) {
            ButtonMenu b = (ButtonMenu) com;
            b.setSelected(false);
        }
    }
    menu.setSelected(true);
}

// Variables declaration - do not modify
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JPanel panelMenu;
private Components_GUI.RoundPanel roundPanel1;
// End of variables declaration
}

```

Файл MenuEvents.java

```

package Components_GUI;

public interface MenuEvents {

    public void SelectedMenuItem(String elementName);
}

```

Файл RoundPanel.java

```

package Components_GUI;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JPanel;

public class RoundPanel extends JPanel {

    public RoundPanel() {
        setOpaque(false);
    }

    @Override
    public void paint(Graphics grphcs) {
        Graphics2D g2 = (Graphics2D) grphcs.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(getBackground());
        g2.fillRoundRect(0, 0, getWidth(), getHeight(), 15, 15);
        g2.dispose();
        super.paint(grphcs);
    }
}

```

Файл SettingsEvents.java

```

package Components_GUI;

import java.util.*;

import SerialProcess.SerialPortConfig;

public interface SettingsEvents {

    public void SettingsUART(SerialPortConfig serialPortConfig);
}

```

Файл SettingsForm.java

```

package Components_GUI;

```

```

import javax.swing.*;
import java.awt.*;
import java.util.List;
import java.awt.Color;
import Main.global_variables;
import SerialProcess.SerialPortConfig;
import com.fazecast.jSerialComm.SerialPort;
import java.util.concurrent.atomic.AtomicBoolean;

public class SettingsForm extends javax.swing.JPanel {

    private SettingsEvents event;

    public SettingsForm(SettingsEvents event) {
        initComponents();
        setOpaque(false);
        this.event = event;

        String[] baudRateNames = {"9600", "115200"};
        String[] lengthDataNames = {"8", "7"};
        String[] stopBitsNames = {"1", "2"};
        String[] parityNames = {"NO", "ODD"};

        DefaultComboBoxModel<String>      baudRateModel      =      new
DefaultComboBoxModel<>(baudRateNames);
        baudRateBox.setModel(baudRateModel);
        DefaultComboBoxModel<String>      lengthDataModel    =      new
DefaultComboBoxModel<>(lengthDataNames);
        lengthDataBox.setModel(lengthDataModel);
        DefaultComboBoxModel<String>      stopBitsModel      =      new
DefaultComboBoxModel<>(stopBitsNames);
        stopBitsBox.setModel(stopBitsModel);
        DefaultComboBoxModel<String>      parityModel        =      new
DefaultComboBoxModel<>(parityNames);
        parityBox.setModel(parityModel);
    }

```

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {

    String nameUART = null;
    int baudRateValue = 0;
    int lengthDataValue = 0;
    int stopBitsValue = 0;
    int parityValue = 0;

    // Получение выбранного элемента
    nameUART = (String) nameBox.getSelectedItem();

    baudRateValue =
Integer.valueOf((String)baudRateBox.getSelectedItem());

    lengthDataValue =
Integer.valueOf((String)lengthDataBox.getSelectedItem());

    stopBitsValue =
Integer.valueOf((String)stopBitsBox.getSelectedItem());

    if ((String)parityBox.getSelectedItem() == "NO")
    {
        parityValue = 0;
    }
    else
    {
        parityValue = 1;
    }

    SerialPortConfig serialPortConfig = new SerialPortConfig();
    serialPortConfig.SetSerialPortConfig(nameUART, baudRateValue,
lengthDataValue, stopBitsValue, parityValue);

    event.SettingsUART(serialPortConfig);

}

```

```

        public void SetUARTList(String[] UARTNames_param) {
            DefaultComboBoxModel<String> nameModel = new
DefaultComboBoxModel<>(UARTNames_param);
            nameBox.setModel(nameModel);
        }

        // Variables declaration - do not modify
        private javax.swing.JComboBox<String> baudRateBox;
        private javax.swing.JButton jButton1;
        private javax.swing.JLabel jLabel12;
        private javax.swing.JLabel jLabel13;
        private javax.swing.JLabel jLabel14;
        private javax.swing.JLabel jLabel15;
        private javax.swing.JLabel jLabel16;
        private javax.swing.JLabel jLabel17;
        private javax.swing.JLabel jLabel18;
        private javax.swing.JLabel jLabel19;
        private javax.swing.JLabel jLabel3;
        private javax.swing.JLabel labelStatus;
        private javax.swing.JComboBox<String> lengthDataBox;
        private javax.swing.JComboBox<String> nameBox;
        private javax.swing.JComboBox<String> parityBox;
        private Components_GUI.RoundPanel roundPanel2;
        private javax.swing.JComboBox<String> stopBitsBox;
        // End of variables declaration
    }

```

Файл StatusControlEvents.java

```

package Components_GUI;

public interface StatusConrtolEvents {

    public void SelectedControlMode(String elementName); //управление
автоматически/вручную
    public void SendControl(byte actuators);
}

```

Файл StatusControlForm.java

```

package Components_GUI;

```

```

import java.awt.Color;
import Main.GardenStatus;
import Main.global_variables;

public class StatusControlForm extends javax.swing.JPanel {

    private StatusConrtrlEvents event;

    public StatusControlForm(StatusConrtrlEvents event_param) {
        initComponents();
        setOpaque(false);

        gardenStatus = new GardenStatus();

        this.event = event_param;
        autoManualButtonGroup.add(autoButton);
        autoManualButtonGroup.add(manualButton);
        servolButtonGroup.add(servoOnButton);
        servolButtonGroup.add(servoOffButton);
        servo2ButtonGroup.add(stepperOnButton);
        servo2ButtonGroup.add(stepperOffButton);
        relayButtonGroup.add(relayOnButton);
        relayButtonGroup.add(relayOffButton);

        //Делаем кнопки выбора режимов неактивными
        autoButton.setEnabled(true);
        manualButton.setEnabled(false);
        servoOnButton.setEnabled(false);
        servoOffButton.setEnabled(false);
        stepperOnButton.setEnabled(false);
        stepperOffButton.setEnabled(false);
        relayOnButton.setEnabled(false);
        relayOffButton.setEnabled(false);
        configButton.setEnabled(false);

        temperature_progress.start();
        humidity_progress.start();
    }

```

```

private void autoButtonActionPerformed(java.awt.event.ActionEvent
evt) {

    //Делаем кнопки выбора режимов неактивными
    servoOnButton.setEnabled(false);
    servoOffButton.setEnabled(false);
    stepperOnButton.setEnabled(false);
    stepperOffButton.setEnabled(false);
    relayOnButton.setEnabled(false);
    relayOffButton.setEnabled(false);
    configButton.setEnabled(false);
    event.SelectedControlMode("Автоматически");
}

```

```

private void
manualButtonActionPerformed(java.awt.event.ActionEvent evt) {

    //Делаем кнопки выбора режимов активными
    servoOnButton.setEnabled(true);
    servoOffButton.setEnabled(true);
    stepperOnButton.setEnabled(true);
    stepperOffButton.setEnabled(true);
    relayOnButton.setEnabled(true);
    relayOffButton.setEnabled(true);
    configButton.setEnabled(true);
    event.SelectedControlMode("Вручную");
}

```

```

private void
configButtonActionPerformed(java.awt.event.ActionEvent evt) {

    byte actuators = 0;

    if (servoOnButton.isSelected())
    {
        actuators |= global_variables.MASK_SERVO;
    }

    if (stepperOnButton.isSelected())
    {
        actuators |= global_variables.MASK_STEPPER;
    }
}

```

```

    }

    if (relayOnButton.isSelected())
    {
        actuators |= global_variables.MASK_RELAY;
    }

    event.SendControl(actuators);
}

public void SetGardenStatus (GardenStatus gardenStatus_param)
{
    gardenStatus = gardenStatus_param;

    temperature_progress.setValue(gardenStatus.GetTemperaturePer());
    humidity_progress.setValue(gardenStatus.GetHumidityPer());

    String  serv1_str  = ((gardenStatus.GetActuators() & (~
global_variables.MASK_SERVO)) != 0) ? "ON" : "OFF";
    serv1_val_label.setText(serv1_str);
    String  serv2_str  = ((gardenStatus.GetActuators() & (~
global_variables.MASK_STEPPER)) != 0) ? "ON" : "OFF";
    serv2_val_label.setText(serv2_str);
    String  relay_str  = ((gardenStatus.GetActuators() & (~
global_variables.MASK_RELAY)) != 0) ? "ON" : "OFF";
    relay_val_label.setText(relay_str);
}

GardenStatus gardenStatus;
// Variables declaration - do not modify
private javax.swing.JRadioButton autoButton;
private javax.swing.ButtonGroup autoManualButtonGroup;
private javax.swing.JButton configButton;
private Components_GUI.Progress humidity_progress;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;

```

```

private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel8;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JRadioButton manualButton;
private Components_GUI.ProgressCircleUI progressCircleUI1;
private javax.swing.ButtonGroup relayButtonGroup;
private javax.swing.JRadioButton relayOffButton;
private javax.swing.JRadioButton relayOnButton;
private javax.swing.JLabel relay_val_label;
private Components_GUI.RoundPanel roundPanel1;
private Components_GUI.RoundPanel roundPanel2;
private javax.swing.ButtonGroup servolButtonGroup;
private javax.swing.JLabel servol_val_label;
private javax.swing.ButtonGroup servo2ButtonGroup;
private javax.swing.JLabel servo2_val_label;
private javax.swing.JRadioButton servoOffButton;
private javax.swing.JRadioButton servoOnButton;
private javax.swing.JRadioButton stepperOffButton;
private javax.swing.JRadioButton stepperOnButton;
private Components_GUI.Progress temperature_progress;
// End of variables declaration
}

```

Файл GardenStatus.java

```

package Main;

import Main.global_variables;
import java.util.*;
import java.util.concurrent.*;

//Класс хранит состояние системы. Изменяется только по приходу
соответствующего кадра по UART
public class GardenStatus {

```

```
public GardenStatus()
{
    temperature = 0;
    temperaturePer = 0;
    humidity = 0;
    humidityPer = 0;
    actuators = 0;
    controlMode = global_variables.AUTOMATIC_CONTROL;
    gate = new Semaphore(1);
}

public byte GetControlMode()
{
    return controlMode;
}

public void SetControlMode(byte control_param)
{
    controlMode = control_param;
}

public byte GetTemperature()
{
    return temperature;
}

public byte GetTemperaturePer()
{
    return temperaturePer;
}

public void SetTemperature(byte temperature_param)
{
    try
    {
        gate.acquire();
        temperature = temperature_param;
    }
}
```

```

        float      temp      =      (float)      temperature      /
((float)global_variables.MAX_TEMPERATURE      -
(float)global_variables.MIN_TEMPERATURE) / 2 * 100;
        temperaturePer = (byte) (50 + temp);
    }
    catch (InterruptedException exception)
    {
        Thread.currentThread().interrupt();
    }
    finally
    {
        gate.release();
    }
}

public byte GetHumidity() // Используется для вывода на графике в
относительных величинах
{
    return humidity;
}

public byte GetHumidityPer() // Используется для вывода на
графике в относительных величинах
{
    return humidityPer;
}

public void SetHumidity(byte humidityParam)
{
    try
    {
        gate.acquire();
        humidity = humidityParam;
        float      temp      =      (float)      humidity      /
((float)global_variables.MAX_HUMIDITY      -
(float)global_variables.MIN_HUMIDITY) * 100;
        humidityPer = (byte) temp;
    }
}

```

```

        catch (InterruptedException exception)
        {
            Thread.currentThread().interrupt();
        }
        finally
        {
            gate.release();
        }
    }

    public byte GetActuators()
    {
        return actuators;
    }

    public void SetActuators(byte actuatorsParam)
    {
        try
        {
            gate.acquire();
            actuators = actuatorsParam;
        }
        catch (InterruptedException exception)
        {
            Thread.currentThread().interrupt();
        }
        finally
        {
            gate.release();
        }
    }

    public void SetAutomaticControl ()
    {
        if (humidityPer >= global_variables.TRESHHOLD_HUMIDITY_PER)
        {
            actuators |= global_variables.MASK_SERVO;
        }
    }

```

```

        if (temperaturePer
    >=global_variables.TRESHHOLD_TEMPERATURE_PER)
    {
        actuators |= global_variables.MASK_STEPPER;
    }
}

public void setManualControl (byte actuatorsParam)
{
    actuators = actuatorsParam;
}

public byte GetControl ()
{
    return actuators;
}

private byte temperature;
private byte temperaturePer;
private byte humidity;
private byte humidityPer;
private byte actuators;
private byte controlMode;

private Semaphore gate;
}

```

Файл Main.java

```

package Main;

import SerialProcess.SerialProcess;
import SerialProcess.SerialPortConfig;
import Components_GUI.StatusControlForm;
import Components_GUI.SettingsForm;
import Components_GUI.MenuEvents;
import Components_GUI.Menu;
import Components_GUI.StatusConrtolEvents;
import Components_GUI.SettingsEvents;
import Main.GardenStatus;
import com.fazecast.jSerialComm.SerialPort;

```

```

import java.awt.Color;
import java.awt.Component;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.concurrent.atomic.AtomicBoolean;

public class Main extends javax.swing.JFrame {

    /**
     * Creates new form Main
     */
    public Main() {
        initComponents();

        controlEn = new AtomicBoolean(false);

        gardenStatus = new GardenStatus();

gardenStatus.SetControlMode(global_variables.AUTOMATIC_CONTROL);

        timer = null;

        //Запуск потока обработчика последовательного порта
        serialProcess = new SerialProcess();
        serial_thread = new Thread(serialProcess);
        serial_thread.start();

        StatusConrtolEvents      statusControlEvents      =      new
StatusConrtolEvents() {
            @Override
            public void SelectedControlMode(String elementName) {
                switch (elementName)
                {
                    case "Автоматически":

gardenStatus.SetControlMode(global_variables.AUTOMATIC_CONTROL);

                        break;
                    case "Вручную":

```

```

gardenStatus.SetControlMode(global_variables.MANUAL_CONTROL);
        break;
    }
}

public void SendControl(byte actuators_param)
{
    controlEn.set(true);
    actuatorsTX = actuators_param;
}
};

SettingsEvents settingsEvents = new SettingsEvents() {
    @Override
    public void SettingsUART(SerialPortConfig
serialPortConfig)
    {
        serialProcess.UpdateSettingsUART(serialPortConfig);
    }
};

statusControlForm = new
StatusControlForm(statusControlEvents);
settingsForm = new SettingsForm(settingsEvents);

MenuEvents event = new MenuEvents() {
    @Override
    public void SelectedMenuItem(String elementName) {
        switch (elementName)
        {
            case "Статус":
                showForm(statusControlForm);
                break;
            case "Статистика":
                //showForm(new StatusControlForm());
                break;
            case "Настройка":

```

```

        String[] UARTNames =
serialProcess.GetUARTList().toArray(new String[0]);
        settingsForm.SetUARTList(UARTNames);
        //Запрос списка интерфейсов UART
        showForm(settingsForm);
        break;
    }
}
};

menu2.initMenu(event);
showForm(statusControlForm);

restartTimer();
}

public void restartTimer() {
    timer = new Timer(200, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            //System.out.println("Timer executed");

            if (serialProcess.GetStatusReady().get() == true)
            {
                gardenStatus = serialProcess.GetStatus();
                statusControlForm.SetGardenStatus(gardenStatus);
            }

            //Опрос возможности передачи управляющего кадра
            if ((controlEn.get() == true) &&
(serialProcess.GetStatusReady().get() == true))
            {
                serialProcess.SendControl(actuatorsTX);
                controlEn.set(false);
            }

            // Логика для перезапуска таймера
            timer.stop(); // Остановка таймера

```

timer.setInitialDelay(1000); // Установка начальной
задержки

```
        timer.restart(); // Перезапуск таймера
    }
});
timer.setInitialDelay(1000); // Установка начальной задержки
timer.start(); // Запуск таймера
}
```

```
private void showForm(Component com) {
    jPanel1.removeAll();
    jPanel1.add(com);
    jPanel1.revalidate();
    jPanel1.repaint();
}
```

```
* @param args the command line arguments
*/
public static void main(String args[]) {
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Main().setVisible(true);
        }
    });
}
```

```
private SerialProcess serialProcess;
private GardenStatus gardenStatus;
volatile AtomicBoolean controlEn;
private byte actuatorsTX;
private StatusControlForm statusControlForm;
private SettingsForm settingsForm;
private SerialPortConfig serialPortConfig;
Timer timer;
Thread serial_thread;
// Variables declaration - do not modify
private javax.swing.JPanel jPanel1;
private Components_GUI.Menu menu2;
```

```

        private Components_GUI.RoundPanel roundPanel2;
        // End of variables declaration
    }

```

Файл global_variables.java

```

package Main;

// Хранение глобальных констант
public final class global_variables {
    public final static int BAD_STATUS_VALUE = -500;
    public final static byte HEAD_STATUS = 0x4B;
    public final static byte HEAD_CONTROL = 0x4A;
    public final static byte MASK_SERVO = 0x01;
    public final static byte MASK_STEPPER = 0x02;
    public final static byte MASK_RELAY = 0x04;
    public final static byte ON = 0x01;
    public final static byte OFF = 0x00;
    public final static byte AUTOMATIC_CONTROL = 0x01;
    public final static byte MANUAL_CONTROL = 0x00;
    public final static byte LENGTH_FRAME_STATUS = 0x05;
    public final static byte LENGTH_FRAME_CONTROL = 0x03;
    public final static byte MIN_TEMPERATURE = -50;
    public final static byte MAX_TEMPERATURE = 50;
    public final static byte TRESHHOLD_TEMPERATURE_PER = 65;
    //Уровень для включения/отключения исполнительного устройства
    public final static byte MIN_HUMIDITY = 0;
    public final static byte MAX_HUMIDITY = 100;
    public final static byte TRESHHOLD_HUMIDITY_PER = 60; //Уровень
    для включения/отключения исполнительного устройства
}

```

Файл SerialPortConfig.java

```

package SerialProcess;

public class SerialPortConfig {

    public SerialPortConfig()

```

```

{
    nameUART = null;
    baudRateValue = 0;
    lengthDataValue = 0;
    stopBitsValue = 0;
    parityValue = 0;
}

    public void SetSerialPortConfig(String nameUARTParam, int
baudRateValueParam, int lengthDataValueParam,
                                int stopBitsValueParam, int
parityValueParam)
    {
        nameUART = nameUARTParam;
        baudRateValue = baudRateValueParam;
        lengthDataValue = lengthDataValueParam;
        stopBitsValue = stopBitsValueParam;
        parityValue = parityValueParam;
    }

    public String GetNameUART ()
    {
        return nameUART;
    }

    public int GetbaudRateValue ()
    {
        return baudRateValue;
    }

    public int GetlengthDataValue ()
    {
        return lengthDataValue;
    }

    public int GetstopBitsValue ()
    {
        return stopBitsValue;
    }

```

```

    public int GetparityValue ()
    {
        return parityValue;
    }

    private String nameUART;
    private int baudRateValue;
    private int lengthDataValue;
    private int stopBitsValue;
    private int parityValue;
}

```

Файл SerialProcess.java

```

package SerialProcess;

import Main.GardenStatus;
import com.fazecast.jSerialComm.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.HashMap;
import java.util.Map;
import Main.global_variables;

public class SerialProcess implements Runnable {

    public SerialProcess() {
        this.statusReady = new AtomicBoolean(false);
        this.controlReady = new AtomicBoolean(false);
        this.gardenStatus = new GardenStatus();
        this.running = new AtomicBoolean(true);
        this.settingsChanged = new AtomicBoolean(false);
        this.serialPortConfig = new SerialPortConfig();
        this.sp = null;
    }

    public AtomicBoolean GetControlReady ()
    {

```

```

        return controlReady;
    }

    public AtomicBoolean GetStatusReady ()
    {
        return statusReady;
    }

    public void SendControl (byte actuatorsParam) // Установка
запроса на отправку байта с управлением исполняющими устройствами
    {
        if (controlReady.get() == true)
        {
            controlReady.set(false);
            actuators_value = actuatorsParam;
        }
    }

    public GardenStatus GetStatus ()
    {
        GardenStatus gardenStatus_out = gardenStatus;
        statusReady.set(false);
        return gardenStatus_out;
    }

    public List<String> GetUARTList ()
    {
        List<String> portNames = new ArrayList<>();
        // Поиск последовательного порта
        SerialPort[] ports = SerialPort.getCommPorts();

        // Заполняем список именами портов
        for (SerialPort port : ports) {
            portNames.add(port.getSystemPortName());
        }

        return portNames;
    }

```

```

        public void UpdateSettingsUART (SerialPortConfig
serialPortConfigParam)
    {
        serialPortConfig = serialPortConfigParam;
        settingsChanged.set (true);
    }

    public void OpenUART ()
    {
        SerialPort[] ports = SerialPort.getCommPorts();
        try {
            for (SerialPort port : ports) {
                if
(port.getSystemPortName().equals(serialPortConfig.GetNameUART())) {
                    sp = port;
                    break;
                }
            }

            if (sp == null) {
                System.out.println("Порт не найден.");
                return;
            }

            // Открытие порта
            sp.setBaudRate(serialPortConfig.GetbaudRateValue());
            sp.setNumDataBits(serialPortConfig.GetlengthDataValue());
            sp.setNumStopBits(serialPortConfig.GetstopBitsValue());
            sp.setParity(serialPortConfig.GetparityValue());

            if (!sp.openPort()) {
                System.out.println("Не удалось открыть порт.");
                return;
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

```

```

    }

    public void CloseUART ()
    {
        try
        {
            if((sp != null) && (sp.isOpen()))
            {
                sp.closePort();
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void ReopenUART ()
    {
        CloseUART();
        OpenUART();
    }

    // Функция для расчета CRC8
    //Алгоритм использует полином  $x^8 + x^5 + x^4 + 1$  (0x8C)
    public static short CRC8(byte[] data, int len) {
        short crc = 0x00; // Инициализация CRC с 0
        for (int i = 0; i < len; i++) {
            short extract = (short)(data[i] & 0xFF); // Извлечение
байта данных (преобразование в беззнаковый)
            for (short tempI = 8; tempI > 0; --tempI) { // Перебор
каждого бита байта данных
                short sum = (short) ((crc ^ extract) & 0x01); //
Вычисление младшего бита суммы
                crc >>= 1; // Сдвиг CRC вправо
                if (sum != 0) { // Если сумма не равна 0, то
выполняем XOR с полиномом 0x8C
                    crc ^= 0x8C;
                }
            }
        }
    }

```

```

        extract >>= 1; // Сдвиг извлеченного байта вправо
    }
}
return crc; // Возвращение конечного значения CRC
}

public void run()
{

    byte[]          buffer_rx          =          new
byte[global_variables.LENGTH_FRAME_STATUS];
    boolean header_found = false;
    boolean crc_error = false;

    //в бесконечном цикле слушаем порт
    try
    {
        while(running.get())
        {
            if (settingsChanged.get())
            {
                if (sp == null)
                {
                    OpenUART();
                    System.out.println("Порт успешно открыт");
                }
                else
                {
                    ReopenUART();
                    System.out.println("Порт успешно переоткрыт");
                }
                controlReady.set(true);
                settingsChanged.set(false);
            }

            if (sp != null)
            {
                if (!statusReady.get())
                {

```

```

        //Ожидание кадра на входном интерфейсе
        if (crc_error)
        {
            sp.readBytes(buffer_rx, 1);
            crc_error = false;
        }
        //Производим поиск заголовка
        if (sp.bytesAvailable() >= 1)
        {
            sp.readBytes(buffer_rx, 1);
            if (buffer_rx[0] ==
global_variables.HEAD_STATUS)
            {
                if (sp.bytesAvailable() >=
(global_variables.LENGTH_FRAME_STATUS - 1))
                {

                    sp.readBytes(buffer_rx,
buffer_rx.length - 1, 1);

                    Byte header = buffer_rx[0];
                    Byte temperature = buffer_rx[1];
                    Byte humidity = buffer_rx[2];
                    Byte actuators = buffer_rx[3];
                    Byte crc = buffer_rx[4];
                    buffer_rx[4] = 0;

                    // Проверка заголовка и CRC
                    byte calculatedCrc =
(byte)CRC8(buffer_rx, 4);

                    if (header == (byte)
global_variables.HEAD_STATUS && crc == calculatedCrc) {

                        System.out.println("Temperature: " + temperature);

                        System.out.println("Humidity:
" + humidity);

                        System.out.println("Servo 1
on/off: " + actuators);

                        System.out.println("Servo 2
on/off: " + actuators);

```

```

gardenStatus.SetTemperature(temperature);

gardenStatus.SetHumidity(humidity);

gardenStatus.SetActuators(actuators);

        statusReady.set(true);
    } else {
        System.out.println("Ошибка
CRC или заголовка");

        crc_error = true;
    }
}

}

}

// Отправка кадра с выходного интерфейса
if (controlReady.get() == false)
{
    byte[] buffer = new
byte[global_variables.LENGTH_FRAME_CONTROL];

    buffer[0] = global_variables.HEAD_CONTROL;
    buffer[1] = actuators_value;

    byte calculatedCrc = (byte)CRC8(buffer, 2);
    buffer[2] = calculatedCrc;

    controlReady.set(true);
}

}

}

catch (Exception e)
{
    System.out.println("UART exception!");
}

}

```

```

public void stop()
{
    running.set(false);
    CloseUART();
}

//Интерфейсы взаимодействия с графикой
private volatile AtomicBoolean statusReady;
private volatile AtomicBoolean controlReady; // Гарантируем
атомарность выполнения операций
private volatile AtomicBoolean running;
private volatile AtomicBoolean settingsChanged;
SerialPortConfig serialPortConfig;

private Byte actuators_value;
private GardenStatus gardenStatus;

SerialPort sp;
}

```

Приложение В

(Обязательное)

«Система контроля климата в теплице»

Презентация



Система контроля климата в теплице

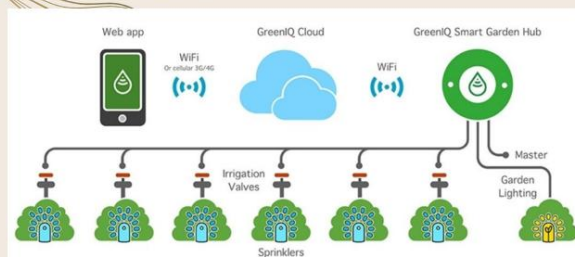


Актуальность темы

- Прогресс. Развитие информационных технологий открывает новые возможности для управления и оптимизации аграрных процессов.
- Эффективность. Системы контроля за климатом в теплицах могут существенно снизить операционные затраты.
- Рациональность. Современные теплицы позволяют значительно сократить использования воды, удобрений и энергии, что является критически важным в условиях дефицита природных ресурсов.



Анализ готовых решений



Собственные автоматизированный теплицы и готовые решения

- Готовые решения лучше подходят для быстрого внедрения и снижения рисков, но могут быть менее гибкими и дороже в долгосрочной перспективе. При выборе готовых решений большую роль имеет адаптируемость их функционала под конкретную область применения. В ходе анализа было установлено, что готовые решения предоставляют ограниченный функционал, который недостаточен для эффективного применения в различных условиях.
- Исходя из результатов анализа, было принято решение о создании собственной “умной” теплицы. Основная цель проекта заключается в предоставлении конечному пользователю оптимального функционала и нужной степени автоматизации системы для обеспечения требуемого результата.



Постановка задачи

- разработать схему теплицы;
- выбрать и приобрести компоненты;
- разработать систему сбора данных;
- разработать систему управления;
- создать интерфейс для мониторинга и управления;
- провести тестирование и отладку системы;
- документировать проект: описать архитектуру системы, используемые компоненты и их взаимосвязи.

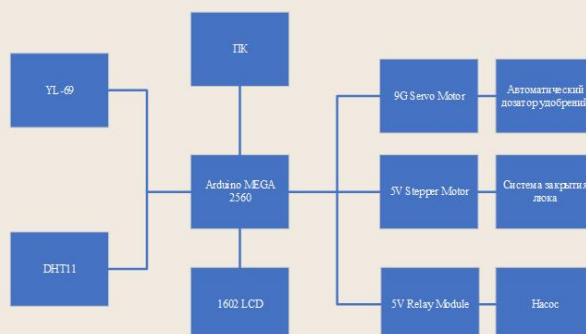


Диаграмма вариантов использования





Структура разработанного макета



Сервер автоматизированной теплицы

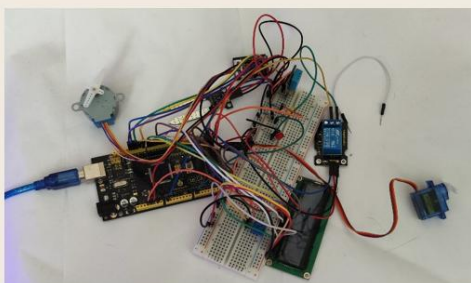
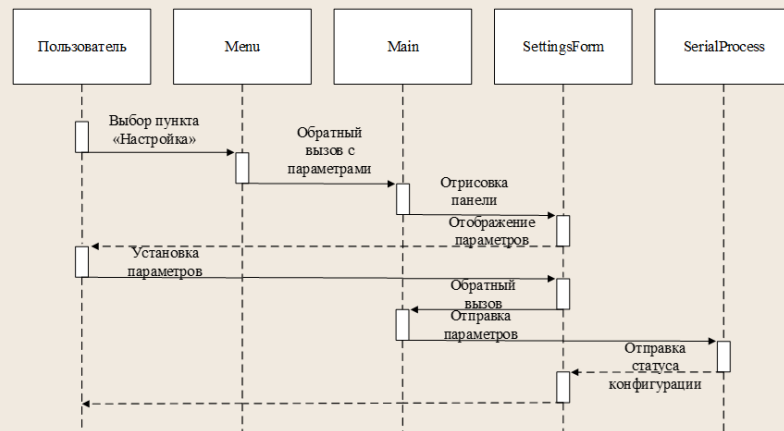


Диаграмма последовательности процедуры настройки интерфейса UART



Окно настройки интерфейса UART

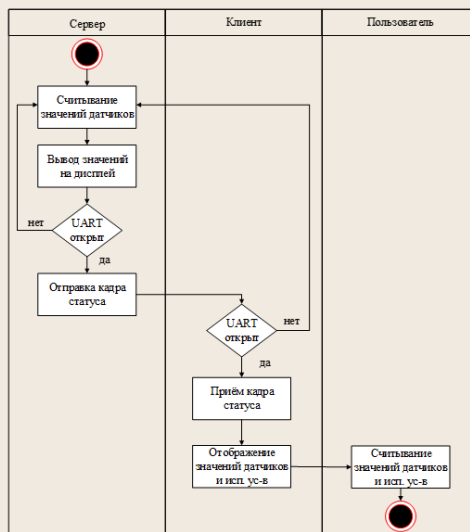
The screenshot shows a window titled 'Настройки интерфейса UART' (UART interface settings). On the left, there is a sidebar with two tabs: 'Статус' (Status) and 'Настройка' (Settings), with 'Настройка' being the active tab. The main area contains several configuration fields:

- Название: COM1 (dropdown menu)
- Скорость: 9600 (dropdown menu) bод/c
- Длина данных: 8 (dropdown menu) бит
- Кол-во стоповых битов: 1 (dropdown menu) бит
- Установка паритета: NO (dropdown menu)

Below these fields is a button labeled 'Настройка' (Settings). At the bottom, there is a 'Статус:' (Status:) label.



Диаграмма деятельности процедуры отображения статуса теплицы



Область «Текущий статус» в StatusControlFor

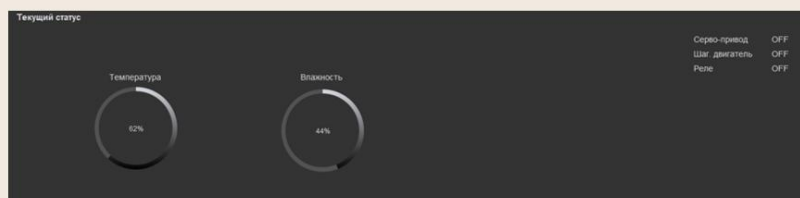
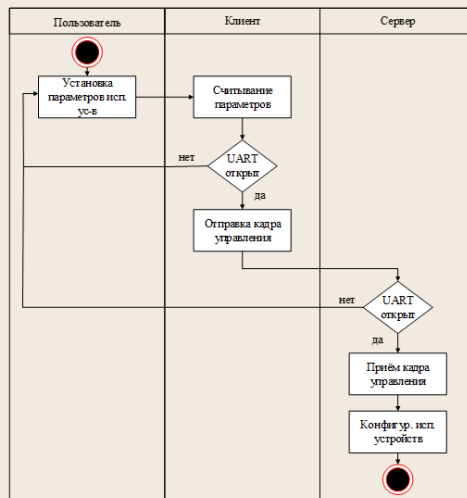
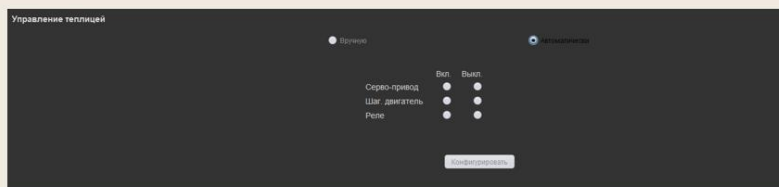




Диаграмма деятельности процедуры управления теплицей



Область «Управление теплицей» в StatusControlForm





ПГУ ФВТ 2024

Спасибо за внимание!

