Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Архитектура вычислительных систем»

К защите допустить:
И.О. Заведующего кафедрой информатики
_____ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
курсового проекта
на тему

КОНТРОЛЛЕР БЕЗОПАСНОСТИ ОТКРЫТИЯ КУПОЛА УДАЛЕННОЙ ОБСЕРВАТОРИИ НА ARDUINO НА ОСНОВАНИИ НЕСКОЛЬКИХ ДАТЧИКОВ (ВЛАЖНОСТИ, СКОРОСТИ ПОТОКА ВЕТРА, ДАВЛЕНИЯ, ОТКРЫТИЯ КУПОЛА)

БГУИР КП 1-40 04 01 013 ПЗ

Студент А.С. Кудош

Руководитель А. А. Калиновская

Нормоконтролёр А. А. Калиновская

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы наблюдается значительный рост интереса к астрономии и астрономическим наблюдениям. Одним из ключевых аспектов работы астрономических обсерваторий является надежное и безопасное управление куполами телескопов. Открытие и закрытие купола – это критически важные процессы, которые должны учитывать различные внешние условия, такие как влажность, скорость потока ветра и атмосферное давление.

В данном курсовом проекте будет разработан контроллер безопасности открытия купола удаленной обсерватории на платформе *Arduino*. Система будет использовать несколько датчиков для мониторинга окружающей среды и принятия решений о целесообразности открытия купола. Использование нескольких датчиков одного типа позволит значительно повысить надежность системы: в случае отключения или неисправности одного из датчиков, данные других датчиков смогут компенсировать недостающую информацию, что снизит риск принятия неверных решений.

Обработка ошибок станет важной частью системы. Для этого будет реализована система диагностики, которая будет отслеживать состояние датчиков и их показания. В случае обнаружения аномалий или несоответствий в данных, контроллер сможет автоматически отключать неисправные датчики и оповещать оператора о возникшей проблеме. Это обеспечит дополнительный уровень безопасности и надежности, минимизируя вероятность сбоев в работе системы.

Проект включает в себя выбор и интеграцию датчиков, разработку алгоритмов обработки данных. В результате, созданная система обеспечит автоматическое принятие решений на основе собранной информации, что сделает процесс открытия купола более безопасным и эффективным.

Таким образом, целью данного курсового проекта является создание надежного решения, которое позволит улучшить эксплуатацию обсерваторий и повысить качество астрономических исследований.

Пояснительная записка оформлена в соответствии с СТП 01-2024 [1].

ТЕРМИНЫ, СОКРАЩЕНИЯ И ОПРЕДЕЛЕНИЯ

В настоящем документе используются следующие термины, сокращения и определения:

1 *Arduino* – платформа для создания прототипов, объединяющая аппаратное и программное обеспечение для разработки интерактивных проектов.

2 Микроконтроллер – интегрированная схема, выполняющая вычислительные задачи и управляющая подключенными компонентами. Примеры: *ATmega328P*, *ATmega2560*.

3 IDE (Integrated Development Environment) – интегрированная среда разработки, используемая для написания, компиляции и загрузки программ на плату *Arduino*.

4 Скетч – программа, написанная для *Arduino*, состоящая из функций *setup*() и *loop*().

5 Flash-память – тип памяти, используемой для хранения программ.

6 *SRAM* (*Static Random Access Memory*) – память для временного хранения данных во время выполнения приложения.

7 *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*) – память для долговременного хранения информации.

8 Цифровые порты – порты, которые могут принимать или передавать только два состояния: высокий (*HIGH*) или низкий (*LOW*).

9 Аналоговые порты – порты, которые могут считывать или передавать переменные значения, например, напряжение от датчиков.

10 *SPI* (*Serial Peripheral Interface*) – стандартный интерфейс для связи между микроконтроллерами и периферийными устройствами.

11 *I2C* (*Inter-Integrated Circuit*) – интерфейс для подключения нескольких устройств с использованием всего двух проводов.

12 *UART* (*Universal Asynchronous Receiver-Transmitter*) – стандартный способ асинхронной передачи данных между устройствами.

13 Кварцевый резонатор – компонент, обеспечивающий стабильную тактовую частоту для работы микроконтроллера.

14 Регулятор напряжения – устройство, понижающее входное напряжение до безопасных уровней для работы микроконтроллера.

15 *USB-TTL* конвертер – устройство для преобразования сигналов USB в сигналы последовательной передачи данных и наоборот.

16 Открытый исходный код – подход, при котором исходный код программного обеспечения доступен для использования, изменения и распространения.

17 *IoT* (*Internet of Things*) – концепция, описывающая сеть физических объектов, подключенных к Интернету и способных обмениваться данными.

18 Компилятор – инструмент, преобразующий написанный код в машинный язык, понятный микроконтроллеру.

19 Загрузчик – программа, передающая скомпилированный код на *Arduino* через *USB*.

20 Библиотеки – наборы функций, расширяющие возможности платформы. Делятся на стандартные и пользовательские.

21 Стек программного обеспечения – архитектура программного обеспечения, состоящая из нескольких уровней: аппаратный уровень, библиотеки и *API*, программный уровень.

22 Инициализация – первый этап выполнения программы, во время которого выполняется функция *setup*().

23 Кроссплатформенность – способность программного обеспечения работать на различных операционных системах, таких как *Windows*, *macOS* и *Linux*.

24 *API* (*Application Programming Interface*) – интерфейс программирования приложений, позволяющий взаимодействовать между различными компонентами программного обеспечения.

25 Скетч – программа, написанная для *Arduino*, состоящая из функций *setup*() и *loop*(), реализующая логику работы устройства.

26 Прерывания – механизм, позволяющий мгновенно реагировать на события, такие как изменения состояния датчиков.

27 Контроллер безопасности – устройство, предназначенное для автоматизации процессов открытия и закрытия купола обсерватории, обеспечивая безопасность и защиту оборудования.

28 Автоматизация – использование технологий для управления процессами с минимальным вмешательством человека, что повышает эффективность и безопасность работы.

29 Датчики – устройства, используемые для измерения различных параметров (например, положение купола, наличие препятствий, погодные условия) и передачи данных для обработки.

30 Метеостанции – устройства, измеряющие погодные параметры, такие как температура, влажность, скорость ветра, обеспечивая информацию для принятия безопасных решений.

31 Положение купола – параметр, определяющий текущее состояние (открыто или закрыто) купола обсерватории.

32 Исполнительные механизмы – устройства, которые выполняют физические действия (например, моторы, сервоприводы) на основе команд, полученных от контроллера.

33 Индикаторы состояния – сигнализаторы, отображающие текущее состояние системы (например, открытие или закрытие купола).

34 Безопасность – обеспечение защиты оборудования и пользователей от потенциальных угроз и аварийных ситуаций.

35 Датчики влажности – устройства, измеряющие уровень влажности в окружающей среде и предотвращающие открытие купола при превышении заданного порога.

36 Датчики давления – устройства, измеряющие атмосферное давление, которые не позволяют открывать купол при снижении давления ниже установленного уровня.

37 Датчики скорости ветра – устройства, определяющие скорость ветра

и блокирующие открытие купола, если ветер превышает безопасный уровень.

38 Герконы – датчики, контролирующие положение купола (открыт/закрыт) для обеспечения безопасности операций с куполом.

39 Обработка ошибок – механизм, позволяющий контроллеру обнаруживать и реагировать на неисправности в работе датчиков.

40 Аномалии в показаниях – ситуации, когда показания датчиков значительно отличаются друг от друга, указывая на возможные неисправности.

41 Светодиод – индикатор, используемый для визуального оповещения пользователя о состоянии системы (ошибки, разрешение на открытие купола).

42 Пороговые значения – заранее определенные уровни, при превышении или понижении которых контроллер принимает меры, чтобы предотвратить открытие купола.

43 Герконы – механические датчики, определяющие положение купола (открыт/закрыт), обеспечивая безопасность операций.

44 Анемометры – устройства для измерения скорости ветра, критически важные для определения безопасности открытия купола.

45 Мультиплексоры – устройства, позволяющие подключать несколько датчиков к одному входу *Arduino*, оптимизируя использование пинов и упрощая подключение.

46 Инициализация датчиков – процесс настройки датчиков для считывания данных, включая передачу необходимых пинов и использование библиотек.

47 Интерфейсы подключения – протоколы, используемые для соединения датчиков с контроллером (например, *I2C, SPI*).

48 Энергопотребление датчиков – характеристика, определяющая, сколько энергии требует датчик для работы, что особенно важно для удаленных систем.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1 Структура и архитектура вычислительной системы

Arduino [2] – это платформа для создания прототипов, которая сочетает в себе аппаратное и программное обеспечение. Она предоставляет пользователям возможность разрабатывать интерактивные проекты, используя доступные компоненты и простые в изучении языки программирования. В этой главе мы рассмотрим структуру и архитектуру вычислительной системы *Arduino*, а также её основные компоненты и принципы работы.

1.1.1 Аппаратное обеспечение [3]. Платформы *Arduino* представляют собой мощные инструменты для разработки и прототипирования электронных устройств. В центре каждой платы находится микроконтроллер, который выполняет все вычислительные задачи и управляет подключенными компонентами. Наиболее распространённые модели микроконтроллеров, используемые в различных платах *Arduino*, включают *ATmega328P* (рисунок 1.1), *ATmega2560* (рисунок 1.2), и другие, каждая из которых имеет свои особенности и характеристики.

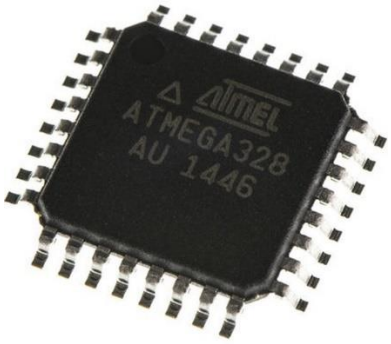


Рисунок 1.1 – Микроконтроллер *Atmega328*

Рисунок 1.2 – Микроконтроллер *Atmega2560*

Микроконтроллеры *Arduino* обладают высокой производительностью с тактовой частотой, достигающей 16 МГц и выше. Они оснащены различными

типами памяти: *flash*-память для хранения программ, *SRAM* для временного хранения данных во время выполнения приложения и *EEPROM* для долговременного хранения информации. Эти характеристики делают *Arduino* подходящей платформой для реализации широкого спектра проектов – от простых до сложных.

Все платы *Arduino* поддерживают разнообразные периферийные устройства, что позволяет легко интегрировать их в проекты. Они оснащены цифровыми и аналоговыми входами/выходами, которые позволяют подключать сенсоры, светодиоды и другие компоненты. В зависимости от модели, количество цифровых и аналоговых контактов может варьироваться, предоставляя пользователям различные возможности для расширения функциональности.

Arduino поддерживает несколько стандартных интерфейсов связи, таких как *SPI, I2C* и *UART*, что упрощает взаимодействие с другими устройствами и модулями. Это позволяет создавать сложные системы, в которых разные устройства могут обмениваться данными, расширяя возможности проектирования.

Платы *Arduino* могут получать питание от различных источников, что делает их универсальными. Основным способом питания является подключение через *USB*-порт, что удобно для разработки и тестирования. Также поддерживается подача питания через разъемы постоянного тока, что позволяет использовать внешние источники, такие как батареи или адаптеры, обеспечивая мобильность проектов.

На всех платах *Arduino* присутствуют ключевые компоненты, такие как кварцевые резонаторы для обеспечения стабильной тактовой частоты, регуляторы напряжения для понижения входного напряжения до безопасных уровней, а также *USB-TTL* конвертеры для взаимодействия с компьютерами. Кнопка сброса, расположенная на платах, позволяет перезапустить микроконтроллер и начать выполнение программы заново.

1.1.2 Программное обеспечение [4]. Для разработки программ для *Arduino* используется *Arduino Integrated Development Environment* (*IDE*) – основная среда разработки, которая обеспечивает пользователям удобный интерфейс для написания, компиляции и загрузки программ (скетчей) на плату. *Arduino* IDE предлагает множество функций, которые делают процесс разработки интуитивно понятным и доступным даже для начинающих пользователей.

Arduino IDE поддерживает синтаксис языка программирования, основанный на *C*/*C++*. Это позволяет разработчикам использовать знакомые конструкции и подходы, что значительно упрощает процесс программирования. Среди основных возможностей *IDE* можно выделить:

1 Редактор кода: Удобный текстовый редактор, который поддерживает подсветку синтаксиса, что помогает пользователям быстрее находить ошибки.

Автозавершение кода и функции проверки синтаксиса делают процесс написания программ более эффективным.

2 Библиотеки: *Arduino IDE* предоставляет доступ к множеству библиотек, которые содержат готовые функции для работы с различными компонентами, такими как сенсоры, дисплеи, модули связи и многие другие. Это позволяет разработчикам сосредоточиться на логике приложения, не тратя время на низкоуровневое программирование.

3 Доступ к примерам: *IDE* включает в себя обширную коллекцию примеров, которые могут служить отправной точкой для новых проектов. Эти примеры охватывают широкий спектр возможностей платформы и помогают пользователям понять, как использовать различные функции и компоненты.

Программы, написанные для *Arduino*, имеют четкую структуру, основанную на двух основных функциях:

1 *setup*(): Эта функция выполняется один раз при старте программы и используется для инициализации переменных, настройки пинов и подключения к устройствам. Здесь задаются начальные состояния и конфигурации, необходимые для работы программы.

2 *loop*(): Эта функция выполняется в бесконечном цикле, обеспечивая постоянное выполнение кода. Внутри этой функции размещается основной алгоритм, который отвечает за взаимодействие с внешними устройствами и обработку данных.

Arduino взаимодействует с внешними устройствами через цифровые и аналоговые порты. Например, для считывания значения с аналогового датчика используется функция *analogRead*(), а для управления светодиодом – *digitalWrite*(). Эти простые команды позволяют легко и эффективно работать с различными компонентами.

Одним из ключевых преимуществ платформы *Arduino* является способность обрабатывать данные в реальном времени. Это открывает возможности для создания интерактивных проектов, таких как системы автоматизации, роботы и умные устройства. Использование прерываний и таймеров позволяет разработчикам эффективно управлять временем выполнения задач, что критически важно для проектов, требующих высокой точности и быстродействия.

Прерывания позволяют реагировать на события мгновенно, например, на нажатие кнопки или изменение состояния датчика, не дожидаясь завершения выполнения текущего кода. Таймеры, в свою очередь, позволяют выполнять определённые задачи через заданные интервалы времени, что удобно для периодической проверки состояния сенсоров или управления устройствами.

1.2 История, версии и достоинства

Arduino была создана в 2005 году группой итальянских разработчиков, среди которых были Массимо Банци (*Massimo Banzi*), Девид Куартиллье (*David Cuartielles*), Том Иго (*Tom Igoe*), Джанлука Мартино (*Gianluca Martino*), Девид Меллис (*David Mellis*), Николас Замбетти (*Nicholas Zambetti*) и Валерий Шумятский (*Valeriy Shymatskiy*) [5]. Изначально платформа разрабатывалась для студентов, стремившихся освоить программирование и электронику в Институте проектирования взаимодействий в Ивреа, Италия.

Проект быстро завоевал популярность благодаря своей доступности и простоте использования. *Arduino* предлагает недорогие микроконтроллерные платы, которые можно легко программировать с помощью специально разработанной интегрированной среды разработки (*IDE*). Это сделало его идеальным инструментом для новичков и энтузиастов, желающих реализовать свои идеи в области электронных самоделок.

В 2006 году *Arduino* был официально представлен широкой публике на различных выставках и конференциях, что способствовало его популяризации. С тех пор платформа активно развивалась, и её возможности расширились. *Arduino* стала основой для множества проектов в области искусственного интеллекта, робототехники, автоматизации и Интернета вещей (*IoT*). Например, пользователи начали разрабатывать системы домашней автоматизации, интерактивные дисплеи, алкотестеры и даже устройства для анализа ДНК.

Одной из ключевых особенностей *Arduino* является открытый исходный код, что позволяет разработчикам создавать свои собственные решения и делиться ими с сообществом. Эта демократизация процесса разработки сделала *Arduino* доступной для широкой аудитории, от студентов до профессионалов. К 2023 году было продано более 250,000 комплектов *Arduino*, не считая множества клонов, что подчеркивает огромный интерес к этой платформе.

Важным аспектом успешного роста *Arduino* стало сотрудничество с другими проектами, такими как *Processing* и *Wiring*, что обеспечило удобные инструменты для работы с микроконтроллерами. В результате, даже те, кто не имел опыта работы с электроникой, смогли легко погрузиться в создание своих собственных устройств.

В последние годы *Arduino* продолжает развиваться, предлагая новые модели плат и расширяя свои функциональные возможности. Появились такие варианты, как *Arduino Mega*, *Arduino Nano* и *LilyPad Arduino*, каждая из которых ориентирована на различные потребности разработчиков. Это позволило *Arduino* оставаться на передовой технологий, соответствуя требованиям быстро меняющегося рынка и интересам сообщества.

Рисунок 1.3 – Команда разработчиков *Arduino*

1.2.1 Версии *Arduino* [6]. С момента своего появления *Arduino* претерпел множество изменений и обновлений. Основные версии плат *Arduino* включают:

1 *Arduino Uno* (рисунок 1.4). Одна из самых популярных моделей, основанная на *ATmega328*. Широко используется для образовательных целей и простых проектов.

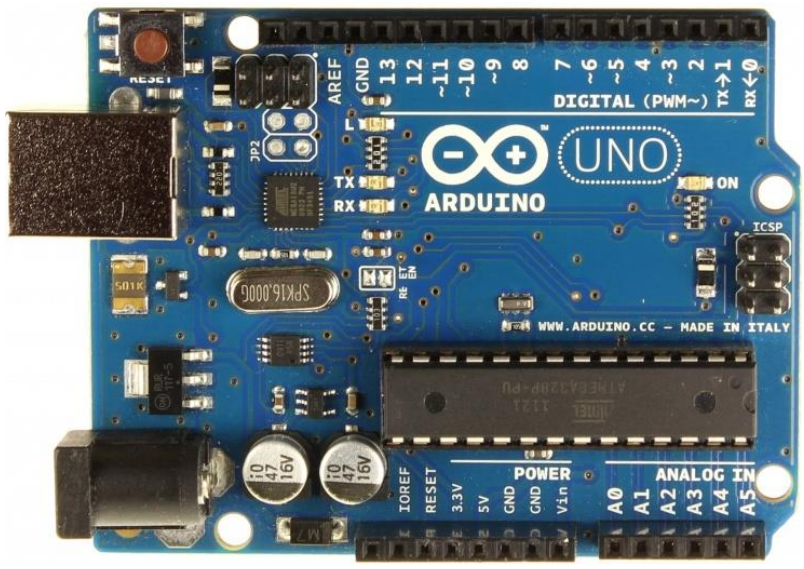


Рисунок 1.4 – Плата *Arduino Uno*

2 *Arduino Mega* (рисунок 1.5). Основан на *ATmega2560* и предлагает большее количество входов/выходов и памяти. Подходит для сложных проектов.

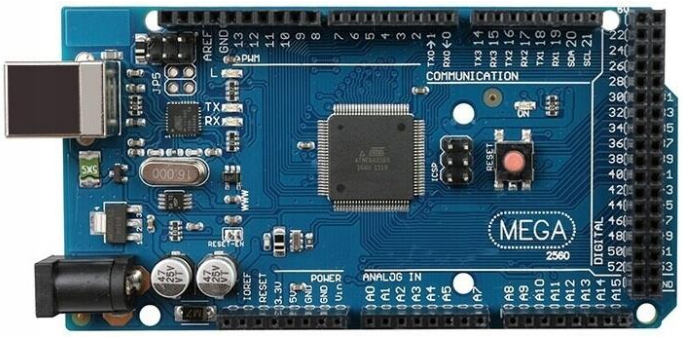


Рисунок 1.5 – Плата *Arduino Mega*

3 *Arduino Nano* (рисунок 1.6). Компактная версия, идеально подходящая для проектов с ограниченным пространством.

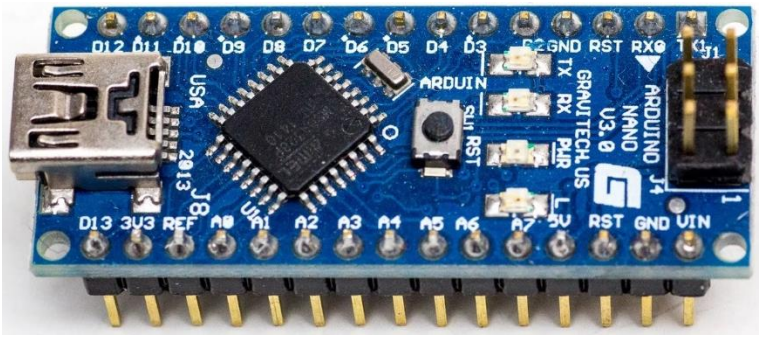


Рисунок 1.6 – Плата *Arduino Nano*

4 *Arduino Leonardo* (рисунок 1.7). Позволяет эмулировать *USB*-устройства, что делает его полезным для создания клавиатур и мышей.

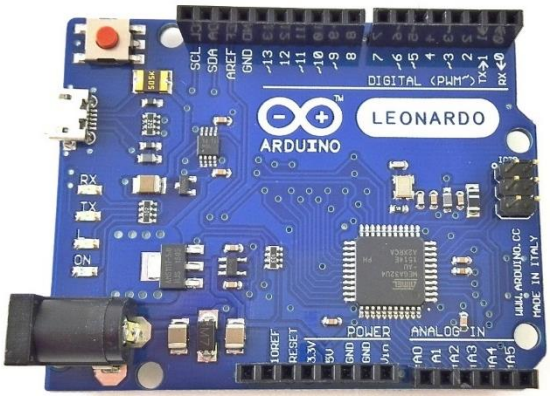


Рисунок 1.7 – Плата *Arduino Leonardo*

5 *Arduino Due* (рисунок 1.8). Первая модель с 32-битным *ARM* процессором, что обеспечивает более высокую производительность и возможности.

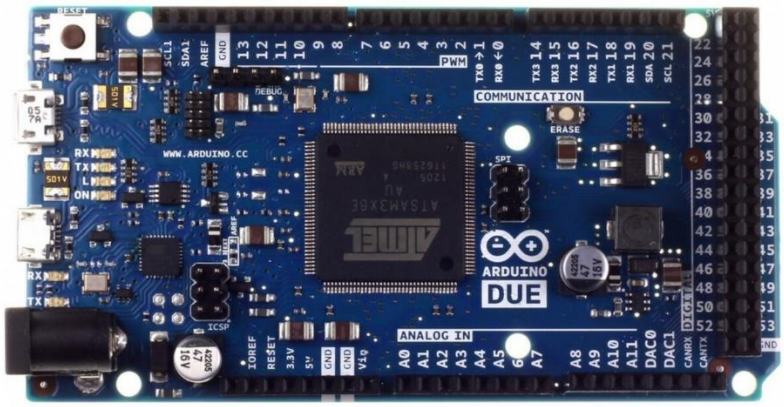


Рисунок 1.8 – Плата *Arduino Due*

Существуют также специальные версии, такие как *Arduino MKR* (рисунок 1.9), ориентированные на проекты *IoT*, с встроенными модулями связи.

Рисунок 1.9 – Плата *Arduino MKR*

1.2.2 Достоинства *Arduino* [7]. *Arduino* обладает рядом достоинств, которые делают его привлекательным для пользователей:

1 Простота использования. Платформа разрабатывалась с акцентом на доступность. *Arduino IDE* интуитивно понятен, а наличие множества обучающих материалов и сообществ облегчает процесс обучения.

2 Гибкость. *Arduino* поддерживает широкий спектр датчиков, модулей и других компонентов, что позволяет создавать разнообразные проекты – от простых до сложных.

3 Большое сообщество. Существует активное сообщество разработчиков, которое делится опытом, библиотеками и проектами. Это предоставляет новичкам возможность получать помощь и вдохновение.

4 Доступность. *Arduino*-платы и компоненты доступны по разумной цене, что делает их доступными для широкой аудитории, включая студентов

и хобби-энтузиастов.

5 Непрерывное развитие. *Arduino* продолжает развиваться, добавляя новые функции и улучшая существующие модели. Это обеспечивает совместимость с современными технологиями и тенденциями.

История *Arduino* – это история успеха, основанная на идее сделать технологии доступными для всех. Разнообразие версий и богатый функционал платформы открывают широкие возможности для применения в различных областях.

1.3 Обоснование выбора вычислительной системы

Arduino была выбрана основываясь на следующих факторах:

1 Простота использования. *Arduino* предлагает интуитивно понятный интерфейс и базовый язык программирования, что позволяет быстро разрабатывать и тестировать прототипы. Это важно для быстрого внедрения системы контроля.

2 Гибкость и масштабируемость. *Arduino* поддерживает множество датчиков и модулей.

3 Надежность. Платформы *Arduino* зарекомендовали себя как надежные и стабильные. Используя проверенные компоненты и схемы, можно создать систему, которая будет работать в разнообразных условиях.

4 Экономичность. *Arduino*-платы и необходимые компоненты доступны по разумным ценам. Это позволяет сократить затраты на разработку и внедрение системы.

5 Сообщество и поддержка. Активное сообщество разработчиков предоставляет доступ к множеству библиотек и примеров, что облегчает процесс разработки и внедрения системы. В случае возникновения вопросов или проблем, можно легко найти решение.

Выбор вычислительной системы для создания контроллера безопасности открытия купола удаленной обсерватории обоснован использованием *Arduino*. Простота, гибкость, надежность и доступность этой платформы делают её идеальным выбором для реализации данной задачи. Система, основанная на *Arduino*, обеспечит эффективное управление и безопасность, что критически важно для функционирования обсерватории.

1.4 Вывод

Раздел демонстрирует, как *Arduino* сочетает в себе доступное аппаратное обеспечение и простое программное обеспечение, что делает её идеальной для создания интерактивных проектов. Основные компоненты, такие как микроконтроллеры *ATmega328* и *ATmega2560*, предоставляют достаточную производительность для реализации различных задач, включая управление сенсорами и другими устройствами.

История *Arduino* – это увлекательный путь, который подтверждает его невероятную популярность и динамичное развитие в мире технологий. С

момента своего появления платформа привлекла внимание как любителей, так и профессионалов, предлагая множество версий, каждая из которых обладает уникальными функциями и характеристиками, чтобы удовлетворить разнообразные потребности пользователей.

Одним из ключевых достоинств *Arduino* является его простота использования. Даже те, кто не имеет глубоких знаний в программировании или электронике, могут легко освоить основы и начать создавать свои первые проекты. Доступность платформы также играет важную роль: *Arduino* предлагает широкий выбор компонентов и аксессуаров по разумным ценам, что делает его доступным для широкой аудитории

Выбор *Arduino* в качестве вычислительной системы для создания контроллера безопасности открытия купола удаленной обсерватории обоснован не только её надёжностью, экономичностью и возможностью быстрого прототипирования, но и множеством других факторов, которые делают эту платформу идеальным решением для подобных задач.

Надёжность *Arduino* является одним из её главных преимуществ. Платформа разработана таким образом, чтобы работать в различных условиях, что особенно важно для удалённых объектов, таких как обсерватории. Благодаря простоте конструкции и высоким стандартам качества, *Arduino* может эффективно функционировать даже в неблагоприятных погодных условиях, что критично для работы обсерватории, где контроль открывания и закрывания купола зависит от внешних факторов, таких как дождь или сильный ветер.

В заключение, *Arduino* представляет собой мощный инструмент, который не только отвечает требованиям надёжности и экономичности, но и предоставляет разработчикам все необходимые ресурсы для реализации инновационных проектов в различных областях. В частности, для создания контроллера безопасности открытия купола удаленной обсерватории, эта платформа становится идеальным выбором, обеспечивая гибкость и возможность масштабирования в будущем.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Структура платформы программного обеспечения

Платформа программного обеспечения для *Arduino* играет ключевую роль в разработке и реализации проектов. Она включает в себя инструменты, библиотеки и среду разработки, которые обеспечивают создание программ для управления аппаратным обеспечением. В данной главе мы рассмотрим структуру и архитектуру программной платформы *Arduino*.

2.1.1 *Arduino IDE* [8]. *Arduino Integrated Development Environment* (*IDE*) – это основное средство разработки, которое используется для написания, компиляции и загрузки кода на плату. Основные компоненты *Arduino IDE*:

1 Редактор кода. Предоставляет пользователю возможность писать и редактировать скетчи.

2 Компилятор. Преобразует написанный код в машинный язык, понятный микроконтроллеру.

3 Загрузчик. Передает скомпилированный код на *Arduino* через *USB*.

2.1.2 Библиотеки [8]. *Arduino* поддерживает множество библиотек, которые расширяют функциональность платформы. Библиотеки могут включать:

1 Стандартные библиотеки. Входят в состав *Arduino IDE* и обеспечивают базовые функции работы с датчиками, дисплеями и другими компонентами.

2 Пользовательские библиотеки. Разработаны сообществом и могут быть добавлены для поддержки специфических модулей или функций.

3 Библиотеки позволяют значительно упростить процесс программирования, предоставляя готовые функции для работы с аппаратным обеспечением.

2.2 Архитектура программного обеспечения

2.2.1 Стек программного обеспечения. Архитектура программного обеспечения *Arduino* можно представить в виде стека, состоящего из нескольких уровней:

1 Аппаратный уровень. Микроконтроллер и подключенные к нему устройства (датчики, модули).

2 Библиотеки и *API*. Набор библиотек, предоставляющих функции для работы с различными компонентами. Библиотеки скрывают сложность работы с аппаратным обеспечением.

3 Программный уровень. Основной код (скетчи), написанный пользователем, который использует функции из библиотек для управления устройствами.

2.2.2 Процесс выполнения программы [9]. Процесс выполнения программы включает следующие шаги:

1 Инициализация: При запуске программы выполняется функция *setup*(), где настраиваются порты и инициализируются переменные.

2 Циклическое выполнение: После инициализации программа переходит в бесконечный цикл *loop*(), в котором выполняется логика работы системы. Здесь происходит считывание данных с датчиков, обработка их и управление исполнительными устройствами.

2.2.3 Поддержка многопоточности [10]. Хотя Arduino не поддерживает полноценную многозадачность, использование прерываний и таймеров позволяет реализовать некоторые элементы параллельного выполнения задач. Это важно для обеспечения быстрого реагирования на события, такие как срабатывание датчиков безопасности.

2.3 История, версии и достоинства *Arduino IDE*

Arduino Integrated Development Environment (*IDE*) [8] была создана одновременно с платформой *Arduino* в 2005 году. Основная цель разработки *IDE* заключалась в упрощении процесса написания и загрузки кода на платы *Arduino*, чтобы сделать программирование доступным для широкой аудитории, включая новичков в электронике и программировании.

С момента своего запуска *Arduino IDE* активно развивалась, добавляя новые функции и улучшения. Сообщество разработчиков постоянно вносило вклад в её развитие, что привело к созданию множества библиотек и примеров, доступных для пользователей. В 2013 году была выпущена версия 1.0, которая закрепила основные функции, а последующие версии добавили поддержку новых плат и улучшили интерфейс.

2.3.1 Версии *Arduino IDE* [11]. *Arduino IDE* имеет несколько версий, каждая из которых предлагает уникальные функции и улучшения:

1 *Arduino IDE 1.x*. Это самая распространенная версия, которая поддерживает большинство плат *Arduino* и предоставляет основной функционал для написания и загрузки кода. Основные версии 1.x включают улучшения в интерфейсе и добавление новых библиотек.

2 *Arduino Web Editor*. Облачная версия *IDE*, которая позволяет писать и сохранять код в интернет-браузере. Это удобно для работы с различными устройствами без необходимости устанавливать программное обеспечение на каждое устройство.

3 *Arduino Pro IDE*. Эта версия была разработана для более опытных пользователей и предлагает расширенные функции, такие как поддержка многозадачности, отладка и интеграция с системами управления версиями.

4 *Arduino CLI*. Командный интерфейс для работы с *Arduino*, который позволяет разработчикам автоматически компилировать и загружать код с

помощью командной строки, что удобно для автоматизации процессов и интеграции с другими инструментами.

2.3.2 Достоинства *Arduino IDE* [12]. *Arduino IDE* обладает следующим рядом достоинтсв:

1 Простота и доступность. *Arduino IDE* имеет интуитивно понятный интерфейс, который упрощает процесс написания кода и его загрузки на платы. Это делает платформу доступной для пользователей с разным уровнем подготовки.

2 Обширная библиотека. *IDE* поддерживает множество библиотек, которые предоставляют готовые функции для работы с различными компонентами, такими как датчики, дисплеи и модули связи. Это значительно ускоряет процесс разработки.

3 Поддержка сообщества. Активное сообщество разработчиков предлагает огромное количество примеров, учебников и форумов, где можно получить помощь и советы. Это облегчает процесс обучения и решение возникающих проблем.

4 Кроссплатформенность. *Arduino IDE* доступна для различных операционных систем, включая *Windows*, *macOS* и *Linux*, что позволяет пользователям работать в удобной для них среде.

5 Возможности для расширения. *IDE* позволяет пользователям добавлять свои библиотеки и инструменты, что делает платформу гибкой и настраиваемой под конкретные нужды.

2.4 Вывод

Раздел подчеркивает важность интегрированной среды разработки (*IDE*) и её архитектуры для успешного создания проектов. *Arduino IDE* является основным инструментом, который позволяет разработчикам писать, компилировать и загружать код на микроконтроллер. Структура *IDE*, включающая редактор кода, компилятор и загрузчик, обеспечивает удобный и эффективный процесс разработки. Библиотеки, как стандартные, так и пользовательские, значительно расширяют функциональность платформы, позволяя разработчикам сосредоточиться на логике приложений, а не на низкоуровневом управлении аппаратным обеспечением. Стек программного обеспечения, состоящий из аппаратного уровня, библиотек и пользовательского кода, демонстрирует организованную архитектуру, которая упрощает взаимодействие с устройствами. История *Arduino IDE* свидетельствует о её постоянном развитии и адаптации к потребностям пользователей, включая создание облачной версии и профессиональных инструментов для более опытных разработчиков. Достоинства *IDE*, такие как простота использования, доступность обширных библиотек и поддержка активного сообщества, делают её привлекательной для новичков и профессионалов. В целом, платформа программного обеспечения *Arduino*, благодаря своей структуре и возможностям, предоставляет мощные

инструменты для разработки инновационных проектов, подходящих для различных областей применения, включая автоматизацию, *IoT* и образовательные инициативы.

Раздел подчеркивает важность интегрированной среды разработки (*IDE*) и её архитектуры для успешного создания проектов. *Arduino IDE* является основным инструментом, который позволяет разработчикам писать, компилировать и загружать код на микроконтроллер. Структура *IDE*, включающая редактор кода, компилятор и загрузчик, обеспечивает удобный и эффективный процесс разработки. Библиотеки, как стандартные, так и пользовательские, значительно расширяют функциональность платформы, позволяя разработчикам сосредоточиться на логике приложений, а не на низкоуровневом управлении аппаратным обеспечением. Стек программного обеспечения, состоящий из аппаратного уровня, библиотек и пользовательского кода, демонстрирует организованную архитектуру, которая упрощает взаимодействие с устройствами. История *Arduino IDE* свидетельствует о её постоянном развитии и адаптации к потребностям пользователей, включая создание облачной версии и профессиональных инструментов для более опытных разработчиков. Достоинства *IDE*, такие как простота использования, доступность обширных библиотек и поддержка активного сообщества, делают её привлекательной для новичков и профессионалов. В целом, платформа программного обеспечения *Arduino*, благодаря своей структуре и возможностям, предоставляет мощные инструменты для разработки инновационных проектов, подходящих для различных областей применения, включая автоматизацию, *IoT* и образовательные инициативы.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Разработка устройства контроллера безопасности открытия купола удаленной обсерватории на базе *Arduino* является важным шагом для повышения уровня автоматизации и безопасности в астрономических исследованиях. В этом разделе будут рассмотрены ключевые аспекты, обосновывающие необходимость и целесообразность данной разработки.

3.1 Обоснование необходимости разработки

С увеличением интереса к астрономии и доступностью астрономических данных, растёт потребность в автоматизированных системах, которые могут эффективно управлять обсерваториями. Автоматизация процессов открытия и закрытия купола обсерватории не только повышает эффективность, но и минимизирует риски, связанные с ручным управлением. Ошибки или задержки в этих процессах могут привести к повреждению дорогостоящего оборудования и создать опасные ситуации для пользователей.

Кроме того, с ростом числа частных и образовательных обсерваторий возрастает необходимость в системах, которые легко интегрируются и настраиваются под специфические требования. Современные обсерватории требуют решений, которые обеспечивают безопасность и могут адаптироваться к изменениям, таким как изменение погодных условий или необходимость быстрого реагирования на внешние факторы. Например, чрезвычайные ситуации, такие как сильный ветер или изменившаяся видимость, требуют мгновенного реагирования, чтобы избежать повреждений оборудования и обеспечить безопасность наблюдателей.

3.2 Технологии программирования

Для решения поставленных задач в разработке контроллера будут использованы языки программирования, основанные на *C*/*C*++, которые являются основными для платформы *Arduino*. Эти языки предоставляют множество преимуществ:

1 Простота и доступность: Синтаксис языков позволяет быстро обучаться и разрабатывать программы, что особенно важно для пользователей с различным уровнем подготовки. Это открывает возможности для широкого круга людей, включая студентов и любителей астрономии, принимать участие в разработке и эксплуатации обсерваторий.

2 Поддержка библиотек. Использование стандартных и пользовательских библиотек значительно упрощает разработку. Библиотеки предоставляют готовые функции для работы с датчиками, модулями связи и другими компонентами, что сокращает время, необходимое для реализации функционала и позволяет сосредоточиться на решении более сложных задач.

3 Обработка данных в реальном времени. Возможности языка

позволяют обрабатывать данные с датчиков и выполнять действия на основе полученных значений. Это критично для обеспечения безопасности, так как система должна мгновенно реагировать на изменения, предотвращая возможные аварийные ситуации.

3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением

Архитектура вычислительной системы *Arduino* тесно связана с разрабатываемым программным обеспечением. Она включает несколько уровней:

1 Аппаратный уровень: Микроконтроллеры, такие как *ATmega328* и *ATmega2560*, обеспечивают необходимую вычислительную мощность для обработки данных с датчиков и управления исполнительными механизмами. Эти микроконтроллеры обладают достаточной производительностью для выполнения всех необходимых задач, включая обработку многоканальных сигналов от датчиков и управление сервоприводами.

2 Библиотеки и *API*: Библиотеки, доступные в среде *Arduino IDE*, упрощают взаимодействие с аппаратным обеспечением, скрывая сложности низкоуровневого программирования. Это позволяет разработчику сосредоточиться на логике приложения, а не на технических аспектах, таких как регистрация данных или управление прерываниями.

3 Программный уровень: Пользовательский код (скетчи) строится на основе функций из библиотек, что обеспечивает простоту и гибкость в создании программ. Процесс выполнения программы, начиная с инициализации в функции *setup*() и заканчивая циклическим выполнением в *loop*(), позволяет эффективно управлять состоянием системы и реагировать на изменения в окружающей среде.

3.4 Система мониторинга

Наличие системы мониторинга состояния купола и окружающих условий является важным аспектом для предотвращения аварий и повреждений. Использование датчиков для отслеживания таких параметров, как положение купола, наличие препятствий и погодные условия, позволяет системе оперативно реагировать на изменения внешней среды.

Программное обеспечение будет обрабатывать данные в реальном времени, что позволяет избежать нежелательных ситуаций. Например, если датчик обнаруживает препятствие на пути открывающегося купола, система может немедленно остановить движение, предотвращая повреждения как купола, так и препятствия. Интеграция дополнительных датчиков, таких как метеостанции, позволит учитывать погодные условия, что особенно важно для безопасности работы обсерватории. Например, система может автоматически закрыть купол при ухудшении погодных условий, минимизируя риск повреждений.

3.5 Современные требования к автоматизации и безопасности

Создание контроллера безопасности открытия купола удаленной обсерватории на базе *Arduino* отвечает современным требованиям к автоматизации и безопасности в астрономии. Автоматизированные решения способны значительно повысить эффективность работы обсерватории, минимизируя необходимость в ручном управлении и снижая риск человеческой ошибки. Это особенно актуально в условиях удалённых обсерваторий, где физическое присутствие операторов может быть ограничено, и требуется высокая степень автономности системы.

Современные обсерватории также требуют интеграции с новыми технологиями, такими как Интернет вещей (*IoT*). Возможность удаленного доступа к системе через интернет позволяет операторам управлять куполом и получать уведомления о состоянии системы из любой точки мира, что существенно упрощает процесс наблюдений и управления.

3.6 Удобство для пользователей

Создание безопасного и надежного контроллера также делает эксплуатацию обсерватории более удобной для пользователей. Упрощение процессов управления и мониторинга позволяет астрономам сосредоточиться на исследовательской деятельности, а не на технических аспектах работы с оборудованием.

Доступность интерфейса и возможность настройки под индивидуальные потребности пользователей делают систему более интуитивной и дружелюбной. Обучающие материалы и поддержка со стороны сообщества помогут пользователям быстро освоить систему и эффективно её использовать.

3.7 Вывод

Таким образом, разработка контроллера безопасности открытия купола удаленной обсерватории на базе *Arduino* представляет собой ответ на современные вызовы в области астрономии и автоматизации. Простота, доступность и надежность платформы *Arduino*, а также возможность интеграции с современными технологиями делают её идеальным выбором для реализации данного проекта. Это решение не только повысит безопасность и эффективность работы обсерватории, но и обеспечит комфортные условия для пользователей, способствуя дальнейшему развитию астрономических исследований и популяризации астрономии среди широкой аудитории.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

Контроллер безопасности открытия купола удаленной обсерватории должен обеспечивать надежное и безопасное управление процессом открытия и закрытия купола на основе показаний различных датчиков. Ниже представлены основные функциональные возможности, которые будут реализованы в устройстве.

4.1 Датчики и их функциональность

4.1.1 Датчики влажности. В системе будет установлено два датчика влажности, которые измеряют уровень влажности в окружающей среде. Эти датчики играют критическую роль в предотвращении попадания влаги внутрь обсерватории. Если уровень влажности превышает заданный порог, контроллер автоматически не позволит открыть купол, что защитит оборудование от возможных повреждений.

4.1.2 Датчики давления. Два датчика давления будут измерять атмосферное давление. Снижение давления может указывать на приближение шторма или других плохих погодных условий. В случае, если давление опускается ниже установленного порога, контроллер также не разрешит открытие купола, что обеспечит дополнительную защиту для оборудования.

4.1.3 Датчики скорости ветра. Два датчика скорости ветра будут определять текущую скорость ветра. Если ветер превышает безопасный уровень, открытие купола будет запрещено. Это предотвратит повреждения, которые могут возникнуть из-за сильного ветра, когда купол находится в открытом состоянии.

4.1.4 Датчики закрытия купола (герконы). Два геркона будут контролировать положение купола (открыт/закрыт). Эти датчики необходимы для обеспечения безопасности при проведении операций с куполом. Они обеспечивают точность определения состояния купола и предотвращают случайные открытия в неподходящее время.

4.2 Обработка ошибок

Контроллер должен иметь возможность обнаруживать и обрабатывать ошибки, связанные с работой датчиков:

4.2.1 Отключение датчика. Программа будет отслеживать состояние каждого датчика. Если какой-либо датчик отключается или перестает корректно функционировать, контроллер должен уведомить пользователя с помощью светодиода. Это позволит быстро реагировать на проблемы и

минимизировать риск аварийных ситуаций.

4.2.2 Аномалии в показаниях. Если показания датчиков сильно разнятся между собой, контроллер должен информировать о неисправности одного из датчиков. Это может быть реализовано через систему сравнения показаний и выдачи предупреждений при обнаружении значительных расхождений, что также поможет в поддержании надежности системы.

4.3 Оповещение пользователя

Контроллер предполагает использование светодиодов для оповещения пользователя о его состоянии. Графический интерфейс пользователя представлен в приложении Д.

4.3.1 Светодиод для оповещения о состоянии датчиков. При обнаружении ошибки (отключение или аномалия) светодиод будет подавать сигнал, привлекая внимание пользователя. Это позволит быстро реагировать на возникшие проблемы и минимизировать время простоя системы.

4.3.2 Светодиод для разрешения открытия купола. Если все условия для открытия купола выполнены (показания датчиков в норме), светодиод будет светиться зеленым цветом, сигнализируя о том, что купол можно открывать. В противном случае светодиод не будет гореть, что станет индикатором для пользователя о необходимости проверки состояния системы.

4.4 Логика принятия решения

Программа будет принимать решение о возможности открытия купола на основе показаний всех датчиков. Логика будет работать следующим образом:

1 Считывание данных с. Программа регулярно считывает данные с датчиков влажности, давления и скорости ветра.

2 Проверка датчиков на ошибки. Если ошибки датчиков присутствуют, купол открывать запрещено. Это включает как физические неисправности, так и аномальные показания.

3 Проверка показаний на соответствие заданным критериям:

– Влажность больше порогового значения;

– Давление больше порогового значения;

– Скорость ветра меньше порогового значение.

4 Активирование сигнализации. Если все условия выполнены, активируется светодиод "можно открывать", и купол открывается. В противном случае светодиод "можно открывать" не активирован, сигнализируя о том, что необходимо дождаться улучшения условий.

4.5 Описание функциональной схемы программы

Функциональная схема программы представлена в приложении В. Она описывает процесс работы системы, связанной с контролем и управлением куполом.

1 Инициализация датчиков. На первом этапе происходит активация всех необходимых датчиков, которые будут использоваться для мониторинга состояния окружающей среды и оборудования.

2 Считывание данных с датчиков. После инициализации система начинает собирать данные с датчиков. Это могут быть параметры, такие как температура, влажность, давление или другие показатели, критически важные для функционирования купола.

3 Обработка ошибок. На этом этапе система проверяет корректность полученных данных. Если обнаруживаются ошибки или аномалии (например, выход данных за допустимые пределы), система регистрирует их и принимает меры для их устранения.

4 Принятие решения об открытии купола: Основываясь на обработанных данных и результатах проверки, система принимает решение о целесообразности открытия купола. Это может зависеть от погодных условий, состояния оборудования и других факторов.

5 Оповещение пользователя. В конце процесса система информирует пользователя о принятом решении. Это может быть уведомление о том, что купол открыт или закрыт, а также предоставление информации о текущих показателях и состоянии системы.

4.6 Описание блок схемы алгоритма программы

Блок схема алгоритма программы представлена в приложении Г. Блок-схема состоит из пяти ключевых функций:

1 *setup*. Эта функция выполняет начальную настройку. Она инициализирует последовательный вывод, настраивает датчики *DHT*, *BMP*, анемометр и реле, а также устанавливает режим работы светодиода.

2 *loop*. Основной цикл программы. Он периодически читает данные с датчиков, управляет светодиодами в зависимости от состояния ошибок и проверяет, можно ли открыть крышу, выводя соответствующие сообщения в последовательный порт.

3 *manageLED*. Управляет состоянием светодиода. В зависимости от состояния ошибки:

– Выключает светодиод, если ошибок нет;

– Мигает светодиодом, если есть ошибка (мигание происходит 10 раз);

– Включает светодиод постоянно, если ошибка критическая.

4 *canOpenRoof*. Проверяет, можно ли открыть крышу. Возвращает *false*, если есть данные *NaN* или если уровень ошибок превышает заданные пороги. Также проверяет значения данных с датчиков на соответствие заданным условиям (например, температура, давление и скорость ветра).

5 *setupDHTSensors*. Эта функция инициализирует *DHT*-датчики и настраивает пин для светодиода.

6 *readDHTSensors*. Эта функция считывает данные с двух *DHT*-датчиков и возвращает структуру *response*, содержащую информацию о состоянии.

7 *setupBMPSensors*. Эта функция инициализирует *BMP*-датчики и настраивает необходимые пины.

8 *readBMPSensors*. Эта функция считывает данные с двух *BMP*-датчиков и возвращает структуру *response*, содержащую информацию о состоянии.

9 *setupAnemometer*. Эта функция инициализирует анемометр и настраивает необходимые пины.

10 *checkAnemometerConnection*. Эта функция проверяет соединение с анемометром.

11 *readAnemometers*. Эта функция считывает данные с двух анемометров и возвращает структуру *response*, содержащую информацию о состоянии.

12 *setupReedSwitches*. Эта функция инициализирует реле и настраивает необходимые пины.

13 *readReedSwitches*. Эта функция считывает состояние реле и возвращает структуру *response*, содержащую информацию о состоянии.

4.7 Вывод

Разработка контроллера безопасности открытия купола удаленной обсерватории на базе *Arduino* включает в себя детальное проектирование функциональных возможностей, основанных на использовании различных датчиков для мониторинга окружающей среды. Эти функции не только обеспечивают безопасность и защиту оборудования, но и делают систему более надежной и удобной в эксплуатации.

Эффективная обработка ошибок и система оповещения позволяют пользователям быстро реагировать на любые неполадки, что критически важно для работы обсерватории. Логика принятия решений, основанная на данных с датчиков, гарантирует, что купол будет открываться только в безопасных условиях, что значительно повышает уровень автоматизации и защищенности.

В результате, проектирование функциональных возможностей программы создает надежную и безопасную систему, способствующую эффективному управлению обсерваторией и обеспечению ее долгосрочной эксплуатации.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

5.1 Подбор датчиков

Для реализации контроллера безопасности открытия купола удаленной обсерватории был произведен подбор нескольких типов датчиков, которые обеспечивают надежный мониторинг окружающей среды. В проекте используются следующие датчики: два датчика *DHT22*, два датчика *BMP180*, два геркона (датчики открытия дверей) и два цифровых анемометра.

1 *DHT22* (рисунок 5.1) [13]. *DHT22* – это цифровой датчик температуры и влажности, который обеспечивает высокую точность измерений. Он способен работать в диапазоне температур от -40°C до +80°C и влажности от 0% до 100%. Преимущества использования *DHT22* включают:

а Высокая точность: датчик имеет небольшую погрешность, что позволяет точно оценивать условия окружающей среды.

б Цифровой выход: легкость интеграции с платформой *Arduino* благодаря цифровому интерфейсу.

в Низкое энергопотребление: подходит для работы в удаленных системах.

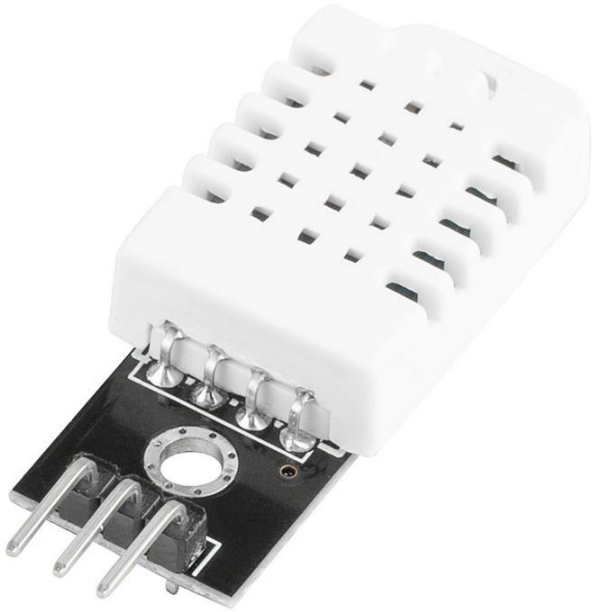


Рисунок 5.1 – Датчик *DHT22*

2 *BMP180* (рисунок 5.2). *BMP180* – это барометрический датчик, который
измеряет атмосферное давление и температуру. Он широко используется в метеорологии и авиации. Основные характеристики *BMP180*:

а Высокая точность давления: датчик способен измерять давление с точностью до 0,01 гПа.

б Широкий диапазон: рабочий диапазон давления от 300 до 1100 гПа,

что позволяет использовать его в различных климатических условиях.

в Совместимость: поддерживает интерфейсы *I2C* и *SPI*, что делает его совместимым с *Arduino*.

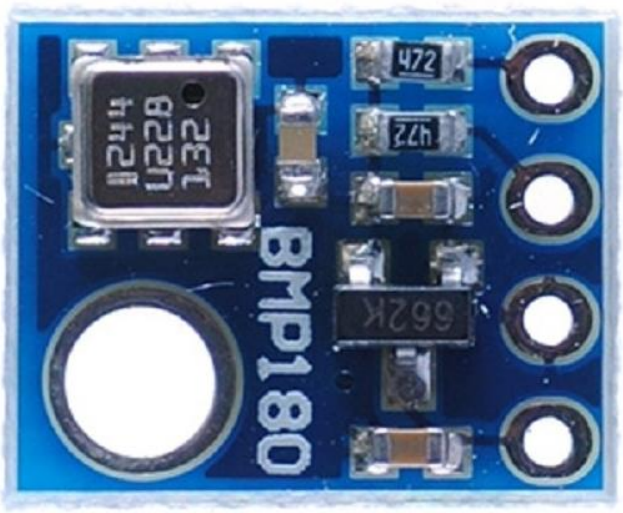


Рисунок 5.2 – Датчик *BMP180*

3 Герконы (рисунок 5.3). Герконы – это механические датчики, которые используются для определения положения и состояния купола. Они реагируют на движение и могут использоваться для подтверждения открытого или закрытого состояния. Преимущества герконов:

а Надежность: простая конструкция и высокая устойчивость к внешним воздействиям.

б Быстрая реакция: мгновенно фиксируют изменения в положении купола.

в Долговечность: долгий срок службы без необходимости замены.

Рисунок 5.3 – Геркон

4 Анемометры. Анемометры измеряют скорость потока ветра, что является критически важным параметром для определения безопасности открытия купола. В проекте могут быть использованы как механические (рисунок 5.4), так и ультразвуковые (рисунок 5.5) анемометры. Основные преимущества анемометров:

а Точность измерений: позволяют получать актуальные данные о скорости ветра.

б Разнообразие типов: возможность выбора между различными типами

анемометров в зависимости от требований проекта.

в Интеграция: легкость подключения к *Arduino* и обработки данных.

Рисунок 5.4 – Механический анемометр

Рисунок 5.5 – Ультразвуковой анемометр

5.2 Подбор платы *Arduino*

Для реализации контроллера безопасности открытия купола удаленной обсерватории была выбрана плата *Arduino Mega 2560*. Этот выбор обусловлен рядом факторов, касающихся функциональности, возможностей подключения и производительности. Преимущества *Arduino Mega 2560*:

1 Высокая производительность. *Arduino Mega 2560* оснащена 16 МГц процессором *ATmega2560*, который обеспечивает достаточную

вычислительную мощность для обработки данных с нескольких датчиков одновременно.

2 Большое количество входов и выходов. Плата имеет 54 цифровых входа/выхода, из которых 15 могут использоваться как ШИМ-выходы, а также 16 аналоговых входов. Это позволяет подключать множество датчиков и исполнительных механизмов без необходимости использования дополнительных расширителей.

3 Объем памяти. *Arduino Mega 2560* располагает 256 КБ флэш-памяти для хранения программного кода, что значительно превышает объем памяти других моделей *Arduino*, таких как *Uno*. Это позволяет реализовывать более сложные алгоритмы и хранить большие объемы данных.

4 Поддержка различных интерфейсов. Плата поддерживает интерфейсы *I2C* и *SPI*, что упрощает подключение и интеграцию различных датчиков, таких как *BMP180* и *DHT22*, а также других периферийных устройств.

5 Расширенные возможности. *Arduino Mega 2560* совместима с множеством библиотек и модулей, что позволяет легко расширять функциональность проекта и интегрировать новые компоненты по мере необходимости.

5.3 Подбор других компонентов

В связи с некоторыми особенностями подключения некоторых датчиков системы были выбраны два мультиплексора: *74HC4052* и *74HC4051*. А также стягивающие резисторы номиналом 10 кОм.

5.3.1 Мультиплексор *74HC4052* (рисунок 5.6) [15]. *74HC4052* – это аналоговый мультиплексор/демультиплексор, который позволяет подключать два канала аналоговых сигналов. Он идеально подходит для подключения датчиков, таких как *BMP180*, которые требуют обработки аналоговых и цифровых сигналов.

Рисунок 5.6 - Мультиплексор 74HC4052

5.3.2 Мультиплексор *74HC4051* (рисунок 5.7) [16]. *74HC4051* – это 8-канальный аналоговый мультиплексор, который позволяет подключать до

восьми различных аналоговых сигналов к одному входу Arduino. Этот мультиплексор будет использован для подключения анемометров, что является ключевым аспектом для мониторинга скорости ветра.

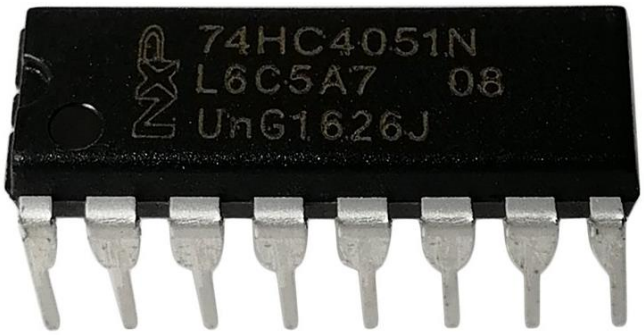


Рисунок 5.7 – Мультиплексор *74HC4051*

5.4 Способ вывода информации о состоянии контроллера пользователю. Подключение светодиодов к *Arduino*.

Графический интерфейс пользователя предоставлен в приложении Г. На листе 1 изображены светодиоды красного и зеленого цветов, которые подключены к контроллеру для информирования пользователя о его состоянии. Всего подключено пять светодиодов: 4 красного цвета и 1 зеленого цвета. Красные светодиоды подключаются через резисторы в пины с 25 по 28 и отвечают за ошибки датчиков открытия купола (25), анемометров (26), давления (27) и влажности (28). Зеленый светодиод информирует нас о том, можно ли открывать купол и подключается в 24 пин. Вторая ножка светодиода заземляется.

Для анемометров, датчиков влажности и давления красные светодиоды могут моргать, гореть постоянно или отключены. Моргание обозначает то, что один из датчиков неисправен, например нет соединения или показания двух датчиков сильно различаются. Если светодиод постоянно горит, то это говорит нам о том, что оба датчика неисправны или отключены. Если светодиод отключен, то ошибок датчиков не обнаружено.

Для красного светодиода датчиков открытия купола есть два состояния: постоянно горит или отключен. Если светодиод горит, то купол открыт и датчики замкнуты. Моргание светодиода сообщает нам о том, что датчики имеют разное состояние, следовательно один из датчиков неисправен или купол закрылся не до конца. Если светодиод отключен, то купол закрыт и датчики разомкнуты.

Зеленый светодиод сообщает нам о том, можно ли открывать купол. Если он горит, значит контроллер считает, что купол можно открыть. Если светодиод отключен, то открывать купол запрещено.

5.5 Подключение датчиков к *Arduino*

Правильное подключение датчиков к плате *Arduino* является одним из ключевых этапов разработки контроллера безопасности открытия купола. В данном подразделе будет рассмотрен процесс подключения выбранных датчиков, таких как *DHT22*, *BMP180*, герконы и анемометры, а также применение мультиплексоров *74HC4052* и *74HC4051* для оптимизации использования входов на *Arduino Mega 2560*.

Эффективная схема подключения позволит обеспечить надежное считывание данных с датчиков и их быструю обработку. Важно учитывать спецификации каждого датчика и особенности работы мультиплексоров, чтобы избежать возможных ошибок и обеспечить корректное функционирование всей системы.

5.5.1 Подключение *DHT22*. *DHT22* имеет 3 пина: *VDD, DATA* и *GND*. Для подключения питания использовался пин *VCC* на *Arduino*, так как он выдаёт нужные 5 В, заземление подключено на пин *GND* на *Arduino*, информационные пины подключены к цифровым пинам 22 и 23 (*AD0* и *AD1*).

5.5.2 Подключение *BMP180*. *BMP180* имеет 4 пина: *SCL, SDA, VDDIO* и *GND* (скрыт в *Proteus*). Данный датчик имеет протокол *I2C*, что позволяет подключить несколько датчиков к одним и тем же пинам на контроллере и обращаться по адресу к каждому датчику, но производитель даёт им одинаковый адрес *0x77*, что приводит к конфликту. Для выхода из этой ситуации был использован демультиплексор *74HC4052*.

Подключение самих пинов датчиков следующее: VDDIO подключен к внешнему питанию 3.3 вольта, *SCL* подключен к 1 (*Y0*) пину демультиплексора, а *SDA* подключен к 12 (*X0*) пину демультиплексора. Второй датчик подключен аналогично, с использованием 5 (*Y1*) и 14 (*X1*) пинов демультиплексора.

Подключение демультиплексора: управляющий пин 10 (*A*) подключен к 19 (*RX1*) пину на *Arduino*, управляющие пины 9 (B) и 6 (*INH*) заземлены, так как не используются. Выходной пин 13 (*X*) подключен к 20 (*SDA*) пину, а выходной пин 3 (*Y*) подключен к 21 (*SCL*) пину на *Arduino*.

5.5.3 Подключение анемометров. Анемометр имеет 2 пина: *VCC* и *GND*. Так как имеющиеся анемометры работают на основе тахометра, нам нужно считать количество вращений (импульсов), поступивших от анемометра. Для этого была выбрана библиотека *FreqCount*, она может считать частоту сигнала, который поступает на пин, но имеет ограничение в виде 47 пина, считывание происходит только с него, поэтому используется мультиплексор и считывание показаний с анемометров будет происходить поочередно.

Подключение самих пинов анемометров следующее: *VCC* подключен к внешнему питанию, *GND* подключен к мультиплексору в пин 13 (*X0*) и к земле через стягивающий резистор номиналом 10 кОм для предотвращения

неопределенных сигналов. Второй анемометр подключен аналогично, с использованием 14 (*X1*) мультиплексора.

Подключение мультиплексора: управляющий пин 10 (*A*) подключен к 2 (*INT4*) пину на *Arduino*, управляющие пины 10 (*B*), 9 (*C*) и 6 (*INH*) заземлены, так как не используются. Выходной пин 3 (*X*) подключен к 47 (*T5*) пину на *Arduino*.

5.5.4 Подключение герконов. Герконы имеют 2 пина, у них нет конкретного назначения, так как геркон может использоваться в различных целях. В данном проекте один пин геркона заземлен, а второй подключен к 18 (*TX1*) пину на *Arduino*. Второй геркон подключен аналогично, с использованием 3 (*INT5*) пина на контроллере.

5.6 Прошивка контроллера

Для написания прошивки контроллера использовался язык *C++*, он является основным языком для Arduino и большинство плат, библиотек и датчиков поддерживают именно его.

5.6.1 Используемые библиотеки. Для считывания показаний с датчиков были использованы следующие библиотеки:

1 *DHT.h* [17] данная библиотека создана разработчиками датчиков *DHT* и служит для взаимодействия с ними. С помощью данной библиотеки мы можем инициализировать датчик, считать температуру, влажность и т.д.

2 *SFE_BMP180.h [18],* данная библиотека служит для взаимодействия с датчиками *BMP180*. С помощью данной библиотеки мы можем инициализировать датчик, начать измерения, считать полученные при измерениях данные и т.д.

3 *FreqCount.h* [19], данная библиотека служит для считывания частоты с определенных пинов. В данном контроллере она используется для считывания частоты вращения анемометров. С помощью неё мы можем начать подсчет частоты, закончить его, считать данные и т.д.

4 *Arduino.h [20],* данная библиотека является стандартной для платформы *Arduino,* она предоставляет базовое взаимодействие с платой, например инициализацию пинов, считывание данных с них, передачу данных через пины, передачу данных через Serial порт и т.д.

5.6.2 Инициализация датчиков. Для инициализации датчиков использовались вышеописанные библиотеки и пины.

Для инициализации датчиков *DHT* в прошивке используется метод *begin*() класса *DHT* из библиотеки *DHT.h*. Конструктор класса принимает пин (в данном контроллере это пины 22 и 23, устанавливаются в *INPUT*) к которому подключен датчик и тип датчика, в данном контроллере это *DHT22*.

Для инициализации датчиков *BMP* в прошивке используется метод *begin*() класса *SFE_BMP180* из библиотеки *SFE_BMP180*. Мы не должны

передавать в класс или метод пин подключения, так как данный датчик использует протокол *I2C*. Также инициализируется 19 пин в режиме *OUTPUT* для выбора датчика.

Для инициализации анемометров в прошивке инициализируется 3 пин в режиме *OUTPUT* для выбора анемометра, а также 47 пин для считывания данных в режиме *INPUT*. Библиотека *FreqCount.h* сама инициализирует 47 пин, поэтому в коде не нужно явно это прописывать.

Для инициализации герконов в прошивке инициализируются пины 3 и 18 в режиме *INPUT_PULLUP*, так как второй конец геркона заземлен и нам нужно инвертировать значение.

5.6.3 Переключение мультиплексоров. Для того, чтобы выбирать датчик, с которого нужно считать значение, необходимо переключить мультиплексор. Это осуществляется с помощью подачи или отключения напряжения на контролирующие пины мультиплексора. Например для выбора одного из датчиков BMP мы подаем или снимаем напряжение с 19 пина с помощью команд *digitalWrite*(*BMP_SELECT_PIN*, *HIGH*) и *digitalWrite*(*BMP_SELECT_PIN*, *LOW*) соответственно.

5.6.4 Считывание данных с датчиков. С различных датчиков прошивка считывает данные разными способами.

Для считывания влажности с датчика *DHT*, используется метод *readHumidity*() класса *DHT* библиотеки *DHT.h*, данный метод возвращает уже готовые данные, которые не нужно преобразовывать.

Для считывания давления с датчика *BMP* необходимо сначала начать его измерение, для этого используется метод *startPressure*() класса *SFE_BMP180* библиотеки *SFE_BMP180*. После измерения мы можем получить давление с помощью метода *getPressure*() того же класса и библиотеки, данный метод принимает 2 параметра: первый это ссылка на переменную давления, а второй это ссылка на переменную температуры.

Для считывания данных с анемометров используется несколько методов библиотеки *FreqCount.h*. Для начала нужно начать измерение, используя метод *begin*(), который принимает время измерения в миллисекундах. После этого нужно проверить, закончила ли библиотека измерения, для этого используется метод *available*(), который возвращает истину, если измерение окончено и ложь в противном случае. Затем необходимо прекратить измерения с помощью метода *end*(), чтобы подготовить библиотеку для следующего измерения. После остановки измерений данные всё еще хранятся в библиотеке и мы можем их считать с помощью метода *read*().

Для считывания данных с герконов мы используем метод *digitalRead*(), она принимает пин с которого нужно считать значение и возвращает *HIGH* или *LOW*.

5.6.5 Обработка ошибок. Обработка ошибок проходит в несколько этапов для всех датчиков кроме герконов. Первый этап – это обработка

ошибок подключения датчиков, второй этап – это обработка аномалий данных.

Для обработки ошибок подключения датчика *DHT* используется метод *readHumidity*(), который возвращает *NAN*, если не может считать данные с датчика.

Для обработки ошибок подключения датчика *BMP* используется два метода. Первый метод это *startPressure*(), который возвращает ложь, если не удалось начать измерение. Второй метод это *getPressure*(), который возвращает ложь, если не удалось считать измерение.

Для обработки ошибок подключения анемометров начинается измерение частоты анемометра, если измерение равно 0, то либо датчик не работает, либо на улице нет ветра.

Для определения аномалии данных используется формула, которая рассчитывает разницу в процентах между показаниями двух датчиков. После этого сравнивается результат с заданным порогом и если он его превышает, то аномалия считается зафиксированной.

Для определения ошибок, которые связаны с герконами, прошивка сравнивает показания двух датчиков, если они не равны, то выдает ошибку

5.6.6 Структура возврата данных из датчиков. Весь функционал считывания значений и определения ошибок вынесен в отдельные функции, которые возвращают структуру *response*, которая содержит данные первого датчика (*data1*), данные второго датчика (*data2*), погрешность измерений датчиков (*ErrorRate*), наличие ошибки (*ErrorState*), пин светодиода, который отвечает за ошибку данного датчика (*ERR_LED_PIN*).

В данной структуре *ErrorState* может принимать значения 0, 1, 2 для датчиков влажности, давления, скорости ветра в зависимости от типа ошибки. Значение 0 говорит о том, что ошибки отсутствуют, значение 1 говорит о том, что один из датчиков отключен или даёт неверные показания, значение 2 говорит о том, что оба датчика неисправны. Для герконов данный параметр принимает такие же значения, но обозначения другие: 0 обозначает, что купол закрыт, 1 обозначает ошибку, 2 обозначает, что купол открыт.

5.6.7 Логика принятия решения об открытии купола. Для того, чтобы принять решение об открытии купола прошивка сначала собирает данные со всех датчиков, а затем обрабатывает данные. Если хотя бы в одной паре датчиков возникла ошибка, связанная с потерей соединения с двумя датчиками или аномалия в показаниях, то контроллер запретит открытие купола. Если ошибок не обнаружено, то прошивка возьмет одно из значений пары датчиков (если одно из них *NAN*, то возьмет второе) и проведет сравнение с пороговыми значениями. При удовлетворении всех условий, контроллер даст разрешение на открытие купола.

5.7 Вывод

Архитектура разрабатываемой программы контроллера безопасности открытия купола удаленной обсерватории представляет собой сложную, но в то же время гармоничную систему, в которой каждый компонент играет важную роль в обеспечении надежности и эффективности работы. При выборе датчиков особое внимание уделяется качеству и точности. Например, использование датчиков температуры и влажности *DHT22* в сочетании с барометром *BMP180* позволяет не только контролировать климатические условия внутри и снаружи купола, но и предсказывать возможные изменения погоды, что критически важно для астрономических наблюдений.

Надежные герконы и анемометры дополняют этот набор, обеспечивая мониторинг состояния открытых дверей и скорости ветра. Это позволяет системе автоматически реагировать на изменения окружающей среды, что особенно актуально в условиях удаленной обсерватории, где доступ к оборудованию может быть ограничен. Плата *Arduino Mega 2560* была выбрана благодаря своей высокой вычислительной мощности и способности работать с несколькими датчиками одновременно. Это делает ее идеальным решением для задач, требующих параллельной обработки данных.

Подключение и инициализация датчиков осуществляется с использованием мультиплексоров, что не только оптимизирует использование входных портов на *Arduino*, но и увеличивает надежность системы. Это позволяет избежать перегрузки и обеспечивает стабильное считывание данных даже в условиях, когда количество подключенных устройств велико. Программное обеспечение, разработанное на языке *C++*, играет ключевую роль в обработке информации, поступающей от датчиков. Оно включает алгоритмы для выявления ошибок и принятия решений о безопасности открытия купола, что минимизирует риски и предотвращает возможные аварии.

Вся архитектура системы направлена на создание надежной и безопасной платформы, способной эффективно функционировать в условиях удаленной обсерватории. Это не только улучшает наблюдательные процессы, но и открывает новые горизонты для дальнейших исследований в области астрономии. Внедрение такой системы автоматизации значительно снижает человеческий фактор, позволяя астрономам сосредоточиться на научных задачах и получать более качественные данные. Таким образом, проект становится важным шагом к улучшению методов наблюдения за небесными телами и расширению возможностей для научных открытий.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была разработана система контроллера безопасности открытия купола удаленной обсерватории на базе платформы *Arduino*. Проект продемонстрировал, как доступное аппаратное обеспечение и удобное программное обеспечение могут быть успешно интегрированы для решения актуальных задач в области астрономии и автоматизации.

Arduino, благодаря микроконтроллеру *ATmega2560*, обеспечила необходимую производительность для управления множеством датчиков, что является ключевым аспектом для обеспечения безопасности функционирования обсерватории. Простота использования и гибкость платформы позволили разработать надежное решение, которое отвечает современным требованиям и вызовам.

Важную роль в успешной реализации проекта сыграла интегрированная среда разработки (*IDE*) *Arduino*, которая обеспечила удобный процесс написания, компиляции и загрузки кода. Поддержка обширных библиотек и активного сообщества позволила сосредоточиться на логике приложения, минимизируя время на низкоуровневое управление аппаратным обеспечением.

Разработка системы, основанной на использовании различных датчиков для мониторинга окружающей среды, позволила создать надежный механизм, который обеспечивает безопасность открытия купола. Эффективная обработка ошибок и система оповещения позволили пользователям быстро реагировать на любые неполадки, что критически важно для работы обсерватории.

Архитектура программы была тщательно продумана, что позволило оптимизировать использование ресурсов *Arduino* и гарантировать надежное считывание данных. Использование высококачественных датчиков, таких как *DHT22* и *BMP180*, обеспечило точный мониторинг условий окружающей среды, что повышает уровень автоматизации и защищенности.

В результате, проект контроллера безопасности открытия купола удаленной обсерватории не только повысил безопасность и эффективность работы обсерватории, но и способствовал улучшению наблюдательных процессов. Это решение открывает новые перспективы для дальнейших исследований в области астрономии и популяризации астрономических знаний среди широкой аудитории.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Коллектив авторов Стандарт предприятия / Коллектив авторов. – Минск БГУИР, 2024. – 178 с.

[2] Arduino [Электронный ресурс]. – Режим доступа: https://www.arduino.cc/en/about.

[3] Аппаратное обеспечение платы Arduino Uno и назначение ее компонентов [Электронный ресурс]. – Режим доступа: https://microkontroller.ru/arduino-projects/apparatnoe-obespechenie-platy-arduino-uno-i-naznachenie-ee-komponentov/.

[4] Среда разработки Arduino [Электронный ресурс]. – Режим доступа: https://arduino.ru/Arduino_environment.

[5] История создания Arduino [Электронный ресурс]. – Режим доступа: https://www.drive2.ru/b/2520138/.

[6] Разновидности плат Arduino, а также про клоны, оригиналы и совместимость [Электронный ресурс]. – Режим доступа: https://robocraft.ru/arduino/1035.

[7] Плюсы Arduino и почему Arduino это круто? [Электронный ресурс]. – Режим доступа: https://pikabu.ru/story/plyusyi_arduino_i_pochemu_arduino_yeto_kruto_6485560.

[8] Arduino Software (IDE) [Электронный ресурс]. – Режим доступа: https://www.arduino.cc/en/Guide/Environment.

[9] Синтаксис и структура кода [Электронный ресурс]. – Режим доступа: https://alexgyver.ru/lessons/syntax/.

[10] Многозадачность в Arduino [Электронный ресурс]. – Режим доступа: https://alexgyver.ru/lessons/how-to-sketch.

[11] Arduino Software Release Notes [Электронный ресурс]. – Режим доступа: https://www.arduino.cc/en/software/ReleaseNotes.

[12] What are the pros and cons of using Arduino IDE vs. other development tools for embedded software? [Электронный ресурс]. – Режим доступа: https://www.linkedin.com/advice/0/what-pros-cons-using-arduino-ide-vs-other-development.

[13] DHT22 (DHT22 also named as AM2302) [Электронный ресурс]. – Режим доступа: https://s3-sa-east-1.amazonaws.com/multilogica-files/datasheets/DHT22.pdf.

[14] Обзор датчика давления BMP180 (BMP080) [Электронный ресурс]. – Режим доступа: https://robotchip.ru/obzor-datchika-davleniya-bmp180/.

[15] 74HC4052; 74HCT4052 Dual 4-channel analog multiplexer/demultiplexer [Электронный ресурс]. – Режим доступа: https://www.farnell.com/datasheets/2051461.pdf.

[16] 74HC4051; 74HCT4051 8-channel analog multiplexer/demultiplexer [Электронный ресурс]. – Режим доступа: https://assets.nexperia.com/documents /data-sheet/74HC_HCT4051.pdf.

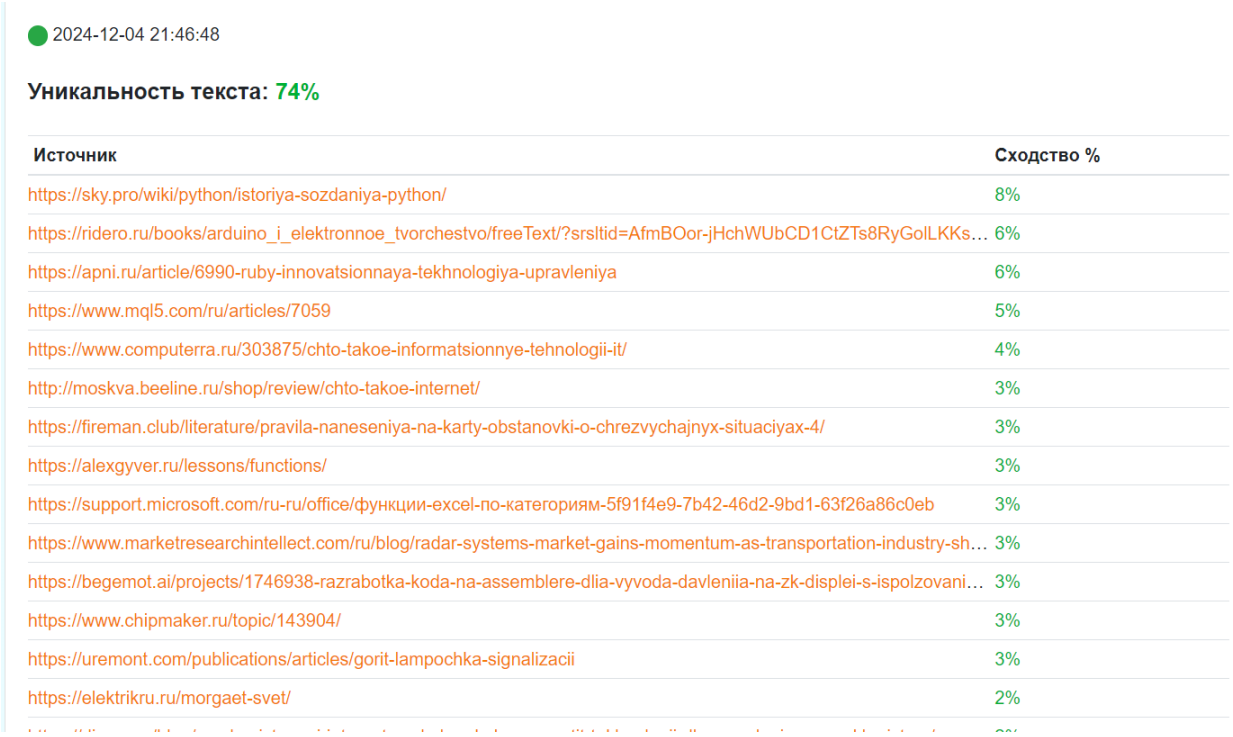
[17] Библиотека DHT.h [Электронный ресурс]. – Режим доступа: https://smdx.ru/blog/arduino/dht-h.

[18] BMP180_Breakout_Arduino_Library [Электронный ресурс]. – Режим доступа: https://github.com/sparkfun/BMP180_Breakout_Arduino_Library.

[19] FreqCount [Электронный ресурс]. – Режим доступа: https://docs.arduino.cc/libraries/freqcount/.

[20] Arduino Libraries [Электронный ресурс]. – Режим доступа: https://exiting-arduino.readthedocs.io/en/latest/arduino_libraries.html.

ПРИЛОЖЕНИЕ А
(обязательное)
Справка о проверке на заимствования



2024-12-04 21:46:48

Уникальность текста: 74%

Источник	Сходство %
https://sky.pro/wiki/python/istoriya-sozdaniya-python/	8%
https://ridero.ru/books/arduino_i_elektronnoe_tvorchestvo/freeText/?srsltid=AfmBOor-jHchWUbCD1CtZTs8RyGolLKKs…	6%
https://apni.ru/article/6990-ruby-innovatsionnaya-tekhnologiya-upravleniya	6%
https://www.mql5.com/ru/articles/7059	5%
https://www.computerra.ru/303875/chto-takoe-informatsionnye-tehnologii-it/	4%
http://moskva.beeline.ru/shop/review/chto-takoe-internet/	3%
https://fireman.club/literature/pravila-naneseniya-na-karty-obstanovki-o-chrezvychajnyx-situaciyax-4/	3%
https://alexgyver.ru/lessons/functions/	3%
https://support.microsoft.com/ru-ru/office/функции-excel-по-категориям-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb	3%
https://www.marketresearchintellect.com/ru/blog/radar-systems-market-gains-momentum-as-transportation-industry-sh…	3%
https://begemot.ai/projects/1746938-razrabotka-koda-na-assemblere-dlia-vyvoda-davleniia-na-zk-displei-s-ispolzovani…	3%
https://www.chipmaker.ru/topic/143904/	3%
https://uremont.com/publications/articles/gorit-lampochka-signalizacii	3%
https://elektrikru.ru/morgaet-svet/	2%

Рисунок 1 – Справка о проверке на заимствования

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг программного кода

Листинг Б.1 – Функция инициализации датчиков и светодиода

```
void setup() {
    Serial.begin(9600);
    setupDHTSensors();
    setupBMPSensors();
    setupAnemometer();
    setupReedSwitches();
    pinMode(LED_PIN, OUTPUT);
}
```

Листинг Б.2 – Зависимости основной программы

```
#include <Arduino.h>
#include "DHTSensors.h"
#include "BMPSensors.h"
#include "Anemometer.h"
#include "ReedSwitches.h"
```

Листинг Б.3 – Инициализация переменных основной программы

```
#define DIFFERENCE_THRESHOLD 5
#define ANEMO_DIFFERENCE_THRESHOLD 20
#define LED_PIN 24
static unsigned long previousMillis = 0;

response DHT_Response;
response BMP_Response;
response Anemo_Response;
response Reed_Response;
```

Листинг Б.4 – Основная функция

```
void loop() {
    delay(100);
    DHT_Response = readDHTSensors();
    manageLED(DHT_Response.ERR_LED_PIN, DHT_Response.ErrorState);
    BMP_Response = readBMPSensors();
    manageLED(BMP_Response.ERR_LED_PIN, BMP_Response.ErrorState);
    Anemo_Response = readAnemometers();
    manageLED(Anemo_Response.ERR_LED_PIN, Anemo_Response.ErrorState);
    Reed_Response = readReedSwitches();
    manageLED(Reed_Response.ERR_LED_PIN, Reed_Response.ErrorState);

    if (canOpenRoof() == false)
    {
      Serial.println("Cant open roof");
      digitalWrite(LED_PIN, LOW);
    }
    else{
      Serial.println("Can open roof");
      digitalWrite(LED_PIN, HIGH);
    }
}
```

Листинг Б.5 – Функция управления светодиодом

```
void manageLED(short int PIN, short int errorState) {
    const long interval = 100;
```

```
            if (errorState == 0) {
                digitalWrite(PIN, LOW);
            } else if (errorState == 1) {
                // Мигает
                unsigned long currentMillis = millis();
                if (currentMillis - previousMillis >= interval) {
                  for(int i = 0; i < 10; i++){

                    int ledState = digitalRead(PIN);
                    delay(100);
                    digitalWrite(PIN, !ledState);
                  }
                  previousMillis = currentMillis;

                }
            } else if (errorState == 2) {
                digitalWrite(PIN, HIGH);
            }
        }
```

Листинг Б.6 – Функция принятия решения об открытии купола

```
        bool canOpenRoof() {
            bool DHT_NAN = isnan(DHT_Response.data1) &&
isnan(DHT_Response.data2);
            bool BMP_NAN = isnan(BMP_Response.data1) &&
isnan(BMP_Response.data2);
            bool Anemo_NAN = isnan(Anemo_Response.data1) &&
isnan(Anemo_Response.data2);
            bool Reed_NAN = isnan(Reed_Response.data1) &&
isnan(Reed_Response.data2);

            if (DHT_NAN || BMP_NAN || Anemo_NAN || Reed_NAN) {
                return false;
            }

            if(DHT_Response.ErrorRate > DIFFERENCE_THRESHOLD
||BMP_Response.ErrorRate > DIFFERENCE_THRESHOLD ||
                Anemo_Response.ErrorRate > ANEMO_DIFFERENCE_THRESHOLD ||
Reed_Response.ErrorRate > DIFFERENCE_THRESHOLD){
                  return false;
                }

            float dht_value = DHT_NAN ? DHT_Response.data2 : DHT_Response.data1;
            float bmp_value = BMP_NAN ? BMP_Response.data2 : BMP_Response.data1;
            float anemo_value = Anemo_NAN ? Anemo_Response.data2 :
Anemo_Response.data1;

            if (dht_value > 85 || bmp_value < 950 || anemo_value > 54) {
                return false;
            }

            return true;
        }
```

Листинг Б.7 – Зависимости программы считывания DHT датчиков

```
        #include <DHT.h>
        #include "response_struct.h"
```

Листинг Б.8 – Инициализация переменных программы считывания DHT датчиков

```
#define DHTPIN0 22
#define DHTPIN1 23
#define DHTTYPE DHT22
#define LED_PIN 28
#define DIFFERENCE_THRESHOLD 5

DHT dht0(DHTPIN0, DHTTYPE);
DHT dht1(DHTPIN1, DHTTYPE);
```

Листинг Б.9 – Функция инициализации DHT датчиков и светодиода

```
void setupDHTSensors() {
    pinMode(LED_PIN, OUTPUT);
    dht0.begin();
    dht1.begin();
}
```

Листинг Б.10 – Функция считывания данных с DHT датчиков и обработки ошибок

```
response readDHTSensors() {
    float hum0 = dht0.readHumidity();
    float hum1 = dht1.readHumidity();

    response resp;
    resp.ERR_LED_PIN = LED_PIN;
    resp.ErrorState = 0;
    resp.ErrorRate = 0;

    resp.data1 = hum0;
    resp.data2 = hum1;

    if (hum0 != NAN && hum1 != NAN){
      float difference = abs(hum0 - hum1);
      float average = (hum0 + hum1) / 2.0;
      float percentageDifference = (difference / average) * 100.0;
      resp.ErrorRate = percentageDifference;
      if (percentageDifference > DIFFERENCE_THRESHOLD) {
        resp.ErrorState = 1;
      }
    }
    else{
      resp.ErrorRate = NAN;
    }


    if (isnan(hum0) && isnan(hum1)) {
      resp.data1 = NAN;
      resp.data2 = NAN;
      resp.ErrorState = 2;
      return resp;
    }

    if (isnan(hum0) || isnan(hum1)) {
        if (isnan(hum0)) {
            resp.data1 = NAN;
            resp.data2 = hum1;
        } else {
            resp.data1 = hum0;
            resp.data2 = NAN;
        }
```

```
        resp.ErrorState = 1;
    }

    return resp;
}
```

Листинг Б.11 – Зависимости программы считывания BMP датчиков

```
#include <SFE_BMP180.h>
#include "response_struct.h"
```

Листинг Б.12 – Инициализация переменных программы считывания BMP датчиков

```
#define BMP_SELECT_PIN 19
#define LED_PIN 27
#define DIFFERENCE_THRESHOLD 5

SFE_BMP180 bmp1, bmp2;
```

Листинг Б.13 – Функция инициализации BMP датчиков и светодиода

```
void setupBMPSensors() {
    pinMode(BMP_SELECT_PIN, OUTPUT);
    pinMode(LED_PIN, OUTPUT);

    digitalWrite(BMP_SELECT_PIN, HIGH);
    bmp1.begin();

    digitalWrite(BMP_SELECT_PIN, LOW);
    bmp2.begin();
}
```

Листинг Б.14 – Функция считывания данных с BMP датчиков и обработки ошибок

```
response readBMPSensors() {
    response resp;
    resp.ErrorState = 0;
    resp.ErrorRate = 0;
    resp.ERR_LED_PIN = LED_PIN;

    bool sensor1Failed = false;
    bool sensor2Failed = false;

    char status1, status2;
    double T1, T2, P1, P2;

    digitalWrite(BMP_SELECT_PIN, HIGH);
    status1 = bmp1.startPressure(3);
    digitalWrite(BMP_SELECT_PIN, LOW);
    status2 = bmp2.startPressure(3);

    if (status1 != 0 || status2 != 0) {
        delay(max(status1, status2));
        digitalWrite(BMP_SELECT_PIN, HIGH);
        status1 = bmp1.getPressure(P1, T1);
        digitalWrite(BMP_SELECT_PIN, LOW);
        status2 = bmp2.getPressure(P2, T2);

        if (status1 != 0) {
            resp.data1 = P1;
        }
        if (status2 != 0) {
```

```
            resp.data2 = P2;
        }
    }
    if (status1 == 0){
      sensor1Failed = true;
      resp.data1 = NAN;
    }
    if (status2 == 0){
      sensor2Failed = true;
      resp.data2 = NAN;
    }

    if (P1 != NAN && P2 != NAN){
      float difference = abs(P1 - P2);
      float average = (P1 + P2) / 2.0;
      float percentageDifference = (difference / average) * 100.0;
      resp.ErrorRate = percentageDifference;
      if (percentageDifference > DIFFERENCE_THRESHOLD) {
        resp.ErrorState = 1;
      }
    }
    else{
      resp.ErrorRate = NAN;
    }

    if (sensor1Failed && sensor2Failed) {
        resp.ErrorState = 2;
    } else if (sensor1Failed || sensor2Failed) {
        resp.ErrorState = 1;
    }

    return resp;
}
```

Листинг Б.15 – Зависимости программы считывания анемометров

```
#include <FreqCount.h>
#include "response_struct.h"
```

Листинг Б.16 – Инициализация переменных программы считывания анемометров

```
#define ANEMO_SELECT_PIN 2
#define LED_PIN 26
#define DIFFERENCE_THRESHOLD 20

const unsigned long TIMEOUT = 2000;
const unsigned long THRESHOLD = 0;
```

Листинг Б.17 – Функция инициализации пинов выбора анемометров и светодиода

```
void setupAnemometer() {
    pinMode(ANEMO_SELECT_PIN, OUTPUT);
    pinMode(LED_PIN, OUTPUT);
}
```

Листинг Б.18 – Функция считывания данных с анемометров и обработки ошибок

```
response readAnemometers() {
    response resp;
    resp.ErrorState = 0;
```

```
    resp.ErrorRate = 0;
    resp.ERR_LED_PIN = LED_PIN;

    digitalWrite(ANEMO_SELECT_PIN, HIGH);
    bool connected1 = checkAnemometerConnection();
    digitalWrite(ANEMO_SELECT_PIN, LOW);
    bool connected2 = checkAnemometerConnection();
    float count1 = 0, count2 = 0;

    digitalWrite(ANEMO_SELECT_PIN, HIGH);
    if (connected1) {
      FreqCount.begin(500);
      while(!FreqCount.available()){continue;}
      FreqCount.end();
      count1 = FreqCount.read();
      resp.data1 = count1 * 4.8;
    }
    else{
      resp.data1 = NAN;
    }

    digitalWrite(ANEMO_SELECT_PIN, LOW);
    if (connected2) {
      FreqCount.begin(500);
      while(!FreqCount.available()){continue;}
      FreqCount.end();
      count2 = FreqCount.read();
      resp.data2 = count2 * 4.8;
    }
    else{
      resp.data2 = NAN;
    }

    if (count1 != NAN && count2 != NAN){
      float difference = abs(count1 - count2);
      float average = (count1 + count2) / 2.0;
      float percentageDifference = (difference / average) * 100.0;
      resp.ErrorRate = percentageDifference;
      if (percentageDifference > DIFFERENCE_THRESHOLD) {
        resp.ErrorState = 1;
      }
    }
    else{
      resp.ErrorRate = NAN;
    }

    if (!connected1 && !connected2) {
        resp.ErrorState = 2;
    } else if (!connected1 || !connected2) {
        resp.ErrorState = 1;
    }

    return resp;
}
```

Листинг Б.19 – Функция проверки подключения анемометров

```
bool checkAnemometerConnection() {
    unsigned long startMillis = millis();
    unsigned long count = 0;

    FreqCount.begin(500);
    while (millis() - startMillis < TIMEOUT) {
        if (FreqCount.available()) {
```

```
            FreqCount.end();
            count += FreqCount.read();
        }
    }

    return count > THRESHOLD;
}
```

Листинг Б.20 – Зависимости программы считывания герконов

```
#include "response_struct.h"
```

Листинг Б.21 – Инициализация переменных программы считывания герконов

```
#define REED_SWITCH1_PIN 3
#define REED_SWITCH2_PIN 18
#define LED_PIN 25
```

Листинг Б.22 – Функция инициализации пинов герконов и светодиода

```
void setupReedSwitches() {
    pinMode(REED_SWITCH1_PIN, INPUT_PULLUP);
    pinMode(REED_SWITCH2_PIN, INPUT_PULLUP);
    pinMode(LED_PIN, OUTPUT);
}
```

Листинг Б.23 – Функция считывания данных с анемометров и обработки ошибок

```
response readReedSwitches() {
    response resp;
    resp.ErrorState = 0;
    resp.ErrorRate = 0;
    resp.ERR_LED_PIN = LED_PIN;

    resp.data1 = digitalRead(REED_SWITCH1_PIN) == HIGH ? 1 : 0;
    resp.data2 = digitalRead(REED_SWITCH2_PIN) == HIGH ? 1 : 0;

    if(resp.data1 != resp.data2){
      resp.ErrorState = 1;
      resp.ErrorRate = 100;
    }
    if(resp.data1 == 0 && resp.data2 == 0){
      resp.ErrorState = 0;
    }
    if(resp.data1 == 1 && resp.data2 == 1){
      resp.ErrorState = 2;
    }

    return resp;
}
```

Листинг Б.24 – Структура ответа

```
struct response {
    float data1;
    float data2;
    float ErrorRate;
    short int ErrorState;
    int ERR_LED_PIN;
};
```

ПРИЛОЖЕНИЕ В
(обязательное)
Функциональная схема алгоритма,
реализующего программное средство

ПРИЛОЖЕНИЕ Г
(обязательное)
Блок схема алгоритма,
реализующего программное средство

ПРИЛОЖЕНИЕ Д
(обязательное)
Графический интерфейс пользователя

ПРИЛОЖЕНИЕ Е
(обязательное)
Ведомость документов