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

к курсовому проекту

на тему

**СИСТЕМА ОПРЕДЕЛЕНИЯ ГОРЕНИЯ ЗАГОТОВКИ В ПРОЦЕССЕ ЛАЗЕРНОЙ РЕЗКИ НА БАЗЕ *ARDUINO***

БГУИР КП 1-40 04 01 019 ПЗ

Студент Н. С. Сенько

Руководитель А. А. Калиновская

Нормоконтролер А. А. Калиновская

Минск 2024

**CОДЕРЖАНИЕ**

[Введение…………………………………………………………………………...5](#_Toc184580644)

[1 Архитектура вычислительной системы 7](#_Toc184580645)

[1.1 Понятие микроконтроллера 7](#_Toc184580646)

[1.2 Платформа Arduino 7](#_Toc184580647)

[1.3 История развития плат Arduino 10](#_Toc184580648)

[1.4 Датчик пламени пятиканальный 14](#_Toc184580649)

[1.5 Сегментный дисплей TM1637 15](#_Toc184580650)

[1.6 Обоснование выбора вычислительной системы 16](#_Toc184580651)

[2 Платформа программного обеспечения 18](#_Toc184580652)

[2.1 Среда разработки (IDE) 18](#_Toc184580653)

[2.2 PlatformIO история, версии, достоинства 19](#_Toc184580654)

[2.3 Процесс прошивки Arduino 20](#_Toc184580655)

[2.4 Язык программирования 21](#_Toc184580656)

[2.5 Обоснование выбора программной платформы 22](#_Toc184580657)

[3 Теоретическое обоснование разработки программного продукта 24](#_Toc184580658)

[3.1 Углубленный анализ инфракрасных датчиков пламени 24](#_Toc184580659)

[3.2 Калибровка датчиков 25](#_Toc184580660)

[3.3 Пользовательский интерфейс 25](#_Toc184580661)

[4 Проектирование функциональных возможностей программы 27](#_Toc184580662)

[4.1 Обоснование и описание функциональных возможностей 27](#_Toc184580663)

[4.2 Описание функциональной схемы программы 28](#_Toc184580664)

[5 Архитектура разрабатываемой программы 30](#_Toc184580665)

[5.1 Подключение устройств к Arduino 31](#_Toc184580666)

[5.2 Общая структура программы 34](#_Toc184580667)

[5.3 Описание блок-схемы алгоритма программы 35](#_Toc184580668)

[5.4 Коды ошибок 36](#_Toc184580669)

[Заключение……………………………………………………………………….38](#_Toc184580670)

[Список литературных источников 39](#_Toc184580671)

[Приложение А (обязательное) Справка о проверке на заимствования 41](#_Toc184580672)

[Приложение Б (обязательное) Листинг программного кода 42](#_Toc184580675)

[Приложение В (обязательное) Функциональная схема алгоритма, реализующего программное средство 49](#_Toc184580676)

[Приложение Г (обязательное) Блок схема алгоритма, реализующего программное средство 50](#_Toc184580677)

[Приложение Д (обязательное) Графический интерфейс пользователя 51](#_Toc184580678)

[Приложение Е (обязательное) Ведомость курсового проекта 52](#_Toc184580679)

# ВВЕДЕНИЕ

Курсовая работа посвящена разработке автоматизированной системы определения горения заготовки в процессе лазерной резки с использованием микроконтроллера *Arduino* и инфракрасных датчиков пламени. Основное назначение системы заключается в надежном мониторинге процесса, выявлении наличия горения в режиме реального времени, а также обработке возможных неисправностей датчиков и аномальных ситуаций. Такая система играет ключевую роль в повышении безопасности и эффективности производственного процесса, минимизируя влияние человеческого фактора и обеспечивая непрерывный контроль над технологическим процессом.

Актуальность данной темы обусловлена растущими требованиями к автоматизации производственных процессов и обеспечению их безопасности. Лазерная резка широко используется в промышленности благодаря высокой точности, скорости и универсальности. Однако процесс лазерной резки сопровождается риском возникновения горения заготовки, что может привести к повреждению оборудования, снижению качества продукции и даже к аварийным ситуациям.

В таких условиях использование системы автоматического определения горения становится неотъемлемой частью современных технологий. Во-первых, это позволяет минимизировать участие человека в опасных или труднодоступных зонах, что повышает уровень безопасности на предприятии. Во-вторых, автоматический контроль обеспечивает высокую точность и оперативность обнаружения горения, что способствует оптимизации технологического процесса и снижению затрат. В-третьих, возможность интеграции такой системы с промышленными интерфейсами и удаленного мониторинга открывает дополнительные перспективы для управления и анализа данных в режиме реального времени.

Целью проекта является разработка автоматизированной системы, которая будет способна стабильно и точно определять наличие горения заготовки, а также интеграция этой системы с промышленным оборудованием для передачи данных и управления процессом. Для достижения поставленной цели необходимо выполнить ряд задач.

Первоочередными задачами являются подбор и настройка инфракрасных датчиков пламени, изучение их технической документации и возможностей интеграции с платформой *Arduino*. При выборе типа датчиков для автоматизированной системы определения горения заготовки в процессе лазерной резки, существует множество вариантов, каждый из которых имеет свои преимущества и недостатки. Датчики света могут обнаруживать изменения в уровне освещенности, что может указывать на наличие пламени, но они могут быть чувствительны к другим источникам света, таким как лазерный луч, что может привести к ложным срабатываниям. Датчики температуры могут обнаруживать изменения в температуре окружающей среды, что может указывать на наличие пламени, но они могут быть менее точными, чем другие типы датчиков, и могут быть чувствительны к другим источникам тепла. Датчики дыма могут обнаруживать наличие дыма, который может указывать на наличие пламени, но они могут быть менее точными, чем другие типы датчиков, и могут быть чувствительны к другим источникам дыма. В данном случае, инфракрасные датчики пламени были выбраны как наиболее подходящий вариант, поскольку они специально разработаны для обнаружения пламени и могут быть более точными, чем другие типы датчиков. Кроме того, они могут быть легко интегрированы с платформой *Arduino*, что упрощает процесс разработки и тестирования системы.

На следующем этапе требуется реализовать алгоритмы анализа данных, которые позволят эффективно обрабатывать сигналы с датчиков, фильтровать шумы, учитывать краевые случаи и определять пороговые значения для детекции горения. Также необходимо протестировать систему, проверив ее надежность, точность и корректность передачи данных. Особое внимание будет уделено разработке механизмов самодиагностики для своевременного выявления неисправностей датчиков и предотвращения возможных сбоев [1].

Реализация анализа данных является ключевым этапом работы, так как от эффективности алгоритмов обработки сигналов с датчиков зависит вся функциональность системы. Для этого необходимо будет разработать подходы к фильтрации шумов, которые могут влиять на точность измерений, а также рассчитать оптимальные пороговые значения, позволяющие корректно детектировать наличие или отсутствие пламени.

Учет краевых случаев, таких как ложные срабатывания или сбои датчиков, обеспечит стабильность и надежность работы системы в реальных условиях. Кроме того, предусмотрены механизмы защиты от выхода из строя отдельных компонентов, что позволяет поддерживать работоспособность системы даже при наличии неисправностей.

Таким образом, разработка автоматической системы определения горения заготовки в процессе лазерной резки является актуальной задачей, направленной на улучшение качества и безопасности производственного процесса. Использование платформы *Arduino* в сочетании с инфракрасными датчиками пламени позволит создать гибкое, надежное и эффективное решение, которое может быть интегрировано в промышленное оборудование.

# 1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

## Понятие микроконтроллера

Появление первых микроконтроллеров стало важной вехой в истории развития компьютерной техники. Именно с их появлением началась новая эра, связанная с массовым применением автоматизации в различных сферах управления. Микроконтроллер представляет собой сложное устройство, в котором на одном кристалле объединены центральный процессор, память и периферийные устройства. Эти компоненты позволяют микроконтроллерам выполнять разнообразные задачи по обработке данных и управлению внешними системами. Первый патент на однокристальную микро-ЭВМ был выдан в 1971 году инженерам Майклу Кокрэну и Гэри Буну, сотрудникам американской *Texas* *Instruments*. Именно они предложили на одном кристалле разместить не только процессор, но и память с устройствами ввода-вывода [2].

Одной из проблем работы с микроконтроллерами является необходимость знания схемотехники и устройства конкретного процессора и умения программировать на ассемблере для данного процессора. Также для разработки необходимы программаторы и отладчики, что долгое время не позволяло любителям использовать микроконтроллеры в своих проектах. Однако на сегодняшний день появились устройства, позволяющие работать с микроконтроллерами без наличия серьезной материальной базы и знания многих предметов [3]. Примером такого устройства является проект *Arduino*, разработанный в Италии в 2005 году.

## Платформа *Arduino*

Устройства из семейства *Arduino* представляют собой наборы, состоящие из электронного блока и программного обеспечения. Электронный блок состоит из печатной платы с установленным микроконтроллером *ATMega* фирмы *Atmel* и набором элементов, необходимых для его работы. По сути электронный блок можно считать аналогом материнской платы современного компьютера. На нем имеются разъемы для подключения внешних устройств, а также разъем для связи с компьютером, по которому и осуществляется программирование микроконтроллера [4].

С 2008 года в компании-разработчике начался раскол, выразившийся в существовании двух независимых ветвей развития и продаж под одной торговой маркой: одна на сайте *arduino*.*cc*, другая на *arduino*.*org*. Докризисные изделия на обоих сайтах продаются под одинаковыми названиями. Набор новых изделий на сайтах различается. Также существует две ветви *Arduino* *IDE*, поддерживающие разный набор плат и библиотек. Одинаковые названия и пересекающиеся номера версий *IDE* вносят путаницу.

Под торговой маркой *Arduino* выпускается несколько плат с микроконтроллером и платы расширения (так называемые шилды). Большинство плат с микроконтроллером снабжены минимально необходимым набором обвязки для нормальной работы микроконтроллера (стабилизатор питания, кварцевый резонатор, цепочки сброса и т. п.).

*Arduino* и *Arduino*-совместимые платы спроектированы таким образом, чтобы их можно было при необходимости расширять, добавляя в устройство новые компоненты. Эти платы расширений подключаются к *Arduino* посредством установленных на них штыревых разъёмов. Существует ряд плат с унифицированным конструктивом, допускающим конструктивно жесткое соединение процессорной платы и плат расширения в стопку через штыревые линейки. Кроме того, выпускаются платы уменьшенных габаритов (например, *Nano*, *Lilypad*) и специальных конструктивов для задач робототехники. Независимыми производителями также выпускается большое количество всевозможных датчиков и исполнительных устройств, в той или иной степени совместимых с базовым конструктивом *Arduino* [5]. На рисунке 1.1 представлена плата расширения *Sensor* *Shield* 4.

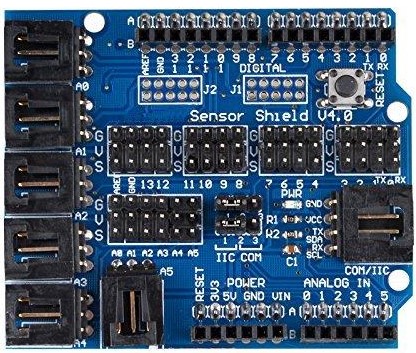


Рисунок 1.1 – Плата расширения *Sensor* *Shield* 4

Микроконтроллеры для *Arduino* отличаются наличием предварительно прошитого в них загрузчика (*bootloader*). С помощью этого загрузчика пользователь загружает свою программу в микроконтроллер без использования традиционных отдельных аппаратных программаторов. Загрузчик соединяется с компьютером через интерфейс *USB* (если он есть на плате) или с помощью отдельного переходника *UART*-*USB*. Поддержка загрузчика встроена в *Arduino* *IDE* и выполняется в один щелчок мыши.

На случай затирания загрузчика или покупки микроконтроллера без загрузчика разработчики предоставляют возможность прошить загрузчик в микроконтроллер самостоятельно. Для этого в *Arduino* *IDE* встроена поддержка нескольких популярных дешевых программаторов, а большинство плат *Arduino* имеет штыревой разъем для внутрисхемного программирования (*ICSP* для *AVR*, *JTAG* для *ARM*).

В *Arduino* *IDE* от компании, базирующейся на сайте *arduino*.*cc*, встроена возможность создания своих программно-аппаратных платформ. Этой возможностью пользуются сторонние компании, добавляющие в *Arduino* *IDE* свои наборы плат и компиляторов-загрузчиков к ним. Компания на сайте *arduino*.*org* не поддерживает такую возможность.

В линейке устройств *Arduino* в основном применяются микроконтроллеры *Atmel* *AVR*: *ATmega328*, *ATmega168*, *ATmega2560*, *ATmega32U4*, *ATTiny85* с частотой тактирования 16 или 8 МГц. В старых изделиях применялись *ATmega8*, *ATmega1280* и другие. Есть также платы на процессоре *ARM* *Cortex* M. Сторонние разработчики портировали в *Arduino* поддержку популярного *Wi*-*Fi* микроконтроллера *ESP8266*. Теперь компилировать и загружать прошивку для *ESP8266* со своими скетчами и поддержкой *Wi*-*Fi* можно прямо из *Arduino* *IDE*, получая одноплатную схему с поддержкой сети *Wi*-*Fi*.

Порты ввода-вывода микроконтроллеров оформлены в виде штыревых линеек. Никакого буферизирования, защиты, конвертации уровней или подтяжек, как правило, нет. Микроконтроллеры питаются от 5В или 3,3В, в зависимости от модели платы. Соответственно порты имеют такой же размах допустимых входных и выходных напряжений. Программисту доступны некоторые специальные возможности портов ввода-вывода микроконтроллеров, например, широтно-импульсная модуляция, аналогово-цифровой преобразователь, интерфейсы *UART*, *SPI*, *I2C*. Количество и возможности портов ввода-вывода определяются конкретным вариантом микропроцессорной платы.

Помимо портов на платах микроконтроллеров иногда устанавливается периферия в виде интерфейсов *USB* или *Ethernet*.

Сторонние производители выпускают широкую гамму датчиков и исполнительных устройств, подключаемых к *Arduino*. Например, гироскопы, компасы, манометры, гигрометры, термометры, релейные модули, индикаторы, клавиатуры и т. п. Всё это превращает *Arduino* в универсальное ядро системы, которое может быть сконфигурировано совершенно разнообразными способами.

Таким образом, можно выделить следующие преимущества платформы *Arduino*:

1 Программное обеспечение *Arduino* совместимо с операционными системами *Windows*, *Macintosh* *OSX* и *Linux*, в отличие от большинства аналогичных систем, работающих только в *Windows*.

2 *Arduino* предоставляет возможность как профессионалам, так и новичкам заниматься разработкой микропроцессорных устройств. Благодаря наличию готовых модулей и библиотек программ, создание функционирующих устройств становится гораздо проще даже для тех, кто не имеет опыта в электронике.

3 Семейство платформ *Arduino* включает различные модели, от компактных до более мощных, что позволяет выбрать наиболее подходящий вариант для конкретного проекта.

4 Для *Arduino* не нужен программатор. Всё сделано так, чтобы программирование *Arduino* для начинающих не составляло труда. Написанный код можно загрузить в микроконтроллер посредством *USB*-кабеля. Достигается это преимущество не каким-то встроенным уже заранее программатором, а специальной прошивкой – бутлоадером. Бутлоадер является специальной программой, которая запускается сразу после подключения и слушает, будут ли какие-то команды, прошивать ли кристалл, есть ли проекты *Arduino* или нет.

## История развития плат *Arduino*

В данном разделе будет рассмотрена история развития официальных плат *Arduino* по годам с перечислением их ключевых изменений.

*Arduino* *Diecimila* – первая версия платы *Arduino*, выпущенная в 2005 году. Она была разработана как простое и доступное средство для обучения программированию и электронике. Данная плата представлена на рисунке 1.2.



Рисунок 1.2 – Плата *Arduino* *Diecimila*

В 2006 году была представлена новая версия платы – *Arduino* *Duemilanove*, которая быстро завоевала популярность среди энтузиастов и разработчиков. Эта плата отличалась улучшенной совместимостью с различными устройствами и поддержкой нескольких типов микроконтроллеров, что значительно расширило её функциональность. Благодаря этим усовершенствованиям, *Duemilanove* стал основой для множества интересных и инновационных проектов в области электроники и программирования. Плата предоставила пользователям возможность легко интегрировать её в различные приложения, от простых до более сложных систем. Данная плата представлена на рисунке 1.3.

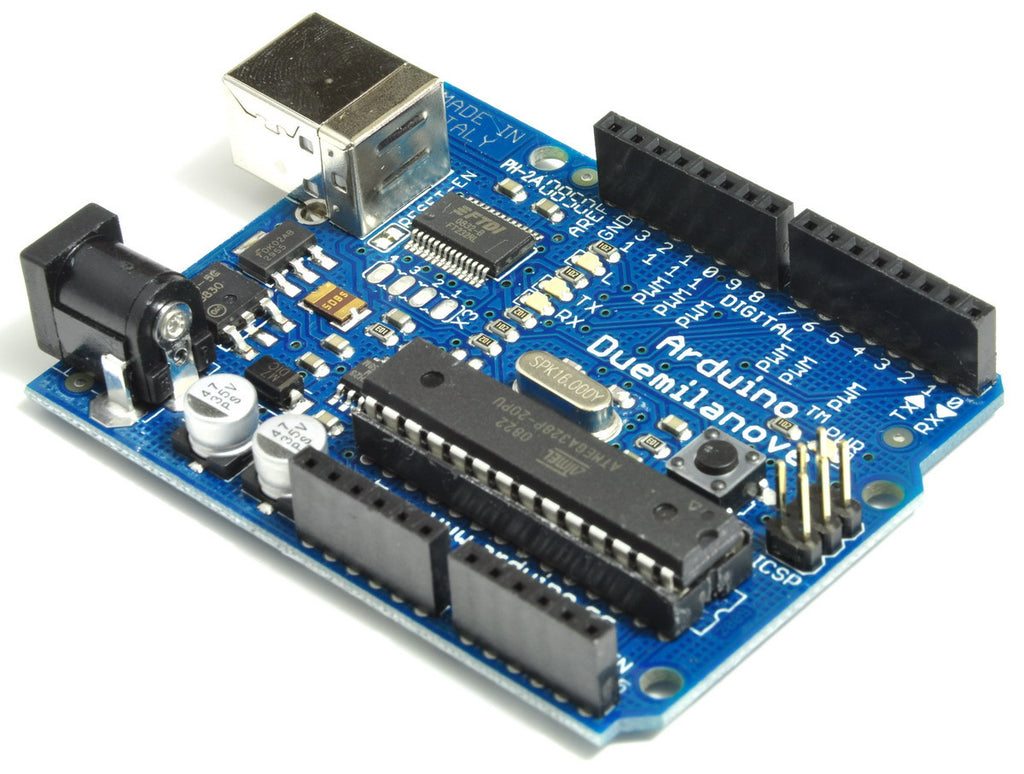


Рисунок 1.3 – Плата *Arduino* *Duemilanove*

*Arduino* *Nano* – это компактная плата, представленная в 2008 году, которая идеально подходит для работы в условиях ограниченного пространства. Благодаря своим минимальным размерам, она легко интегрируется в различные проекты, где важно экономить место. *Nano* использует микроконтроллеры *ATmega168* или *ATmega328*, что обеспечивает ей хорошую производительность при малом энергопотреблении. Эта плата стала особенно популярной среди разработчиков, работающих над малыми проектами и прототипами, благодаря своей универсальности и простоте в использовании. Данная плата представлена на рисунке 1.4.



Рисунок 1.4 – Плата *Arduino* *Nano*

*Arduino* *Uno* – это плата, которая пришла на смену *Duemilanove* и была представлена в 2009 году. Она основана на микроконтроллере *ATmega328*, который обеспечивает высокую производительность и широкие возможности для разработчиков. *Uno* быстро стала стандартом для большинства проектов благодаря своей простоте в использовании и обширному сообществу. Эта плата поддерживает множество библиотек и примеров, что делает её идеальным выбором как для новичков, так и для опытных пользователей. Данная плата представлена на рисунке 1.5.

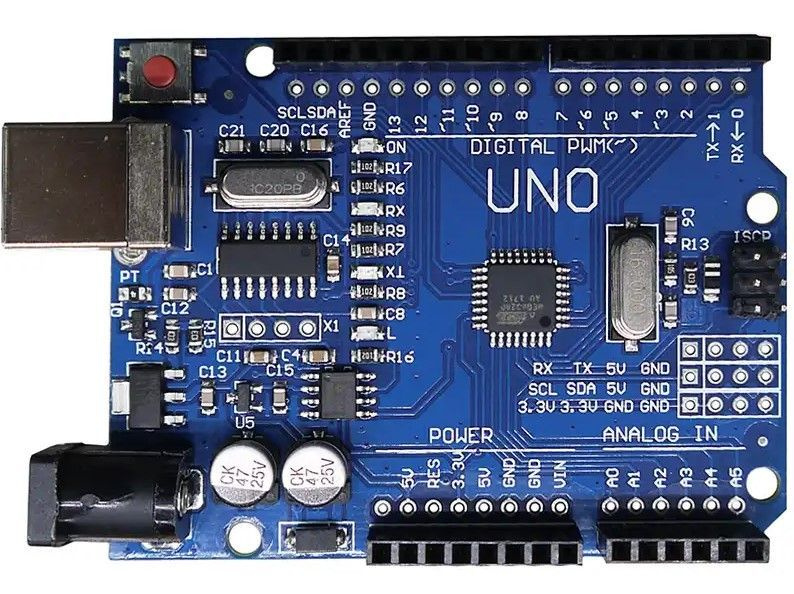


Рисунок 1.5 – Плата *Arduino* *UNO*

В 2010 году вышла новая плата – *Arduino* *Mega* с большим количеством входов/выходов (54 цифровых и 16 аналоговых). Она предназначена для более сложных проектов, требующих большей памяти и вычислительной мощности. Данная плата представлена на рисунке 1.6.



Рисунок 1.6 – Плата *Arduino* *Mega* 2560

*Arduino* *Leonardo* – это уникальная плата, представленная в 2011 году, которая выделяется своей способностью эмулировать *USB*-устройства, такие как клавиатуры и мыши. Эта функция значительно расширила возможности взаимодействия с компьютером, позволяя разработчикам создавать инновационные проекты, которые могут напрямую взаимодействовать с программным обеспечением на ПК. Благодаря использованию микроконтроллера *ATmega32U4*, *Leonardo* может отправлять данные через *USB*, что делает её отличным выбором для создания различных интерфейсов и управляемых устройств. Например, с помощью этой платы можно разработать проекты, которые автоматически вводят текст, управляют курсором или выполняют другие действия, как будто это делает обычная клавиатура или мышь. Данная плата представлена на рисунке 1.7.

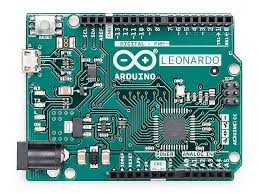


Рисунок 1.7 – Плата *Arduino* *Leonardo*

В 2013 году была представлена *Arduino* *Due* – первая плата в линейке *Arduino*, оснащённая 32-битным *ARM* *Cortex*-*M3* процессором. Это значительное обновление обеспечило пользователям более высокую производительность и расширенные возможности по сравнению с предыдущими моделями. *Arduino* *Due* предлагает увеличенную скорость обработки данных и большую ёмкость памяти, что делает её идеальной для сложных проектов, требующих значительных вычислительных ресурсов. Плата поддерживает множество современных интерфейсов, включая *USB*, *I2C* и *SPI*. Данная плата представлена на рисунке 1.8.



Рисунок 1.8 – Плата *Arduino* *Due*

*Arduino* *MKR* – это серия плат, ориентированных на Интернет вещей (*IoT*), представленная в 2015 году. Эти платы оснащены встроенными модулями для беспроводной связи, такими как *Wi*-*Fi*, *LoRa* и *GSM*, что делает их идеальными для разработки проектов, связанных с удалённым мониторингом и управлением. Данная плата представлена на рисунке 1.9.

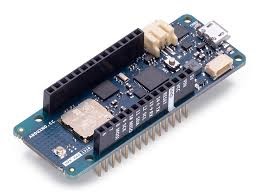


Рисунок 1.9 – Плата *Arduino* *MKR*

Серия *MKR* предлагает компактный и энергоэффективный дизайн, что позволяет легко интегрировать платы в различные устройства и системы. Каждая плата в этой серии поддерживает стандарты подключения, обеспечивая простоту в создании сетевых приложений.

## Датчик пламени пятиканальный

Пятиканальный датчик пламени представляет собой устройство, предназначенное для обнаружения пламени в ближнем инфракрасном диапазоне. Он способен регистрировать пламя в диапазоне длин волн от 700 до 1100 нм и выдавать электрический сигнал (напряжение), который можно использовать для дальнейшей обработки и анализа. Внешний вид данного датчика представлен на рисунке 1.10.



Рисунок 1.10 – Пятиканальный инфракрасный датчик пламени

Этот модуль выполнен на одной печатной плате, что делает его компактным и удобным для интеграции в различные системы.

Аналоговое выходное напряжение датчика (на аналоговом выводе) обычно обеспечивает различную пропорцию аналогового выходного напряжения в зависимости от интенсивности пламени. Поэтому высокая интенсивность пламени соответствует высокому выходному напряжению и наоборот.

Затем компаратор получает тот же аналоговый сигнал. Затем он подает сигнал на цифровой выход (*DO*).

Принцип работы датчика заключается в том, что он использует фотодиоды, чувствительные к ближнему инфракрасному излучению. Когда пламя попадает в поле зрения датчика, фотодиоды регистрируют инфракрасное излучение и преобразуют его в электрический сигнал. Этот сигнал затем обрабатывается и выдается на аналоговый выход.

Особенности модуля:

1 Двоичная индикация – позволяет определить наличие или отсутствие алкоголя в окружающей среде.

2 Аналоговый выход – предоставляет информацию о концентрации алкоголя в виде изменяющегося аналогового напряжения.

3 Легкость использования – простое взаимодействие с *Arduino* и другими микроконтроллерами.

4 Наличие пяти каналов на одной плате – позволяет существенно упростить работу с модулем, по сравнению с пятью отдельными модулями.

## Сегментный дисплей *TM1637*

Сегментный дисплей *TM1637* является четырехзначным дисплеем, который используется для отображения цифровой информации. Он состоит из четырех отдельных сегментных индикаторов, каждый из которых может отображать цифры от 0 до 9, а также некоторые специальные символы. Внешний вид данного дисплея представлен на рисунке 1.10.

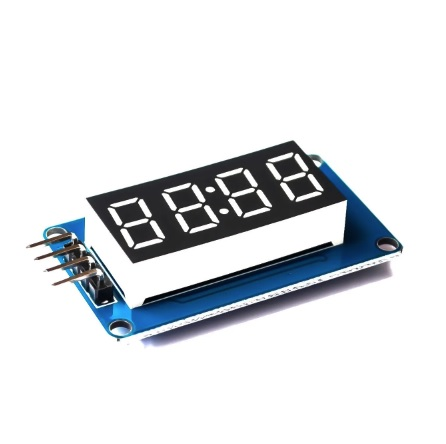


Рисунок 1.11 – Сегментный дисплей *TM1637*

Дисплей имеет следующие технические характеристики:

1 Количество знаков: 4.

2 Тип дисплея: Сегментный.

3 Размер дисплея: 12мм х 25мм х 5мм.

4 Интерфейс: 2-*wire*.

5 Напряжение питания: 5В.

6 Ток потребления: 20мА.

7 Рабочая температура дисплея: от -20 °С до +70 °С.

Дисплей подключается к плате *Arduino* *Uno* через интерфейс 2-*wire*, что позволяет легко управлять отображением информации на дисплее. Библиотека *TM1637* позволяет программно управлять дисплеем, отображая необходимую информацию. Использование сегментного дисплея *TM1637* в проекте имеет следующие преимущества:

1 Легкость использования: Дисплей легко подключается к плате *Arduino* *Uno* и управляется через простой интерфейс.

2 Низкое энергопотребление: Дисплей потребляет мало энергии, что важно для портативных устройств.

3 Высокая читаемость: Дисплей имеет высокую яркость и контрастность, что обеспечивает хорошую читаемость информации.

В целом, сегментный дисплей *TM1637* является надежным и эффективным решением для отображения информации в проекте.

## Обоснование выбора вычислительной системы

Основные преимущества выбранных для выполнения данной курсовой работы устройств можно рассмотреть более подробно:

Во-первых, микроконтроллерная платформа *Arduino* представляет собой готовый электронный модуль, включающий в себя установленный микроконтроллер и базовый набор необходимых компонентов. Это универсальное устройство, которое было разработано с целью облегчения процесса создания различных электронных проектов.

Одной из ключевых особенностей *Arduino* является возможность легкого расширения функционала устройства за счет подключения дополнительных модулей и компонентов. Это позволяет адаптировать платформу под конкретные задачи, что делает ее крайне удобной для использования в учебных и исследовательских проектах.

Еще одним важным преимуществом является то, что разработчикам нет необходимости тратить время на самостоятельное проектирование и сборку сложных схем, так как *Arduino* уже предоставляет готовую основу. Это особенно ценно в условиях ограниченного времени, поскольку позволяет сосредоточиться непосредственно на программировании и решении поставленных задач.

Во-вторых, инфракрасные датчики пламени, выбранные для данной работы, имеют возможность обнаруживать пламя в ближнем инфракрасном диапазоне, который составляет примерно 700-1100 нанометров. Такой диапазон позволяет эффективно регистрировать тепловое излучение, характерное для пламени, что делает эти датчики надежным инструментом для систем обнаружения огня. Важным аспектом является возможность использования нескольких датчиков одновременно. Это обеспечивает устойчивость системы к отказам: если один из датчиков выходит из строя, остальные продолжают выполнять свои функции, что значительно повышает общую надежность и точность устройства. Такая архитектура гарантирует, что система будет работать корректно даже в случае частичного выхода из строя отдельных элементов.

На начальной стадии проекта будет использоваться *Arduino* *Uno*, который обеспечит необходимую гибкость и простоту разработки. Однако, для уменьшения габаритов и электропитания, в дальнейшем планируется переход на *Arduino* *Nano*. Это позволит создать более компактную и энергоэффективную систему, что особенно важно для портативных устройств или систем, работающих на батарейках.

Более того, использование *Arduino* и инфракрасных датчиков пламени позволяет создать систему, которая может быть легко интегрирована с другими устройствами и системами. Это открывает широкие возможности для расширения функционала системы и создания более сложных проектов.

Кроме того, *Arduino* имеет большое сообщество разработчиков, которые создали множество библиотек и примеров кода, которые можно использовать для ускорения разработки. Это означает, что разработчикам не нужно тратить время на создание собственных библиотек и функций, а можно сосредоточиться на решении конкретных задач.

Еще одним важным преимуществом использования Arduino является его низкая стоимость. Это позволяет создавать сложные электронные проекты без значительных финансовых затрат, что особенно важно для студентов и начинающих разработчиков.

В заключение можно сказать, что использование платформы *Arduino* в сочетании с инфракрасными датчиками пламени представляет собой оптимальное решение для реализации данного курсового проекта. *Arduino* предоставляет удобную и гибкую основу для разработки, а датчики пламени обеспечивают надежное обнаружение огня, что крайне важно для выполнения поставленных задач. Эти устройства не только облегчают процесс проектирования, но и способствуют повышению эффективности и надежности системы в целом.

# 2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В этом разделе будет подробно рассмотрена программная платформа, выбранная для выполнения курсового проекта. Будет проанализирована как среда разработки, так и язык программирования, который будет использоваться в процессе реализации проекта. Особое внимание будет уделено характеристикам каждой из выбранных технологий, их возможностям и ограничениям, а также тому, как они соответствуют требованиям проекта. В конце раздела будут приведены обоснования выбора данных средств разработки, включая их преимущества, совместимость и удобство использования в контексте поставленных задач. Это позволит понять, почему именно эти инструменты были выбраны для достижения оптимальных результатов в проекте.

## Среда разработки (*IDE*)

Среда разработки (*IDE*) играет значимую роль в процессе создания проекта. Удобный интерфейс, встроенные инструменты для отладки и тестирования, а также поддержка расширяемости делают работу более продуктивной. Выбор *IDE* должен учитывать особенности проекта и предпочтения команды разработчиков.

Средой разработки данной курсовой работы было выбрано *Visual* *Studio* *Code*, имеющий понятный и функциональный интерфейс, изображенный на рисунке 2.2.

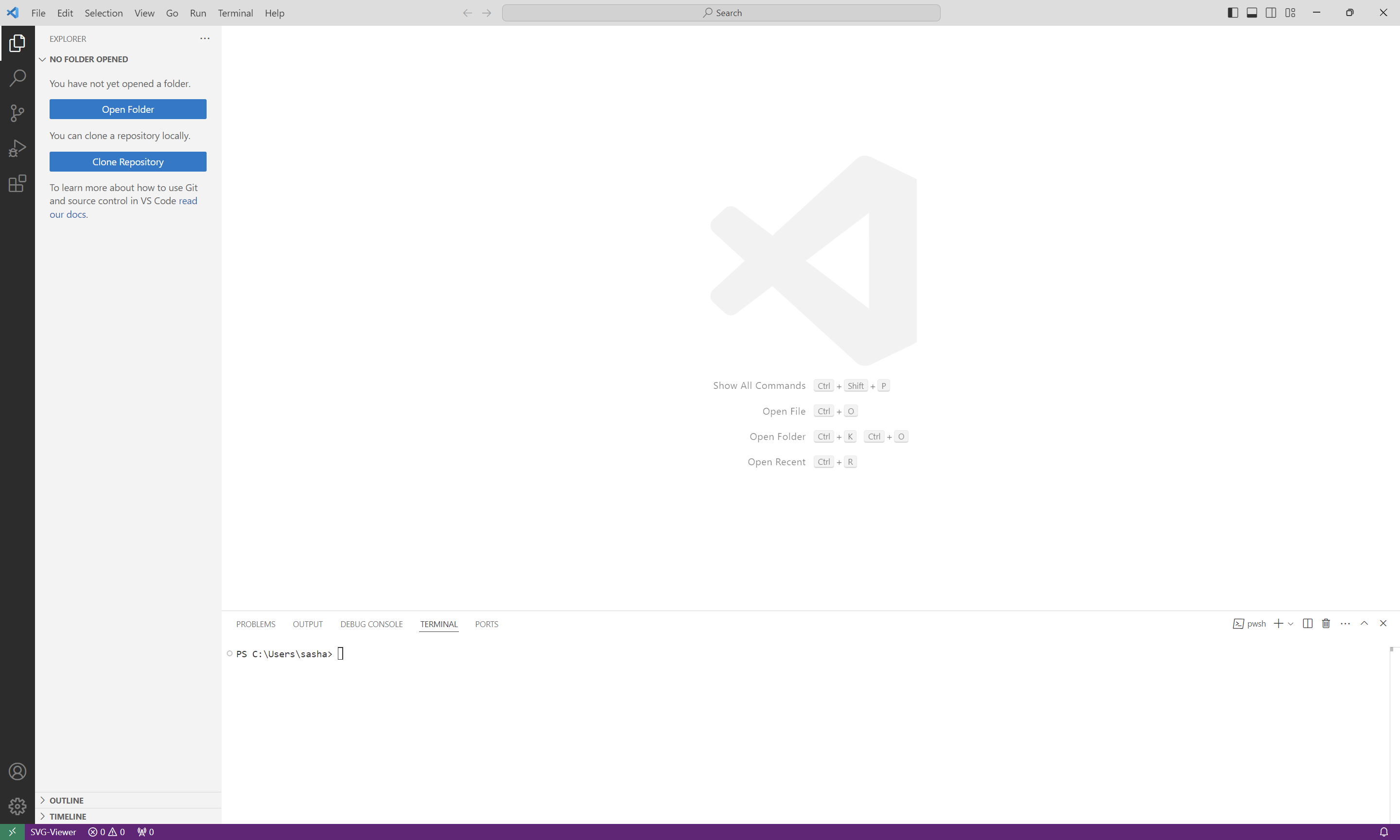


Рисунок 2.2 – Интерфейс *Visual* *Studio* *Code*.

*Visual* *Studio* *Code* (*VS* *Code*) был выпущен *Microsoft* в апреле 2015 года [6]. Первоначально позиционировался как лёгкий редактор для веб-разработки, но вскоре благодаря широким возможностям и поддержке множества языков программирования стал универсальным инструментом для разработчиков. В 2016 году проект стал открытым и получил большое сообщество разработчиков.

Первая версия *Visual* *Studio* *Code*, выпущенная в 2015 году, предоставила базовую поддержку веб-технологий, таких как *HTML*, *CSS* и *JavaScript*. В версии 1.x, охватывающей период с 2016 по 2021 год, были добавлены расширения, интеграция с *Git*, поддержка отладки и улучшения в автодополнении кода. Современные версии, начиная с 2022 года, включают поддержку работы с удалёнными серверами и контейнерами, а также улучшения интерфейса и производительности.

*Visual* *Studio* *Code* – это кроссплатформенный редактор, который работает на *Windows*, *macOS* и *Linux*. Он поддерживает множество языков программирования через плагины, обеспечивая широкую языковую поддержку. Встроенная интеграция с *Git* позволяет работать с системами контроля версий. Несмотря на свои мощные возможности, *VS* *Code* остаётся легким и отзывчивым. Благодаря обширной библиотеке расширений, редактор можно адаптировать под любые нужды разработчика.

## *PlatformIO* история, версии, достоинства

Чтобы легко запускать написанный код на *Arduino* было выбрано расширение *PlatformIO* для *Visual* *Studio* *Code* [7].

*PlatformIO* – это открытая экосистема для разработки встроенного программного обеспечения (*embedded*) с поддержкой множества платформ и микроконтроллеров.Интерфейс *PlatformIO* изображен на рисунке 2.3.

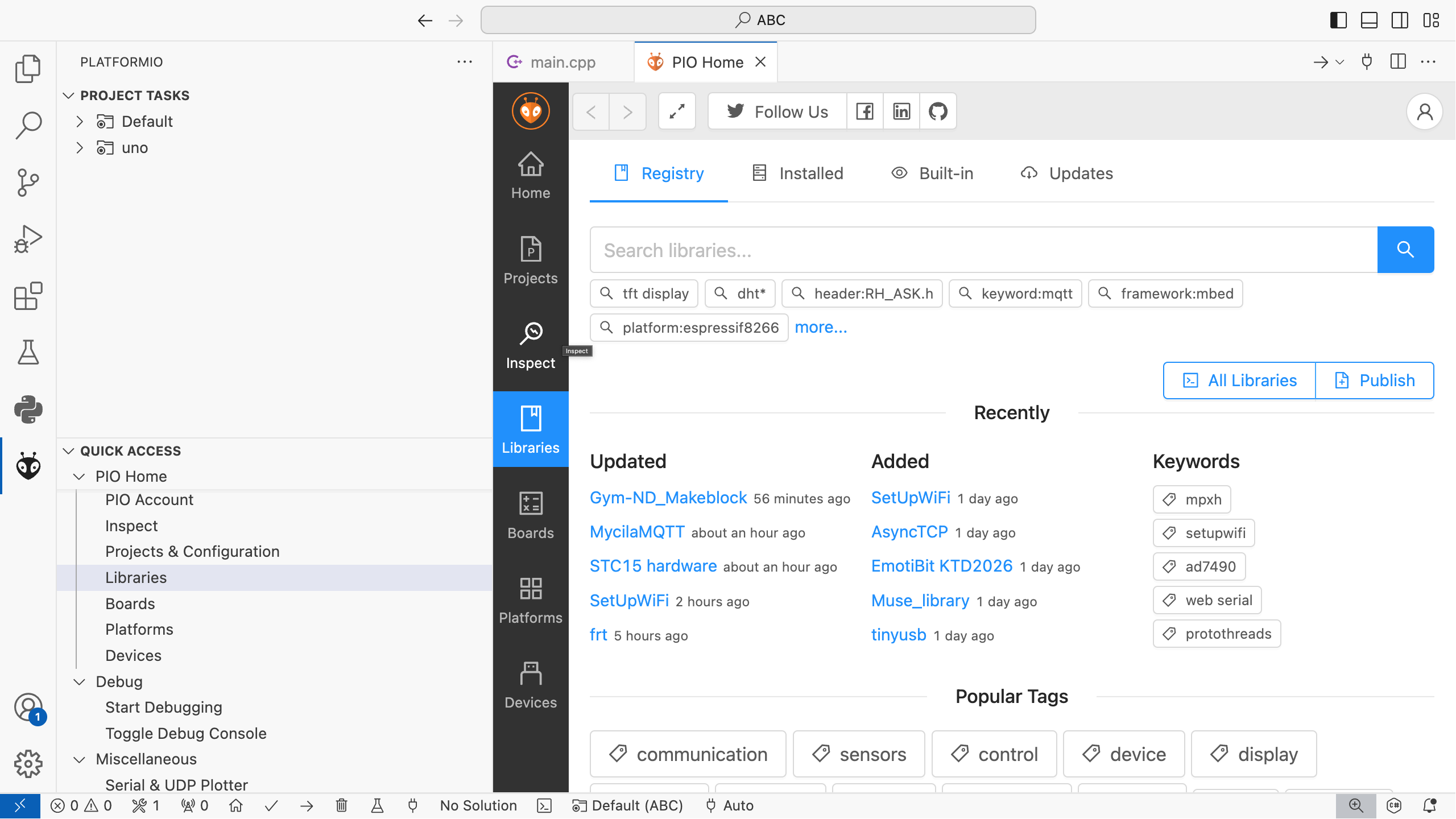


Рисунок 2.3 – Интерфейс *PlatformIO*

Проект был запущен в 2014 году для упрощения разработки программ для микроконтроллеров и быстрого управления зависимостями. Благодаря интеграции с *Visual* *Studio* *Code* и другими *IDE*, *PlatformIO* быстро завоевал популярность среди разработчиков встраиваемых систем. Первая версия, выпущенная в 2014 году, предоставила базовые возможности для управления проектами и компиляции кода для нескольких микроконтроллеров. В *PlatformIO* *Core* 3.x, выпущенной в 2016 году, была добавлена поддержка библиотек, интеграция с *Visual* *Studio* *Code* и улучшена система сборки. Современные версии 4.x и 5.x расширили поддержку платформ и микроконтроллеров, улучшили отладочные возможности и добавили поддержку удалённой разработки [8].

*PlatformIO* поддерживает множество платформ и совместим с более чем 800 микроконтроллерами, включая *Arduino*, *ESP32* и *STM32*. Интеграция с *Visual* *Studio* *Code* предоставляет мощные инструменты разработки прямо внутри этой популярной среды. Он обеспечивает автоматическое управление зависимостями, библиотеками и сборкой проекта. Также есть поддержка встроенной отладки и мониторинга работы микроконтроллеров. *PlatformIO* является кроссплатформенным и работает на *Windows*, *macOS* и *Linux*.

## Процесс прошивки *Arduino*

Процесс прошивки *Arduino* включает в себя несколько этапов, которые можно рассмотреть как с технической, так и с физической точки зрения. Когда вы пишете код для *Arduino* в *IDE*, он компилируется в машинный код, понятный микроконтроллеру *Arduino*. Компилятор преобразует исходный код в объектный код, который затем связывается с библиотеками и другими необходимыми файлами. После компиляции кода создается файл с расширением .*hex*, который содержит машинный код, готовый для загрузки в микроконтроллер *Arduino*. Когда вы нажимаете кнопку «Загрузить» в *IDE*, компьютер отправляет файл .*hex* через *USB*-порт в *Arduino*.

Микроконтроллер *Arduino* принимает файл .*hex* и записывает его в свою флеш-память. Этот процесс называется прошивкой. *Arduino* подключается к компьютеру через *USB*-кабель, который обеспечивает электрическую связь между устройствами. Когда компьютер отправляет файл .*hex* в *Arduino*, электрические сигналы передаются через *USB*-кабель и принимаются микроконтроллером *Arduino*. Микроконтроллер *Arduino* записывает полученные данные в свою флеш-память, которая представляет собой тип памяти, который сохраняет данные даже после выключения питания.

Физически, запись данных в флеш-память происходит путем изменения состояния транзисторов в микроконтроллере, что позволяет хранить данные в виде набора электрических зарядов. В целом, процесс прошивки *Arduino* включает в себя компиляцию кода, создание файла .*hex*, обмен данными с компьютером и запись данных в флеш-память микроконтроллера *Arduino*.

## Язык программирования

Выбор языка программирования является критически важным этапом, так как он определяет удобство и эффективность разработки. Например, если выбранный язык обладает хорошей поддержкой библиотек и инструментов для работы с необходимыми технологиями, это ускорит процесс реализации проекта и повысит его качество.

На языке программирования *Arduino*, основанном на *C++*, удобно разрабатывать проекты для микроконтроллеров благодаря множеству библиотек и удобному синтаксису. Эти библиотеки упрощают работу с различными аппаратными компонентами, такими как датчики, моторы и светодиоды, позволяя легко взаимодействовать с физическим миром.

История *C++* началась в 1980-х годах с разработки Бьёрном Страуструпом как расширения языка C, чтобы обеспечить поддержку объектов и классов. Первая версия была представлена в 1985 году, и язык быстро завоевал популярность благодаря сочетанию структурного и объектно-ориентированного программирования. *C++* 98 (1998 г.) – первая стандартизированная версия, в которую вошли основные концепции языка .

1 *C++* 03 (2003 г.) – корректировка стандарта *C++* 98.

2 *C++* 11 (2011 г.) – важное обновление, добавившее новые функции, такие как лямбда-выражения, *auto*-типизация и умные указатели.

3 *C++* 14 (2014 г.) – улучшения и исправления, основанные на *C++* 11.

4 *C++* 17 (2017 г.) – добавлены новые библиотеки и улучшения производительности.

5 *C++* 20 (2020 г.) – одно из самых значительных обновлений, включающее концепции, модули и корутины [9].

Программы для *Arduino* называются скетчами и сохраняются в файлах с расширением .*ino*. Эти файлы проходят обработку специальным препроцессором *Arduino*, который автоматически подключает необходимые библиотеки и преобразует код в подходящий для компиляции формат. Это делает разработку более интуитивной и доступной даже для начинающих программистов.

Скетчи включают функции, специфичные для работы с аппаратными компонентами, что упрощает реализацию проектов в области робототехники, автоматизации и Интернета вещей. Это делает платформу *Arduino* популярной среди энтузиастов электроники, инженеров и преподавателей, стремящихся обучать основам программирования и электроники.Основные элементы языка программирования *Arduino*:

1 *setup*(). Это функция, которая вызывается один раз при запуске микроконтроллера. В ней выполняются начальные настройки и инициализация компонентов, таких как порты ввода-вывода, подключенные устройства и параметры работы. Например, здесь можно установить режимы работы пинов (входные или выходные), инициализировать последовательный порт для обмена данными или настроить начальные значения переменных.

2 *loop*(). Эта функция вызывается в бесконечном цикле сразу после завершения выполнения функции *setup*(). В ней размещается основная логика программы, которая выполняется на протяжении всего времени работы микроконтроллера. Здесь можно реализовывать различные действия, такие как считывание данных с датчиков, управление *actuators*, обработка входных сигналов и выполнение условий. Поскольку функция *loop*() непрерывно повторяется, она позволяет реагировать на изменения окружающей среды и поддерживать активное взаимодействие с подключенными компонентами. Это делает программу динамичной и способной выполнять задачи в реальном времени.

3 Библиотеки. *Arduino* предлагает обширный набор библиотек функций для управления различными компонентами, такими как датчики, актуаторы, дисплеи и другие устройства. Эти библиотеки значительно упрощают взаимодействие с аппаратными средствами, позволяя разработчикам сосредоточиться на логике программы, а не на низкоуровневых деталях управления. Кроме стандартных библиотек, предоставляемых *Arduino*, разработчики имеют возможность создавать свои собственные пользовательские библиотеки. Это позволяет не только повторно использовать код в различных проектах, но и делиться им с другими разработчиками. Пользовательские библиотеки помогают организовать код, делают его более понятным и облегчают процесс разработки, особенно при работе над крупными проектами или в командах [10].

Использование уже существующего высокоуровневого языка программирования, такого как *C++*, значительно снижает порог входа для новичков. Это позволяет большему числу людей легко создавать свои проекты на *Arduino*. Высокоуровневый синтаксис и доступные библиотеки упрощают процесс разработки, позволяя сосредоточиться на концепциях и идеях, а не на сложностях низкоуровневого программирования. Кроме того, наличие обширной документации, примеров кода и активного сообщества способствует быстрому обучению и решению возникающих вопросов. Это делает *Arduino* популярным инструментом как для начинающих, так и для опытных разработчиков, желающих реализовать свои идеи в сфере электроники, робототехники и Интернета вещей (*IoT*).

## Обоснование выбора программной платформы

Важным аспектом программной платформы *Arduino* являются расширенные возможности микроконтроллеров, которые обеспечивают широкий спектр функциональности для разработчиков. Микроконтроллеры *Arduino* поддерживают различные интерфейсы, такие как *I2C*, *SPI* и *UART* [11], что позволяет легко интегрироваться с разнообразными периферийными устройствами. Эти интерфейсы обеспечивают высокую скорость передачи данных и позволяют подключать несколько устройств одновременно, что является критически важным при создании сложных встроенных систем.

Используя возможности этих интерфейсов, разработчики могут работать с множеством сенсоров, таких как датчики температуры, влажности, давления и освещения, а также с актуаторами, такими как моторы, реле и светодиоды. Это открывает широкие горизонты для реализации различных проектов – от простых устройств до сложных автоматизированных систем.

Кроме того, *Arduino* предоставляет обширную библиотеку программного обеспечения, которая упрощает взаимодействие с периферийными устройствами. Это позволяет разработчикам сосредоточиться на логике и функциональности своих проектов, а не на низкоуровневом программировании. Таким образом, *Arduino* становится идеальной платформой для обучения и прототипирования, позволяя как новичкам, так и опытным инженерам реализовывать свои идеи быстро и эффективно.

С учетом поддержки различных коммуникационных модулей, таких как *Wi*-*Fi* и *Bluetooth*, разработчики могут создавать проекты с удаленным управлением и мониторингом. Это делает *Arduino* особенно привлекательным для разработки *IoT*-устройств (интернета вещей), что открывает новые возможности для автоматизации и интеграции в повседневную жизнь [12].

Для проектов, требующих более высокого уровня точности и надежности, могут быть необходимы операционные системы реального времени (*RTOS*). *RTOS* обеспечивают обработку данных в режиме реального времени, что означает, что они гарантируют выполнение задач в течение определенного времени или в соответствии с конкретным графиком. Интересно, что *RTOS* могут работать на платформе *Arduino*, что позволяет разработчикам использовать возможности реального времени на популярной платформе. Однако из-за аппаратных ограничений выбранной платформы *Arduino* в рамках курсового проекта не имеет смысла использование подобных систем [13].

Таким образом, программная платформа *Arduino* представляет собой мощный инструмент для разработки встроенных систем и приложений. С ее помощью разработчики могут легко создавать программы для микроконтроллеров *Arduino*, используя удобную интегрированную среду разработки, стандартный язык программирования и обширную библиотеку функций. Также стоит отметить и ее постоянное развитие. *Arduino* остается в центре инноваций в области встраиваемых систем, активно внедряя новые технологии и поддерживая требования современных разработчиков.

Благодаря открытой архитектуре и плодотворному взаимодействию с сообществом, платформа продолжает привлекать внимание как опытных инженеров, так и стартапов, исследующих новые горизонты в области встраиваемых технологий.

# 3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Процесс лазерной резки требует высокой точности и безопасности. При работе с лазерными системами необходимо учитывать множество факторов, чтобы избежать нежелательных последствий, таких как повреждение оборудования или материалов. Одной из ключевых проблем, с которой сталкиваются операторы лазерных резаков, является контроль за горением заготовки. Это явление может привести к неконтролируемому распространению пламени, что, в свою очередь, может вызвать серьезные повреждения оборудования и даже травмы операторов.

В связи с этим разработка компактного и надежного устройства для определения горения заготовки в процессе лазерной резки является крайне актуальной задачей. Такое устройство может стать важным инструментом в обеспечении безопасности и эффективности процесса лазерной резки. Оно способствует снижению числа неисправностей и повышению общей производительности. Применение передовых технологий, таких как инфракрасные датчики пламени, позволяет быстро и точно определять наличие горения. Эти датчики способны реагировать на изменения температуры и светового излучения, характерные для пламени, что обеспечивает мгновенную реакцию на инциденты.

Компактные и недорогие датчики могут быть интегрированы в лазерные резаки или использоваться как портативные устройства. Это делает их доступными для широкой аудитории, включая малые и средние предприятия, которые часто сталкиваются с необходимостью оптимизации производственных процессов без значительных затрат. Внедрение таких устройств может также способствовать улучшению условий труда, снижая уровень стресса и повышая уверенность операторов в безопасности оборудования.

В итоге, создание эффективного и доступного устройства для определения горения заготовки может существенно снизить риски, связанные с лазерной резкой. Это не только повышает безопасность всех участников процесса, но и способствует улучшению качества продукции, снижению производственных простоев и затрат на ремонт оборудования. Таким образом, внедрение подобных технологий является важным шагом на пути к более устойчивым и безопасным производственным процессам.

## Углубленный анализ инфракрасных датчиков пламени

Инфракрасные датчики пламени разработаны специально для обнаружения пламени в ближнем инфракрасном диапазоне (700-1100 нм), что делает их идеальными инструментами для контроля за горением в процессе лазерной резки. Эти датчики способны быстро и точно определять наличие пламени, что критически важно для обеспечения безопасности процесса.

Однако стоит отметить, что одной из проблем датчиков является их чувствительность к шумам и различным внешним факторам. Для повышения точности измерений рекомендуется проводить многократное снятие показаний. Полученные данные следует обрабатывать с использованием соответствующих алгоритмов, и только после этого выводить результаты на индикаторные светодиоды. Такой подход поможет снизить вероятность ошибок и повысить надежность показаний.

Для обеспечения устойчивости к возможным сбоям в работе системы рекомендуется использовать несколько инфракрасных датчиков одновременно. В случае выхода одного из них из строя или его отключения от платы *Arduino*, обработка данных может основываться на показаниях других датчиков. Это гарантирует, что даже при неисправности одного из устройств, система продолжит функционировать и предоставлять актуальные результаты.

## 3.2 Калибровка датчиков

Калибровка датчиков инфракрасного пламени – важный этап для обеспечения точности и надежности устройства. Рассмотрим основные шаги калибровки, используя приведенный код. Сначала необходимо настроить окружение, используя контролируемые условия горения для измерения эталонных значений. Это поможет установить базовые уровни для датчиков. Затем нужно считывать значения с каждого датчика и усреднять их, чтобы определить базовый уровень для неактивного состояния. В коде это реализовано через массивы, которые помогают вычислять средние значения.

Далее, следует определить пороги. Используйте увеличенный порог для определения скачков, которые могут свидетельствовать о наличии пламени, что поможет избежать ложных срабатываний. Затем важно обрабатывать данные, постоянно обновляя текущие показания и сравнивая их с базовыми значениями. Отклонения определяются с помощью специальных переменных.

Регулярная проверка калибровки необходима для поддержания точности устройства, учитывая возможные изменения характеристик датчиков со временем. На основе результатов калибровки следует корректировать значения, такие как максимальное и минимальное, а также чувствительность, для оптимальной работы устройства. Эти шаги обеспечат высокую точность и надежность устройства для обнаружения пламени.

## 3.3 Пользовательский интерфейс

Создание простого и интуитивно понятного пользовательского интерфейса является ключевым аспектом успешного проекта. Четкое и понятное отображение результатов на сегментном дисплее существенно повысит удобство использования устройства, позволяя пользователям легко воспринимать необходимую информацию. Это особенно важно для устройств, которые используются в различных отраслях промышленности, где оперативное получение информации имеет решающее значение для эффективной работы.

Кроме того, на пользовательский интерфейс можно выводить данные о различных ошибках, возникающих в процессе работы. Это не только упростит процесс диагностики и устранения неполадок, но и позволит пользователям быстрее реагировать на возможные проблемы. В результате они смогут оперативно принимать меры для устранения неисправностей, что значительно повысит надежность устройства. Также стоит отметить, что наличие информативных сообщений об ошибках и состоянии системы может снизить вероятность неправильного использования устройства, обеспечивая тем самым более безопасную эксплуатацию.

Продуманный интерфейс не только улучшает взаимодействие с пользователем, но и способствует повышению общей эффективности и долговечности устройства. Это особенно актуально для устройств, которые используются в условиях высоких температур, влажности или других агрессивных сред, где любая неисправность может иметь серьезные последствия. Поэтому создание надежного и информативного интерфейса является важнейшим аспектом проектирования устройств.

Таким образом, такие устройства могут стать важным элементом в рамках комплексного подхода к безопасности в промышленности. Их интеграция в программы профилактики неисправностей может включать установку датчиков на производственных предприятиях и в лабораториях, где возможно возникновение горения. Это создает дополнительные возможности для повышения общественной безопасности и формирования культуры ответственного отношения к процессу лазерной резки. Кроме того, такие устройства могут быть использованы в различных отраслях промышленности, таких как металлургия, химическая промышленность, энергетика и другие, где безопасность является приоритетом номер один.

В целом, создание простого и интуитивно понятного пользовательского интерфейса является важнейшим аспектом проектирования устройств, которые используются в промышленности. Это не только улучшает взаимодействие с пользователем, но и способствует повышению общей эффективности и долговечности устройства, а также обеспечивает более безопасную эксплуатацию. Поэтому разработчики устройств должны уделять особое внимание созданию надежного и информативного интерфейса, который будет соответствовать требованиям промышленной безопасности.

# 4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

## 4.1 Обоснование и описание функциональных возможностей

**4.1.1** Запуск и инициализация

Функция: Запуск и инициализация системы после подачи питания или перезагрузки.

Обоснование: В момент включения устройства требуется выполнить ряд начальных настроек и подготовительных действий, таких как настройка портов на ввод или вывод и задание калибровочных параметров.

Реализация: При включении устройства микроконтроллер *Arduino* будет инициализировать все подключенные компоненты, устанавливать начальные значения и запускать цикл обработки ввода.

**4.1.2** Измерение уровня излучения

Функция: Считывание данных с инфракрасных датчиков пламени для определения признаков наличия горения.

Обоснование: Это ключевая функция системы, так как именно здесь происходит сбор данных, необходимых для определения наличия горения. Инфракрасные датчики пламени осуществляют измерения, и полученные данные передаются в программное обеспечение для дальнейшей обработки.

Реализация: Программное обеспечение будет периодически опрашивать инфракрасные датчики пламени, считывать значения и преобразовывать их в интенсивность пламени с использованием заранее определенных коэффициентов.

**4.1.3** Анализ результатов

Функция: Анализ данных для определения характера скачков на датчиках, чтобы определить, это шум или горение.

Обоснование: Анализ характера скачков на датчиках позволяет отличить шум от реального горения. Это важно для предотвращения ложных срабатываний и повышения точности системы.

Реализация: Программное обеспечение будет анализировать данные с инфракрасных датчиков пламени, чтобы определить характер скачков. Если скачки имеют характер шума, система будет их игнорировать. Если скачки указывают на наличие горения, система будет принимать.

**4.1.4** Отображение результатов

Функция: Индикация обнаружения пламени.

Обоснование: Пользователь должен видеть результаты измерения и рекомендации для принятия соответствующих решений. Отображение

Реализация: При измерении если прибор обнаружит признаки наличия горения, результат выведется на индикаторный светодиод.

**4.1.5** Отображение ошибок

Функция: Вывод на экран возникающих в процессе работы ошибок.

Обоснование: Отображение ошибок увеличивает безопасность устройства, так как предупреждает пользователя о том, что данные могут быть некорректными и детекция может не сработать, что критически важно для устройства.

Реализация: При обнаружении ошибки, выведется сообщение об ошибке с ее кодом на экран.

## 4.2 Описание функциональной схемы программы

Функциональная схема – это графическая (приложение В) или текстовое представление взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Она используется для визуализации и анализа функциональных аспектов системы, позволяя легче понять, как система выполняет определенные задачи или функции. Ниже представлено текстовое описание функциональной схемы программы.

**4.2.1** Начало программы:

1 Запуск программы на микроконтроллере *Arduino*.

2 Настройка портов микроконтроллера и инициализация подключенных устройств.

3 Инициализация последовательного порта для взаимодействия с компьютером, подключенным через *USB*.

**4.2.2** Основной цикл:

1 Программная задержка для уменьшения количества выводимых сообщений.

2 Проверка подключенных датчиков на наличие ошибок в подключении.

3 Считывание данных с датчиков, для каждого из датчиков считывается по 20 значений, а затем находится их среднее арифметическое. Данное действие необходимо для корректировки значений и получения более точных данных.

4 Проверка значений, нахождение аномалий в сигнале.

5 Возврат к началу основного цикла.

Таким образом, представлены обоснования и описание функциональных возможностей системы, каждая из которых играет ключевую роль в обеспечении надежности и эффективности устройства. Запуск и инициализация системы являются первыми шагами, необходимыми для корректной работы устройства после подачи питания. Этот процесс включает настройку портов ввода-вывода и установку калибровочных параметров, что позволяет микроконтроллеру *Arduino* инициализировать все компоненты и запустить цикл обработки данных. Это создает основу для последующих операций, обеспечивая правильную работу всех систем.

Измерение уровня излучения является центральной функцией устройства, которая отвечает за сбор данных с инфракрасных датчиков пламени. Эти датчики фиксируют изменения в интенсивности излучения и передают информацию в программное обеспечение для дальнейшего анализа. Данная функция критична для определения наличия горения, так как именно здесь происходит первичный сбор данных, необходимых для функционирования системы. Анализ результатов позволяет отличить реальные изменения от шумовых помех, что существенно для предотвращения ложных срабатываний и повышения точности детектора. Программное обеспечение анализирует скачки на датчиках, игнорируя шумовые данные и фиксируя только те изменения, которые указывают на наличие пламени.

Отображение результатов является важным аспектом взаимодействия пользователя с устройством. Это критически важно для обеспечения надежности работы устройства. В целом, все описанные функции взаимосвязаны и направлены на создание безопасной и эффективной системы, способной оперативно реагировать на изменения в окружающей среде. Более того, измерение уровня излучения и анализ результатов позволяют устройству определять тип пламени и его интенсивность. Это позволяет системе реагировать на различные типы пожаров, обеспечивая более эффективную защиту.

Отображение результатов также может быть настроено в соответствии с потребностями пользователя. Например, устройство может быть настроено на вывод сообщений об ошибках или предупреждений на экран, что позволяет пользователю быстро реагировать на потенциальные проблемы.

В целом, измерение уровня излучения и анализ результатов являются критически важными функциями устройства, которые обеспечивают его эффективную работу и безопасность. Отображение результатов и настройка устройства позволяют пользователю быстро реагировать на потенциальные опасности и обеспечивают более эффективную защиту.

# 5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Для создания макета курсового проекта была использована оригинальная *Arduino* *UNO*, которая обеспечивает все необходимые функции. Этот выбор обусловлен её доступностью и простотой работы, что идеально подходит для выполнения поставленных задач. Компоненты проекта соединяются с помощью макетной платы, что упрощает настройку и модификацию схемы, особенно на этапе разработки и тестирования [14]. Такой подход обеспечивает гибкость и упрощает эксперименты с различными конфигурациями устройства, в финальном варианте плата будет изменена на *Arduino* *NANO* с целью экономии энергии при автономной работе [15].

Макетная плата, или *breadboard* – это беспаечная плата для монтажа. Универсальный инструмент для моделирования прототипов устройств. Альтернатива создания схем без применения паяльника. Монтажную плату применяют для конструирования, отладки и тестирования будущей схемы устройства при разных условиях подключения и эксплуатации [16]. Внешний вид макетной платы представлен на рисунке 5.1.

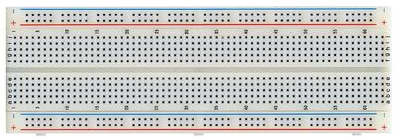


Рисунок 5.1 – Макетная плата

Макетная плата состоит из пластиковой пластины-основания с большим количеством отверстий. Расстояние между отверстиями – 2,54 мм, их диаметр – 0,8 мм. Внутри расположены два вида дорожек: контактные группы пластин и вертикальные дорожки для подачи питания. Ряды металлических пластин-рельс имеют по пять клипс каждая. Отверстия, в которые будут вставляться ножки элементов, соединяют клипсы и таким образом замыкают схему. Проводник, подключенный к отверстию в одном из рядов, одновременно соединяется с остальными контактами этого ряда. На одной рельсе можно подключить до 5 элементов, которые будут связаны между собой. Две рельсы одного ряда изолированы друг от друга. Рельсы питания расположены вертикально по краям. Как правило, синий цвет линии указывает, что нужно подавать «-» напряжения, а красный – «+». Чтобы легче работать с макетной платой, на нее производители наносят обозначения дорожек латинскими буквами и цифрами. Так проще описывать процесс сборки или писать инструкции для подключений [17]. На рисунке 5.2 представлено обозначение дорожек на макетной плате.

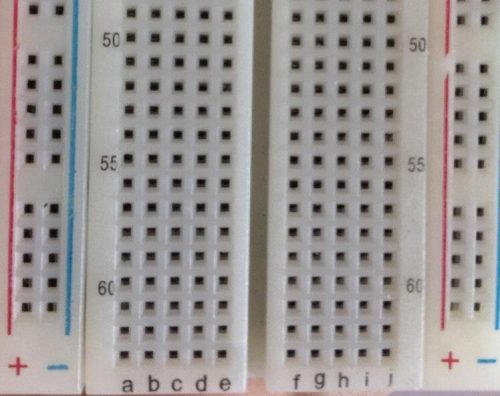


Рисунок 5.2 – Обозначение дорожек на макетной плате

Макетная плата – это современный вариант сборки схем, который позволяет легко изменять варианты схемы, добавлять компоненты и проверять теории и способы подсоединения. Если вы паяете и что-то неправильно соединили или решили что-то изменить в схеме – придется перепаивать. На *breadboard* процесс реконструкции, отладки и модификации осуществляется мгновенно: снял, подключил по-другому. Однако беспаечный способ соединения элементов схемы не такой надежный, как при сцеплении припоем. Например, при постоянной вибрации устройства контакты будут понемногу ослабляться. Поэтому в конечном варианте устройства макетная плата использоваться не будет, вместо этого все будет подключено напрямую витой парой с разъемом для возможности отключения датчика.

## 5.1 Подключение устройств к *Arduino*

Так как прямое подключение устройств к *Arduino* невозможно, для этих целей будут использоваться провода «папа-папа», которые относится к типам разъемов, используемым для подключения проводов и периферийных устройств к плате *Arduino* или другим микроконтроллерам. Эти разъемы могут быть представлены в виде «мама» (*female*) и «папа» (*male*), где «мама» представляет собой разъем с отверстиями для вставки, а «папа» – выступающий разъем для вставки в «маму» [18].

Для подключения пяти инфракрасных датчиков к *Arduino*, предположим, что у вас есть пять датчиков, каждый из которых имеет три вывода: *GND*, *VCC* и *AO* (аналоговый выход). Подключение датчиков смотр на рисунке 5.3:

При проведении тестирования, из-за того, что мкетная плата создает наводки, при испльзовании ее нет возможности снять точные показания с датчиков. Чтобы исправить это, в финальной версии устройства было решено использовать для подключения витую пару, устройство которой позволяет убрать создаваемые наводки на проводах датчиков. Это достигается благодаря тому, что провода в витой паре скручены друг вокруг друга. Эта структура является ключевым элементом для уменьшения помех. При скручивании проводов электромагнитные поля, создаваемые токами в проводниках, частично компенсируют друг друга. Помехи, которые попадают в кабель извне (например, от электроприборов или соседних кабелей), воздействуют на оба провода пары примерно одинаково. Это называется **сбалансированным воздействием**. Благодаря этому разница потенциалов между проводами остается минимальной, и сигнал на выходе получается чистым. Частота витков (количество скручиваний на единицу длины) влияет на эффективность подавления помех. Чем больше витков, тем лучше подавляются внешние наводки и перекрестные помехи. В многожильных кабелях (которые и будут использованы) пары проводов имеют разное количество витков на единицу длины. Это предотвращает перекрестные помехи между парами внутри одного кабеля [19].

Для возможности отключения будет использваван стандартный для такого типа кабеля разъем RJ-45. Также для упрощения взаимодействия с устройство было решено использовать дисплей TM1637.

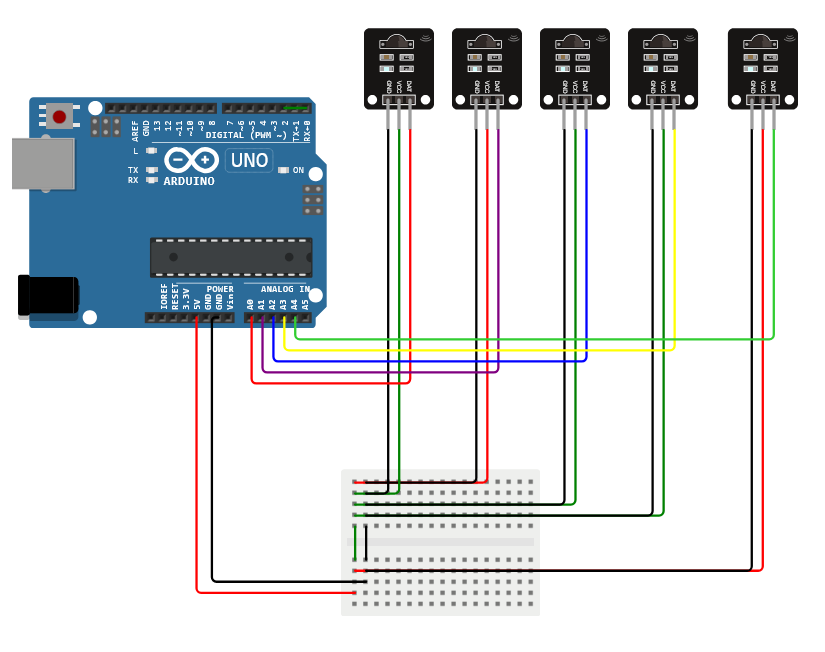


Рисунок 5.3 – Подключение всех датчиков к плате

Подключение питания для *Arduono* *UNO*, для *Ardino* *Nano* все подключения будут аналогичны:

1 Все выводы *GND* датчиков подключаются к общему *GND* на *Arduino*.

2 Все выводы *VCC* датчиков подключаются к общему 5V на *Arduino*, согласно потреблению датчика.

Подключение аналоговых выходов:

1 Разъем *AO* первого датчика подключается к аналоговому порту *A0* на *Arduino*.

2 Разъем *AO* второго датчика подключается к аналоговому порту *A1*.

3 Разъем *AO* третьего датчика подключается к аналоговому порту *A2*.

4 Разъем *AO* четвертого датчика подключается к аналоговому порту *A3*.

5 Разъем *AO* пятого датчика подключается к аналоговому порту *A4*.

Для подключения светодиода к *Arduino* понадобится резистор (обычно 220 Ом или 330 Ом) [20]. схема подключения указана ниже на рисунке 5.4.

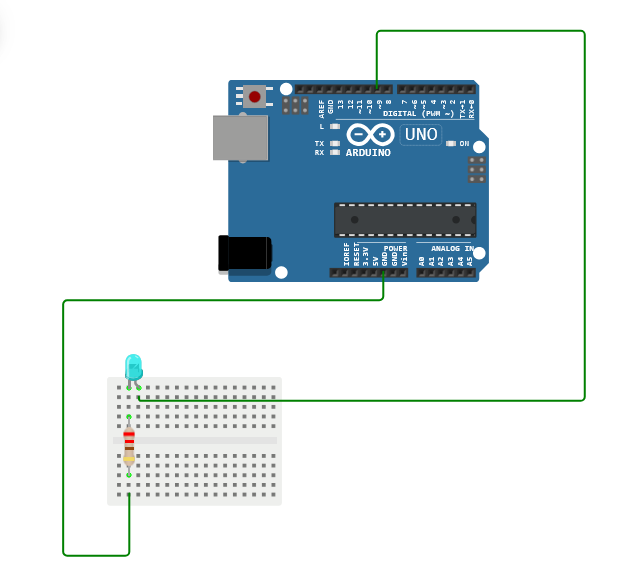


Рисунок 5.4 – Подключение информационного светодиода

В конечной версии устройства вместо светодиодов используется дисплей *TM1637*. Схема подключения представлена на рисунке 5.5.

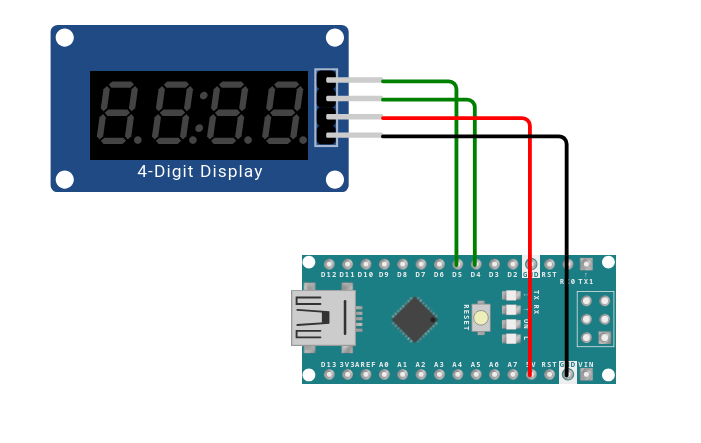


Рисунок 5.5 – Подключение дисплея

На *Arduino* *UNO* имеется всего два пина пять вольт, что может ограничивать количество напрямую подключаемых устройств, особенно если требуется питание для нескольких компонентов . В этой ситуации использование макетной платы становится особенно актуальным и удобным. Макетная плата, или прототипная плата, позволяет эффективно организовать подключение множества различных элементов, таких как датчики, светодиоды и модули, обеспечивая при этом простоту и гибкость в настройке схемы. Одним из ключевых преимуществ макетной платы является возможность быстрого монтажа и демонтажа компонентов. Это делает процесс разработки более динамичным, так как позволяет легко вносить изменения в соединения без необходимости пайки, что экономит время и усилия. Например, если нужно протестировать новый датчик или изменить конфигурацию, можно сделать это, просто переставив провода. Кроме того, макетная плата способствует лучшему управлению пространством и уменьшает беспорядок на рабочем месте. Все соединения можно организовать аккуратно и логично, что облегчает отладку и поиск возможных ошибок в схеме. Также такая платформа подходит как для новичков, так и для опытных разработчиков, предоставляя возможность экспериментировать с различными концепциями и проектами без значительных затрат [21].

## 5.2 Общая структура программы

Архитектура программного обеспечения разрабатываемого детектора имеет модульную структуру, что обеспечивает организацию программы в виде набора отдельных функциональных блоков, выполняющих определенные задачи и взаимодействующих с другими блоками для обеспечения корректной работы устройства. Ниже описаны основные модули программы:

1 Модуль инициализации и настройки отвечает за первоначальную инициализацию платы *Arduino*, настройку портов ввода-вывода и инициализацию подключенный устройств.

2 Модуль измерения уровня инфракрасного излучения с последующей записью в память для сравнения с предыдущими и последующими измерениями.

3 Модуль анализа данных отвечает за обработку данных, полученных от датчиков для дальнейшего вывода этих данных на дисплей.

4 Модуль вывода данных отвечает за отображение пользовательского интерфейса и вывод обработанных данных, полученных с датчиков, на экран.

5 Модуль обработки ошибок отвечает за обработку данных, поступающих от датчиков, и определение ошибок в работе сенсоров на основе этих данных. В случае обнаружения ошибки, ей присваивается определенный код, который далее выводится на экран.

6 Модуль вывода ошибок отвечает за отображение кодов ошибок на дисплее.

7 Основной цикл программы отвечает за работу устройства и взаимодействие остальных модулей программы между собой.

Такая организация программы упрощает ее разработку и поддержку, а также упрощает поиск и исправление ошибок в ее работе. Также такой подход является стандартным для программирования контроллеров *Arduino*, что позволяет поддерживать ее другим специалистам и находить решение распространенных ошибок в общем доступе.

## 5.3 Описание блок схемы алгоритма программы

Блок схема – это схематичное представление процесса (приложение Г), системы или компьютерного алгоритма. Блок-схемы часто применяются в разных сферах деятельности, чтобы документировать, изучать, планировать, совершенствовать и объяснять сложные процессы с помощью простых логичных диаграмм.

Для построения блок-схем применяются прямоугольники, овалы, ромбы и некоторые другие фигуры (для обозначения конкретных операций), а также соединительные стрелки, которые указывают последовательность шагов или направление процесса. Ниже представлено текстовое описание блок-схемы алгоритма программы.

**5.3.1** Подключение библиотек, инициализация устройств и глобальных переменных программы:

– подключение библиотеки *Arduino*;

– создание глобальных переменных для хранения данных с датчиков, состояния светодиодов и пороговых значений;

– создание массивов для хранения чтений с датчиков, индексов, сумм, средних значений, базовых линий и флагов обнаружения скачков.

**5.3.2** Установка программы (*setup*):

– инициализация последовательного порта и установка скорости его работы;

– инициализация массивов для хранения данных с датчиков и усреднения значений;

– первоначальное считывание данных с датчиков и усреднение значений для установки базовых линий.

**5.3.3** Обработка ошибок в работе датчиков:

– чтение данных с датчиков;

– проверка наличия питания на датчиках (в случае его отсутствия на аналоговом выходе, к которому подключен датчик, будет значение 0). В случае обнаружения ошибки вывод сообщения об ошибке;

– проверка подключения датчиков к аналоговым выходам (в случае отсутствия подключения к устройству на пине будут случайные значение, наводимые на контакты платы). В случае обнаружения ошибки вывод сообщения об ошибке;

– проверка датчиков на наличие засветки (показания датчиков близки к максимальным, невозможно определить скачки, присущие возгоранию);

– проверка сигнала на несвойственное поведение (кратковременные скачки как от пульта дистанционного управления);

– выход из функции обработки ошибок.

**5.3.4** Вывод кода ошибки на экран:

– проверка на наличие кода ошибки в переменной *err*\_*code*;

– в случае, если ошибок нет, возврат из функции вывода кода ошибки;

– установка размера текста;

– установка курсора в необходимую позицию;

– вывод кода ошибки на экран.

**5.3.5** Чтение данных с датчиков:

– считывание данных с каждого датчика и обновление массивов чтений;

– нахождение среднего арифметического из считанных значений;

– обновление базовых линий с использованием плавной адаптации;

– результата из функции.

**5.3.6** Обнаружение скачков:

– обнаружение скачков в данных с датчиков и установка флага скачка, если разница между текущим значением и базовой линией превышает порог;

– сброс флага скачка, если сигнал вернулся в норму;

– обнаружение засветки датчиков и установка флага засветки, если значение превышает максимальное значение.

**5.3.7** Обработка засветки и сигнализации:

– если засветка обнаружена, включение светодиода через 5 секунд;

– сигнализация при множественных скачках: если количество скачков превышает порог, включение светодиода сигнализации;

– понижение уровня опасности: плавное снижение счётчика скачков через определенное количество циклов.

**5.3.8** Программная задержка:

– программная задержка для стабилизации ввода-вывода;

– возврат к началу основного цикла.

## 5.4 Коды ошибок

В программе используются следующие коды ошибок:

– *Err0*: Ошибка подключения датчика (отсутствие питания на датчике);

– *Err1*: Ошибка подключения датчика (отсутствие подключения к аналоговому выходу);

– *Err2*: Ошибка датчика (наличие засветки);

– *Err3*: Ошибка сигнала (несвойственное поведение).

Эти коды ошибок выводятся на экран (приложение Д) в случае обнаружения ошибки в работе датчиков или программы. Каждый код ошибки имеет соответствующее описание и рекомендации по устранению ошибки. В случае обнаружения ошибки, программа выводит на экран соответствующий код ошибки. Это позволяет пользователю быстро определить причину ошибки и принять необходимые меры для ее устранения. Однако, существуют также необрабатываемые ошибки, которые невозможно обработать из-за того, что данные поступают одинакового вида. Например, если датчик выдает одинаковые данные при различных условиях, программа не сможет определить ошибку и выдать соответствующий код ошибки. В этом случае, программа будет продолжать работать, но результаты измерений могут быть неверными.

Есть несколько примеров таких ошибок. Ошибка датчика, вызванная повреждением или износом датчика. Ошибка при обработке данных, когда сигнал, поступающий на датчики от другого источника инфракрасного излучения, неотличим от сигнала, поступающего от пламени.

В целом, программа обеспечивает высокую точность и надежность работы, быстрое время отклика на события и запросы, предсказуемое поведение системы, многозадачность и управление ресурсами, высокий уровень безопасности и отказоустойчивости. Однако, существуют также необрабатываемые ошибки, которые невозможно обработать из-за того, что данные поступают одинакового вида. В этом случае, пользователь должен принять необходимые меры для устранения ошибки и обеспечения правильной работы программы.

# ЗАКЛЮЧЕНИЕ

Разработка автоматизированной системы определения горения заготовки в процессе лазерной резки является актуальной задачей, направленной на повышение безопасности и эффективности производственного процесса. Использование платформы *Arduino* в сочетании с инфракрасными датчиками пламени представляет собой оптимальное решение для реализации данного курсового проекта. *Arduino* предоставляет удобную и гибкую основу для разработки, а датчики пламени обеспечивают надежное обнаружение огня, что крайне важно для выполнения поставленных задач.

В ходе работы была рассмотрена архитектура вычислительной системы, в том числе платформа *Arduino* и датчики пламени. Была проанализирована программная платформа, включая среду разработки и язык программирования. Также были рассмотрены теоретические основы) разработки программного продукта, включая углубленный анализ инфракрасных датчиков пламени и калибровку датчиков.

Была разработана функциональная схема программы, которая включает в себя запуск и инициализацию системы, измерение уровня излучения, анализ результатов, отображение результатов и отображение ошибок. Также была рассмотрена архитектура разрабатываемой программы, включая общую структуру программы и модули программы.

В целом, разработка автоматизированной системы определения горения заготовки в процессе лазерной резки является важным шагом на пути к повышению безопасности и эффективности производственного процесса. Использование платформы *Arduino* и инфракрасных датчиков пламени позволяет создать гибкое, надежное и эффективное решение, которое может быть интегрировано в промышленное оборудование.

# СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Стандарт предприятия. Дипломные проекты (работы). Общие требования [Электронный ресурс]. – Режим доступа: *https*://*www*.*bsuir*.*by*/m/12\_100229\_1\_185586.*pdf*. – Дата доступа: 23.10.2024.

[2] Аливерти, П Изучаем *Arduino*. Руководство для начинающих / П Аливерти. – Москва : Эксмо, 2021.– 400 с.

[3] Что такое *Arduino* – Проекты на микроконтроллерах [Электронный ресурс]. – Режим доступа: [*http*://*micpic*.*ru*/*home*/*proekty*-*na*-*arduino*/186-*chto*-*takoe*-*arduino*.*html*](http://micpic.ru/home/proekty-na-arduino/186-chto-takoe-arduino.html). – Дата доступа: 23.10.2024.

[4] Микроконтроллеры – *Arduino* – Программирование [Электронный ресурс]. – Режим доступа: [*https*://*radio*-*hobby*.*org*/*modules*/*instruction*/*arduino*/*microcontroller*](https://radio-hobby.org/modules/instruction/arduino/microcontroller). – Дата доступа: 23.10.2024.

[5] История создания *Arduino* [Электронный ресурс]. – Режим доступа: [*https*://*arduino*.*ua*/*ru*/*art2*-*istoriya*-*sozdaniya*-*arduino*](https://arduino.ua/ru/art2-istoriya-sozdaniya-arduino). – Дата доступа: 23.10.2024.

[6] *Visual* *Studio* *Code* [Электронный ресурс]. – Режим доступа: *https*://*code*.*visualstudio*.*com*/ – Дата доступа: 29.09.2024.

[7] *PlatformIO* [Электронный ресурс]. – Режим доступа: *https*://*github*.*com*/*platformio* – Дата доступа: 29.09.2024.

[8] Документация *PlatformIO* [Электронный ресурс]. – Режим доступ: *https*://*github*.*com*/*platformio*/*platformio*-*docs* – Дата доступа: 29.09.2024.

[9] История С++ [Электронный ресурс]. – Режим доступа: *https*://*www*.*geeksforgeeks*.*org*/*history*-*of*-c/ – Дата доступа: 29.09.2024.

[10] Документация компилятор gсс [Электронный ресурс]. – Режим доступ: *https*://*gcc*.*gnu*.*org*/*onlinedocs*/ – Дата доступа: 29.09.2024.

[11] Интерфейс *I2C* в *Arduino* [Электронный ресурс]. – Режим доступа: *https*://*voltiq*.*ru*/*wiki*/*i2c*-*interface*-*in*-*arduino*/. – Дата доступа: 23.10.2024.

[12] Э. Таненбаум Архитектура Компьютера / Э. Таненбаум. – СПБ.: Питер, 2007. – 844 с.

[13] Э. Таненбаум Операционные системы / Э. Таненбаум. – СПБ.: Питер, 2015. – 51 с.

[14] Среда разработки *Arduino* | Аппаратная платформа [Электронный ресурс]. – Режим доступа: *https*://*arduino*.*ru*/*Arduino*\_*environment*. – Дата доступа: 23.10.2024.

[15] Библиотеки *Arduino* [Электронный ресурс]. – Режим доступа: [*https*://*www*.*arduino*.*cc*/*en*/*Reference*/*Libraries*](https://www.arduino.cc/en/Reference/Libraries). – Дата доступа: 23.10.2024.

[16] *Breadboard* – макетная (монтажная) беспаечная плата [Электронный ресурс]. – Режим доступа: *https*://*playarduino*.*ru*/*project*/*breadboard*-*bespaechnaya*-*plata*/. – Дата доступа: 23.10.2024.

[17] Макетная плата (База знаний) [Электронный ресурс]. – Режим доступа: *https*://*ampermarket*.*kz*/*base*/*breadboard*/. – Дата доступа: 23.10.2024.

[18] Что такое папа мама в электрике? [Электронный ресурс]. – Режим доступа: [*https*://*uchet*-*jkh*.*ru*/i/*cto*-*takoe*-*papa*-*mama*-v-*elektrike*/](https://uchet-jkh.ru/i/cto-takoe-papa-mama-v-elektrike/). – Дата доступа: 23.10.2024.

[19] *Arduino* *Uno*. Официальная документация [Электронный ресурс]. Режим доступа: [*https*://*www*.*arduino*.*cc*/*en*/*Guide*/*ArduinoUno*](https://www.arduino.cc/en/Guide/ArduinoUno). – Дата доступа: 23.10.2024.

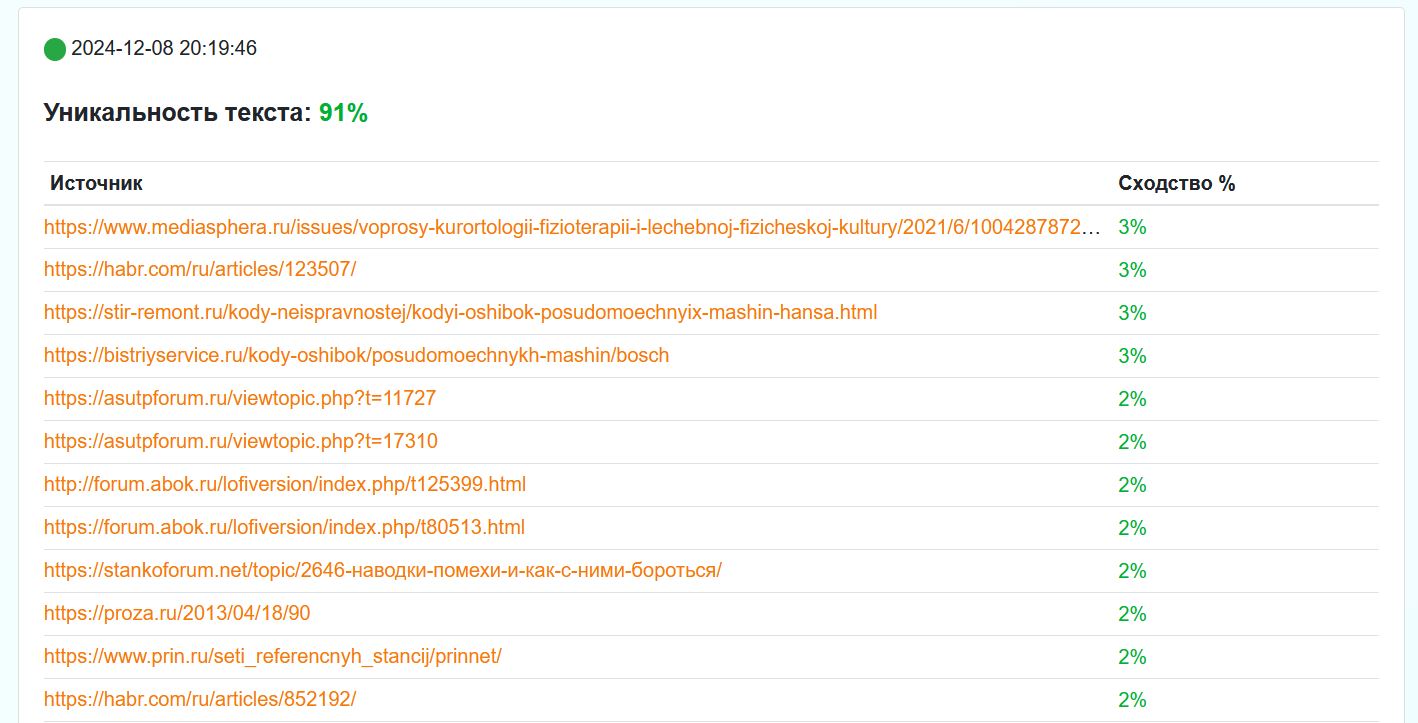
[20] Схема подключения диода [Электронный ресурс]. – Режим доступа: *https*://*schems*.*ru*/*electrics*/*diod*-*shema*-*podklyucheniya* – Дата доступа: 29.09.2024.

[21] Схема подключения *TM1637* [Электронный ресурс]. – Режим доступа: *https*://*schems*.*ru*/*electrics*/*TM1637*-*shema*-*podklyucheniya* – Дата доступа: 29.09.2024.

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Справка о проверке на заимствования



# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Листинг программного кода**

#include <Arduino.h>

#include <TM1637Display.h>

#include <ArduinoQueue.h>

/\* hardware settings \*/

// sensors configuration

#define sensorsCount 5

const int sensorPins[] = {A0, A1, A2, A3, A4};

// display configuration

#define CLK 9

#define DIO 10

/\* hardware settings \*/

/\* software settings \*/

#define readingsCount 10

#define delayMillisec 2

#define allowedSpikeDelay 40

#define allowedSpikeThreshold 60

#define maximumValueAllowed 900

/\* software settings \*/

template <class T>

class Queue

{

private:

int \_front, \_back, \_count, \_maxitems;

T\* \_data;

public:

// Конструктор

Queue(int maxitems = 256)

: \_front(0), \_back(0), \_count(0), \_maxitems(maxitems)

{

\_data = new T[\_maxitems];

}

// Деструктор

~Queue()

{

delete[] \_data;

}

// Возвращает количество элементов в очереди

inline int count() const

{

return \_count;

}

// Возвращает индекс начала очереди

inline int front() const

{

return \_front;

}

// Возвращает индекс конца очереди

inline int back() const

{

return \_back;

}

// Добавление элемента в очередь

void push(const T& item)

{

if (\_count == \_maxitems) // Если очередь заполнена

{

// Удаляем самый старый элемент (сдвигаем \_front)

\_front = (\_front + 1) % \_maxitems;

--\_count;

}

\_data[\_back] = item; // Добавляем новый элемент в конец

\_back = (\_back + 1) % \_maxitems; // Сдвигаем конец очереди

++\_count; // Увеличиваем счетчик элементов

}

// Возвращает элемент из начала очереди без удаления

T peek() const

{

if (\_count > 0)

{

return \_data[\_front];

}

}

// Удаляет и возвращает элемент из начала очереди

T pop()

{

if (\_count > 0)

{

T result = \_data[\_front];

\_front = (\_front + 1) % \_maxitems;

--\_count;

return result;

}

}

// Очистка очереди

void clear()

{

\_front = 0;

\_back = 0;

\_count = 0;

}

// Итератор для очереди

class iterator

{

const Queue<T>\* queue;

int current;

int remaining; // Количество оставшихся элементов для итерации

public:

iterator(const Queue<T>\* q, int pos, int count)

: queue(q), current(pos), remaining(count) {}

T operator\*() const

{

return queue->\_data[current];

}

iterator& operator++()

{

if (remaining > 0) // Если есть элементы для итерации

{

current = (current + 1) % queue->\_maxitems;

--remaining;

}

return \*this;

}

bool operator!=(const iterator& other) const

{

return remaining != other.remaining;

}

};

// Возвращает итератор на начало очереди

iterator begin() const

{

return iterator(this, \_front, \_count);

}

// Возвращает итератор на конец очереди

iterator end() const

{

return iterator(this, \_back, 0); // Итерация заканчивается, когда remaining == 0

}

};

class Screen{

// letters

#define DE (uint8\_t)(0b01111001)

#define DR (uint8\_t)(0b01010000)

#define DF (uint8\_t)(0b01010000)

#define DI (uint8\_t)(0b01010000)

// digits

#define D0 (uint8\_t)(0b00111111)

#define D1 (uint8\_t)(0b00000110)

#define D2 (uint8\_t)(0b01011011)

#define D3 (uint8\_t)(0b01001111)

// labels

const uint8\_t error0[4] = {DE, DR, DR, D0};

const uint8\_t error1[4] = {DE, DR, DR, D1};

const uint8\_t error2[4] = {DE, DR, DR, D2};

const uint8\_t error3[4] = {DE, DR, DR, D3};

const uint8\_t fire[4] = {DF, DI, DR, DE};

TM1637Display display = TM1637Display(CLK, DIO);

public:

Screen(){

display.setBrightness(7);

display.clear();

}

void clear(){

display.clear();

}

void show(int value, bool leadinZero = false){

display.showNumberDec(value, leadinZero);

}

void showFire(){

display.setSegments(fire);

}

void showError(uint8\_t code){

switch (code)

{

case 0:

display.setSegments(error0);

break;

case 1:

display.setSegments(error1);

break;

case 2:

display.setSegments(error2);

break;

default:

display.setSegments(error3);

break;

}

}

};

class Sensor{

public:

Queue<int> readings = Queue<int>(readingsCount);

const int spikeReadings = allowedSpikeDelay / delayMillisec;

int getMid(){

int mid = 0;

for(int i: readings){

mid += i;

}

return mid/readings.count();

}

int getMax(){

int maximum = 0;

for(int i: readings){

maximum = max(i, maximum) ;

}

return maximum;

}

int spikeCount(int average){

int spikeCount = 0;

int currentSpike = 0;

int lastSpikeTime = 0;

int currentTime = 0;

for(int i: readings){

currentTime += delayMillisec;

if (i - average >= allowedSpikeThreshold){

currentSpike += 1;

if (currentSpike >= spikeReadings){

spikeCount++;

currentSpike=0;

lastSpikeTime = currentTime;

}

} else {

currentSpike = 0;

}

}

// Check if spikes are too close

if (spikeCount > 1) {

int timeBetweenSpikes = currentTime - lastSpikeTime;

if (timeBetweenSpikes < allowedSpikeDelay) {

spikeCount = -1; // error

}

}

return spikeCount;

}

void updateReading(int value) {

readings.push(value);

}

bool isTooBright() {

return this->getMax() > maximumValueAllowed;

}

};

class Detector{

public:

Screen screen;

Sensor sensors[sensorsCount];

bool fire = false;

int error = -1;

Detector(){

}

void updateReadings() {

for (int i = 0; i < sensorsCount; i++) {

int reading = analogRead(sensorPins[i]);

sensors[i].updateReading(reading);

}

}

int getAverage(){

int average = 0;

for (int i = 0; i < sensorsCount; i++) {

average += sensors[i].getMid();

}

return average/sensorsCount;

}

void checkForSpikes() {

int average = getAverage();

for (int i = 0; i < sensorsCount; i++) {

if (sensors[i].isTooBright()) {

error = 2;

}

int spike = sensors[i].spikeCount(average);

Serial.println(spike);

if (spike > 0) {

fire = true;

} else if (spike == -1) {

error = 3;

}

}

}

unsigned long lastBlinkTime = 0;

bool isShowingError = false;

void displayState(int value) {

if(fire){

screen.showFire();

}else{

if(error > -1)

{

unsigned long currentTime = millis();

if (currentTime - lastBlinkTime >= 750) {

lastBlinkTime = currentTime;

isShowingError = !isShowingError;

}

if (isShowingError) {

screen.showError(error);

} else {

screen.show(value);

}

} else {

screen.show(value);

}

}

}

void tick(){

updateReadings();

checkForSpikes();

displayState(getAverage());

delay(delayMillisec); // Небольшая задержка для стабилизации

}

};

Detector detector;

void setup() {

Serial.begin(9600);

}

void loop() {

detector.tick();

}

# 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 курсового проекта