СОДЕРЖАНИЕ

стр.

[ВВЕДЕНИЕ 6](#_Toc75109541)

[1 Аналитический обзор учебных стендов Festo's Mechatronics и методов их управление 8](#_Toc75109542)

[2 Разработка функциональной схемы системы 11](#_Toc75109543)

[1.1 Функциональная схема программы на ПК 11](#_Toc75109544)

[1.2 Функциональная схема кода для микроконтроллера 18](#_Toc75109545)

[1.3 Общая функциональная схема всей системы 26](#_Toc75109546)

[3 Разработка приложения для ПК 28](#_Toc75109547)

[3.1 Разработка архитектуры 32](#_Toc75109548)

[3.2 Выбор и описание необходимых наборов библиотек Qt Framework 36](#_Toc75109549)

[3.3 Разработка и обоснование стркутуры отсылаемых пакетов 39](#_Toc75109550)

[4 Разработка функциональной схемы модуля работы с датчиками и органами управления Festo's Mechatronics 42](#_Toc75109551)

[5 Разработка электрической принципиальной схемы модуля работы с датчиками и органами управления 44](#_Toc75109552)

[6 Технико-экономическое обоснование целесообразности разработки программного продукта и оценка его конкурентоспособности 48](#_Toc75109553)

[6.1 Технико-экономическое обоснование целесообразности разработки программного продукта 48](#_Toc75109554)

[6.2 Оценка конкурентности программного продукта 50](#_Toc75109555)

[6.3 Определение общего объема функций программного обеспечения 53](#_Toc75109556)

[6.4 Расчет поправочных коэффициентов, учитывающих организационно-технические условия разработки программного обеспечения 56](#_Toc75109557)

[6.5 Расчет трудоемкости выполняемых работ по стадиям разработки программного обеспечения 62](#_Toc75109558)

[6.6 Расчет общей трудоемкости разработки программного обеспечения 63](#_Toc75109559)

[6.7 Расчет затрат на разработку (себестоимости) программного продукта 65](#_Toc75109560)

[6.7.1 Расчет затрат на оплату труда разработчиков 66](#_Toc75109561)

[6.7.2 Расчет затрат на машинное время 70](#_Toc75109562)

[6.7.3 Расчет затрат на изготовление эталонного экземпляра 75](#_Toc75109563)

[6.7.4 Расчет затрат на материалы 76](#_Toc75109564)

[6.7.5 Расчет общепроизводственных затрат 77](#_Toc75109565)

[6.7.6 Затраты на непроизводственные затраты 77](#_Toc75109566)

[7 Раздел по энерго- и ресурсоснабжению 80](#_Toc75109567)

[8 Раздел по охране труда и технике безопасности 86](#_Toc75109568)

[8.1 Общие сведения 86](#_Toc75109569)

[8.2 Общие сведение о точках ударов 88](#_Toc75109570)

[8.3 Общие сведения о защите от ударов молний 90](#_Toc75109571)

[ЗАКЛЮЧЕНИЕ 95](#_Toc75109572)

[Список использованных источников 96](#_Toc75109573)

ВВЕДЕНИЕ

На кафедре «Промышленная электроника» находятся машины для обучения от компании Festo’s Didactic[[1]](https://www.festo.com/us/en/). Данные машины предназначены для обучения студентов и облегчение работы преподавателей. На данных момент они работают с помощью аппаратуры Festo’s Mechatronics[[1]](https://www.festo.com/us/en/) и программного обеспечения от той же компании. Данное программное обеспечение устарело, и не может считаться современным программным обеспечением, отвечающим на сегодняшние задачи. По этим причинам будет разработана новая система для работы с машинами Festo’s Mechatronics, состоящая из программной части на ПК, микроконтроллерной части, находящейся между ПК и части самой машины.

Разработка данной системы необходима для упрощения процесса работы с пневмо-машинами компании Festo Mechantronics. Данные машины предназначены для обучения студентов кафедры, они симулируют работы реальных аппаратов, в том числе на заводе, предприятии, компании и т.л.

Данный программный продукт (далее ПП), благодаря своему удобству, позволяет гораздо проще и быстрее организовывать учебный процесс. Также, благодаря своей внутренней организации, он позволяет с наименьшим количеством проблем добавить в модуль основные принципы работы с теоретически новой машиной, добавляя “команды общения” с этой машиной в уже существующий API (Application Programming Interface)[[2]](https://ru.wikipedia.org/wiki/API).

Отдельные функции данного ПП не являются новыми, однако их объединение в данном продукте обладает новизной. Самым главным отличием данного ПП является то, что приложение способно работать абсолютно на разных операционных системах (далее ОС). Однако, не поддерживаются системы, которые официально не включает в себя QT 5.15 (Qt Company)[[2]](https://ru.wikipedia.org/wiki/The_Qt_Company).

Для разработки данного ПП используется только ПО распространяемое по свободной лицензии. Это необходимо для того, чтобы в случае, если один из компонентов системы, по каким либо причинам станет платным, у кафедры не возникли дополнительные издержки. Используются такие технологии и ПП, как Qt Framework, C++, AtmelStudio, Qt Creator. Данный ПП предназначен для различного рода работы с участием компьютером или микроконтроллером(хотя нельзя отрицать, что при доработке данного ПП, его функциональность может быть расширена, а работа ещё более автоматизирована).

Основные требования к системе подразумевает работостойкость, кроссплатформенность, масштабрируемость и скорость работы.

1 Аналитический обзор учебных стендов Festo's Mechatronics и методов их управление

Обучение, знания и образование – главные темы нынешнего века Опираясь на свой 40-летней опыт, компания Festo Didactic[1] разрабатывает решения, повышающие эффективность обучения во всех областях техники автоматизации.

Их ноу-хау гарантируют ключевые позиции на этом высокоперспективном рынке. Потребность в образовании будет продолжать расти значительными темпами. Поэтому цель – сделать процесс обучения максимально эффективным.

На сегодняшний день учебные системы по технике автоматизации более обширны и актуальны, чем когда бы то ни было. Они охватывают все актуальные темы: пневматика, электропневмоавтоматика, гидравлика, электрогидравлика, электроника, электротехника, сенсорная техника, робототехника, технологии ЧПУ, ПЛК и полевых шин Fieldbus, автоматизация производства и непрерывных процессов, мехатроника.

Качество продукции и процессов на предприятии неразрывно связано с квалификацией персонала, а, следовательно, с инновациями в области учебных технологий. Как ведущий мировой поставщик услуг по обучению технике автоматизации Festo Didactic предлагает гарантированное качество. Лучшие концепции обучения оказываются малоэффективными, если продукция и производственные процессы не отвечают требованиям практики.

Связь между ПО/виртуальной средой и реальным учебным оборудованием/ любыми ПЛК обеспечивается с помощью интерфейса для измерения, управления, регулирования – EasyPort USB. Принцип достаточно прост: через USB разъём создается соединение с ПК. Подключение к оборудованию осуществляется при помощи стандартных штекерных соединений (SysLink). Таким образом, создаётся возможность обмена с ПК

входными и выходными сигналами.

Для адаптации EasyPort к различным ситуациям разработали к аппаратному драйверу специальную программу, в графическом интерфейсе которой можно создавать различные соединения. Управление моделями процессов, приближёнными к практике.

С помощью EasyPort и входящей в объём поставки программы моделирования пользователь может управлять при помощи любого ПЛК большим количеством виртуальных моделей, приближённых к практике. Модели документированы и отвечают самым разнообразным требованиям. Некоторые из рассматриваемых тем:

– железнодорожный переход;

– парковка;

– сортировочное устройство;

– шлюзовая камера;

– лифт;

– стиральная машина;

– семисегментный дисплей;

– гаражные ворота.

Технические характеристики:

– Электропитание 24 В через разделённые винтовые зажимы или штекер SysLink

– Интерфейс связи с ПК (гальванически развязанный): USB 2.0, RS 232. С концентратором USB возможно подключение до 4 модулей. Скорость передачи данных: 115 кбод

– Аналоговый интерфейс: 15-конт. разъём D-Sub, разрешение 12 бит, 4 аналоговых входа/2 аналоговых выхода, частота дискретизации: 0,5 кГц

– Дискретный интерфейс: 16 дискретных входов/16 дискретных выходов на 2 x 24-конт. разъёмах Centronic

– по 8 дискретных входов (24 В), 8 дискретных выходов (24 В). Электропитание 24 В. Визуализация дискретных сигналов светодиодами

– Большой ЖК-дисплей, отображение канала, единицы, тренда и измеренного значения (4 знака). Выбор канала и единицы клавишами

– Возможность управления с Labview, C++ или Visual Basic посредством ActiveX Control

2 Разработка функциональной схемы системы

Вся система в сути своей представляет из себя три разделенные логически и физически между собой подсистемы.

Список этих подсистем:

– Программы, работающие на собственной машине пользователя. В качестве машин могут выступать ноутбуки и стационарные компьютеры. По сути любые машины, имеющие USB разъём и способные работать на операционных системах: Linux подобных системах и системах семейства Windows

– Микроконтроллерная часть. Представляет из себя физическое тело микроконтроллера, соединенное с одной стороны к первой подсистеме, программе на ПК, а с другой стороны к работающей машине Festo's Mechatronics

– Подсистемы машины Festo’s Mechatronics. Работы с данной подсистемой будет производиться в минимальном объёме, т.к. существующие машины подходят под все наши требования, и нам остаётся лишь подключиться к ним.

В данной части дипломного проекта разберём общее функциональное действие каждой подсистемы и всей системы в целом.

* 1. Функциональная схема программы на ПК

Программа на ПК является той частью нашей системы, которая связана непосредственно с пользователем.

Программное обеспечение — программа или множество программ, используемых для управления компьютером.

Другие определения из международных и российских стандартов:

– совокупность [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) системы обработки информации и программных документов, необходимых для эксплуатации этих программ

– все или часть программ, процедур, правил и соответствующей документации системы обработки информации

– компьютерные программы, процедуры и, возможно, соответствующая документация и данные, относящиеся к функционированию компьютерной системы.

Программное обеспечение является одним из видов обеспечения [вычислительной системы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), наряду с техническим (аппаратным), математическим, информационным, лингвистическим, организационным, методическим и правовым обеспечением.

Академические области, изучающие программное обеспечение, — это [информатика](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) и [программная инженерия](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%B8%D1%8F).

В [компьютерном сленге](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D1%81%D0%BB%D0%B5%D0%BD%D0%B3) часто используется слово «*софт*», произошедшее от английского слова «software», которое в этом смысле впервые применил в статье журнала [American Mathematical Monthly](https://ru.wikipedia.org/wiki/American_Mathematical_Monthly) математик из Принстонского университета [Джон Тьюки](https://ru.wikipedia.org/wiki/%D0%A2%D1%8C%D1%8E%D0%BA%D0%B8,_%D0%94%D0%B6%D0%BE%D0%BD) в 1958 году.

Наше программное обеспечение будет распространяться по принципу открытого программного обеспечения.

Открытое программное обеспечение — [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) с открытым [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Исходный код таких программ доступен для просмотра, изучения и изменения, что позволяет убедиться в отсутствии уязвимостей и неприемлемых для пользователя функций (к примеру, скрытого слежения за пользователем программы), принять участие в доработке самой открытой программы, использовать код для создания новых программ и исправления в них ошибок — через заимствование исходного кода, если это позволяет совместимость лицензий, или через изучение использованных алгоритмов, структур данных, технологий,

методик и интерфейсов (поскольку исходный код может существенно дополнять документацию, а при отсутствии таковой — сам служит документацией).

Термин open source (с [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) — «программное обеспечение с открытыми исходными кодами») был использован в качестве определения в 1998 году [Эриком Реймондом](https://ru.wikipedia.org/wiki/%D0%A0%D1%8D%D0%B9%D0%BC%D0%BE%D0%BD%D0%B4,_%D0%AD%D1%80%D0%B8%D0%BA_%D0%A1%D1%82%D0%B8%D0%B2%D0%B5%D0%BD) и [Брюсом Перенсом](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BD%D1%81,_%D0%91%D1%80%D1%8E%D1%81), которые утверждали, что термин free software ([свободное программное обеспечение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) в английском языке неоднозначен и смущает многих коммерческих [предпринимателей](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BF%D1%80%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D1%82%D0%B5%D0%BB%D1%8C).

Большинство программ с открытым исходным кодом является одновременно «[свободными](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)». Определение программного обеспечения с открытым кодом по [Реймонду](https://ru.wikipedia.org/wiki/%D0%A0%D1%8D%D0%B9%D0%BC%D0%BE%D0%BD%D0%B4,_%D0%AD%D1%80%D0%B8%D0%BA_%D0%A1%D1%82%D0%B8%D0%B2%D0%B5%D0%BD) и определение свободного программного обеспечения не полностью совпадают друг с другом, но близки, и большинство лицензий соответствует обоим

Функциональную схему программного обеспечения вы можете видеть на рисунке 2.1, 2.2.

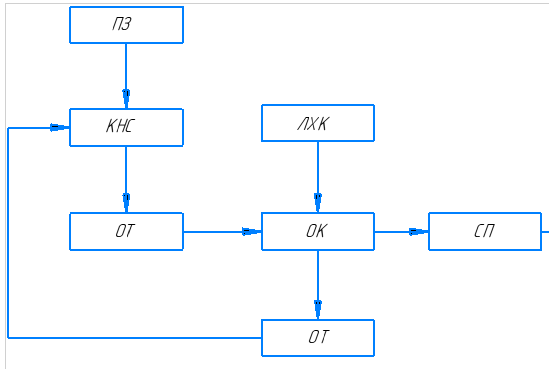


Рисунок 2.1 – Функциональная схема консольной программы на ПК

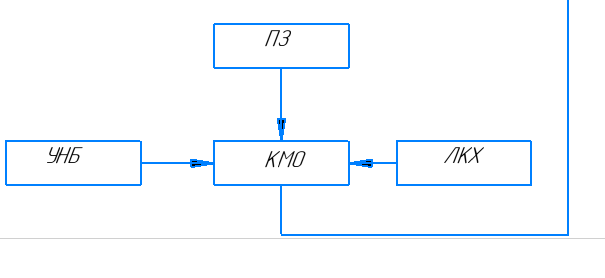


Рисунок 2.2 – Функциональная схема модуля программы на ПК

На данной функциональной схеме представлены два блока. Два блока кода будут предоставлять собой два разных подхода к применению программного обеспечения. Более подробно поговорим об этом ниже.

Разберём каждую из частей схемы в приближении. Для начала выберем верхнюю часть схемы и обсудим все её функциональные блоки.

Она содержит:

ПЗ ‒ Пользватель. Этот логический блок отвечает за теоретического пользователя программы. Данный блок связан лишь с одним другим блоков в нашем коде, в обязанности которого входит интерфейс работы пользователь/машина. В данном случае это консоль;

КНС ‒ Блок текстового редактора, отвечающего за ввод команд пользователем. В данной части схемы этот блок обязателен. Данная часть схемы представляет собой консольную программу, поэтому для нас обязателен блок, отвечающий за консольный ввод/вывод. Именно сюда пользователь будет вводить команды и читать выводящуюся информацию;

ОТ ‒ Обработчик текста. Блок обработки команд, вводимых пользователем. Этот блок необходим для того что бы рассматривать синтаксическую правильность введённой команды и возвращать результат обработки команды в консоль, если команда была введена не верно. Если же она была введена верно, блок отдаёт следующей части программы команду, преобразованную в понятный программе вид. К примеру, вместо «Включить пятый диод» команда будет преобразована в одну из 256 наборов команд, пронумерованных в шестнадцатиричной системе счисления. К примеру, 0х01, 0х02, 0х03 … 0хFF;

ЛХК ‒ Локальное хранилище команд. Этот блок логически представляет собой библиотеку упакованных пакетов. Блок обработчика отсылает сюда номер пакета, который хочет получить. Блок набора пакетов возвращает общему обработчику нужный пакет;

ОК --- Обработчик команд. Эта самый важный блок. Это так называемый «Handler». Он принимает набор команд от предыдущего блока, уже преобразованный в простой набор шестнадцатириного кода. Уже исходя из этой команды, он выбирает один из пакетов протокола общения ПК-микроконтроллер, упаковывает данных пакет и отсылает микроконтроллеру. Этот же блок принимает данные от микроконтроллера. В качестве данных служат пакеты от того же самого протокола, но уже в принимающем виде. Исходя из пакета, выбирается нужной набор информации из блока данных и выводится на консоль пользователя в удобочитаемом виде;

СП --- Серийный порт. Блок, отвечающий за пересылание пакетов байт микроконтроллеру.Этот блок должен отослать набор байт, в превидущем блоке называемых «пакетом» данных. Разница будет в том, что предыдущий блок знал, что он работает с пакетами и что из себя эти пакеты представляет. Данных блок не знает что такое пакет, он работает с простым набором байт класса QByteArray;

В данном случае программы, консоль – это всё, до чего может добраться пользователь. Подобный подход обеспечивает стойкость итоговой программы, её ошибко устойчивость. Конечный пользователь

может вводить команды в консоль. В случае, если команда была введена с ошибкой, ему будет выведена соответствующая ошибка. В случае, если команда была введена правильно, она начнёт своё выполнение в следующих блоках. Не смотря на все явные плюсы такого подхода, для обучения он не особо подходит, так как обучающийся не влияет на процесс с разных сторон. Поэтому был разработан второй вариант программного обеспечения в виде модуля.

Модуль представляет из себя статическую библиотеку C++. Она поставляется в любую программу C++, подключается с помощью директивы include и используется в дальнейшем по принципу ООП. Основой набор команд и интерфейс будет описан в документации. Рассмотим логические блоки программного обеспечения.

Схема содержит:

ПЗ ‒ Пользватель. Этот логический блок отвечает за теоретического пользователя программы. Данный блок связан лишь с одним другим блоков в нашем коде, в обязанности которого входит интерфейс работы пользователь/машина. В данной варианте программы пользователю доступен набор команд/инструкций работы, которые намного обширнее чем в прошлом варианте. Пользователь может вызывать команды не из консоли, а из программного кода, что даст пространства для реализации и обучения. Так же пользователю будет предложен вариант «послать любой набор байт»;

КМО --- Класс модульного обработчика. Объект данного класса пользователь создаст в своей программе и будет использовать интерфейс класса. Класс внутри себя будет заниматься выбором нужного пакета, упаковкой наборов байт и отсыланием пакетов/байт микроконтроллеру;

ЛХК ‒ Локальное хранилище команд. Этот блок логически представляет собой библиотеку упакованных пакетов. Блок обработчика отсылает сюда номер пакета, который хочет получить. Блок набора пакетов возвращает общему обработчику нужный пакет;

УНБ ‒ Упаковщик набора байт. Блок упаковщик «упаковывает» набор байт, введёных пользователем, в наш вариант протокола общения ПК-микроконтроллер;

Данный вариант программного обеспечение намного более гибкий и предоставляет пользователю практически полную свободу в выборе отсылаемой информации микроконтроллеру.

* 1. Функциональная схема кода для микроконтроллера

Код для микроконтроллера предоставляет из себя программу, написанную на языке C, и, соответственно, подходит под архитектурную логику данного языка.

Си ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) C) — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) [статический типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, разработанный в 1969—1973 годах сотрудником [BellLabs](https://ru.wikipedia.org/wiki/Bell_Labs) [Деннисом Ритчи](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D1%82%D1%87%D0%B8,_%D0%94%D0%B5%D0%BD%D0%BD%D0%B8%D1%81) как развитие языка [Би](https://ru.wikipedia.org/wiki/%D0%91%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Первоначально был разработан для реализации [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [UNIX](https://ru.wikipedia.org/wiki/UNIX), но впоследствии был [перенесён](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) на множество других платформ. Согласно дизайну языка, его конструкции близко сопоставляются типичным [машинным инструкциям](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), благодаря чему он нашёл применение в проектах, для которых был свойственен [язык ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0), в том числе как в [операционных системах](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), так и в различном [прикладном программном обеспечении](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) для множества устройств — от [суперкомпьютеров](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) до [встраиваемых систем](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Язык программирования Си оказал существенное влияние на развитие индустрии программного обеспечения, а его [синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) стал основой для таких языков программирования, как [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [C#](https://ru.wikipedia.org/wiki/C_Sharp), [Java](https://ru.wikipedia.org/wiki/Java) и [Objective-C](https://ru.wikipedia.org/wiki/Objective-C).

Программа представляет собой набор файлов с кодом на языке Си, которые могут компилироваться в объектные файлы. Объектные файлы затем проходят этап компановки друг с другом, а также с внешними библиотеками, в результате чего получается итоговый исполняемый файл или библиотека. Связь файлов друг с другом, равно как и с библиотеками, требует описания прототипов используемых функций, внешних переменных и необходимых типов данных в

каждом файле. Такие данные принято выносить в отдельные заголовочные файлы, которые подключаются с помощью [директивы](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) #include в тех файлах, где требуется та или иная функциональность, и позволяют организовывать систему, похожую на систему модулей. Модулем в таком случае может выступать:

— набор отдельных файлов с исходным кодом, для которых представлен интерфейс в виде заголовочных файлов;

— объектная библиотека или её часть, с соответствующими заголовочными файлами;

— самодостаточный набор из одного или более заголовочных файлов (интерфейсная библиотека);

— статическая библиотека или её часть с соответствующими заголовочными файлами;

— динамическая библиотека или её часть с соответствующими заголовочными файлами.

Поскольку директива #include лишь подставляет текст другого файла на этапе [препроцессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80_%D0%A1%D0%B8), многократное подключение одного и того же файла может приводить к ошибкам этапа компиляции. Поэтому в таких файлах используется [защита от повторного включения](https://ru.wikipedia.org/wiki/Include_guard) с помощью [макрокоманд](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0) #define и #ifndef.

Текст файла исходного кода на языке Си состоит из набора глобальных определений данных, типов и функций. Глобальные переменные и функции, объявленные со спецификаторами static и inline, доступны только в пределах того файла, в котором они объявлены, либо при включении одного файла в другой через директиву #include. При этом функции и переменные, объявленные в заголовочном файле со словом static, будут создаваться заново при каждом подключении заголовочного файла к очередному файлу с исходным кодом. Глобальные переменные и прототипы функции, объявленные со спецификатором extern, считаются подключаемыми из других файлов. То есть

их допускается использовать в соответствии с описанием; предполагается, что после сборки программы они будут связаны [компоновщиком](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%BE%D0%B2%D1%89%D0%B8%D0%BA) с оригинальными объектами и функциями, описанными в своих файлах.

Глобальные переменные и функции, кроме static и inline, могут быть доступны из других файлов при условии их надлежащего объявления там со спецификатором extern. Переменные и функции, объявленные с модификатором static, также могут быть доступны в других файлах, но лишь при передаче их адреса по указателю. Объявления типов typedef, struct и union не могут импортироваться в других файлах. При необходимости использования в других файлах они должны быть там продублированы либо вынесены в отдельный заголовочный файл. То же самое относится и к inline-функциям.

Рассмотрим функциональную схему нашего микроконтроллера и разберём принцип работы и основные задачи каждого блока в отдельности, а так же взаимосвязь между ними.

Для нас важно понять, по какому принципу и с какой целью был написан исходный код для нашего микроконтроллера.

Микроконтроллер ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Micro Controller Unit, MCU) — [микросхема](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D1%85%D0%B5%D0%BC%D0%B0), предназначенная для управления [электронными](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0) [устройствами](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE).

Типичный микроконтроллер сочетает на одном кристалле функции [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и [периферийных устройств](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B8%D1%84%D0%B5%D1%80%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), содержит [ОЗУ](https://ru.wikipedia.org/wiki/%D0%9E%D0%97%D0%A3) и (или) [ПЗУ](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B7%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE). По сути, это однокристальный [компьютер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), способный выполнять относительно простые задачи. На рисунке 2.2 вы можете увидеть пример микроконтроллера.

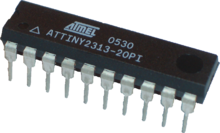


Рисунок 2.3 --- Пример микроконтроллера. В качестве примера

ATiny2313 американской фирмы Atmel

Отличается от [микропроцессора](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) интегрированными в микросхему устройствами ввода-вывода, таймерами и другими периферийными устройствами.

При проектировании микроконтроллеров приходится соблюдать компромисс между размерами и стоимостью с одной стороны и гибкостью и производительностью с другой. Для разных приложений оптимальное соотношение этих и других параметров может различаться очень сильно. Поэтому существует огромное количество типов микроконтроллеров, отличающихся архитектурой процессорного модуля, размером и типом встроенной памяти, набором периферийных устройств, [типом корпуса](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D1%8B_%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81%D0%BE%D0%B2_%D0%BC%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D1%85%D0%B5%D0%BC) и т. д. В отличие от обычных компьютерных микропроцессоров, в микроконтроллерах часто используется [гарвардская архитектура](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%80%D0%B2%D0%B0%D1%80%D0%B4%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) памяти, то есть раздельное хранение данных и команд в [ОЗУ](https://ru.wikipedia.org/wiki/%D0%9E%D0%97%D0%A3) и [ПЗУ](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B7%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE) соответственно.

Кроме ОЗУ, микроконтроллер может иметь встроенную [энергонезависимую память](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D0%B5%D1%80%D0%B3%D0%BE%D0%BD%D0%B5%D0%B7%D0%B0%D0%B2%D0%B8%D1%81%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) для хранения программы и данных. Многие модели контроллеров вообще не имеют [шин](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D0%BD%D0%B0_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80)) для подключения внешней памяти.

Наиболее дешёвые типы памяти допускают лишь однократную запись, либо хранимая программа записывается в кристалл на этапе изготовления (конфигурацией набора технологических масок). Такие устройства подходят для массового производства в тех случаях, когда программа контроллера не будет обновляться. Другие модификации контроллеров обладают возможностью многократной перезаписи программы в энергонезависимой памяти.

Для дипломной работы был выбран микроконтроллер семейства ATmega, а конкретнее ATmega328PA.

Пример данного микроконтроллера вы можете наблюдать на рисунке 2.3

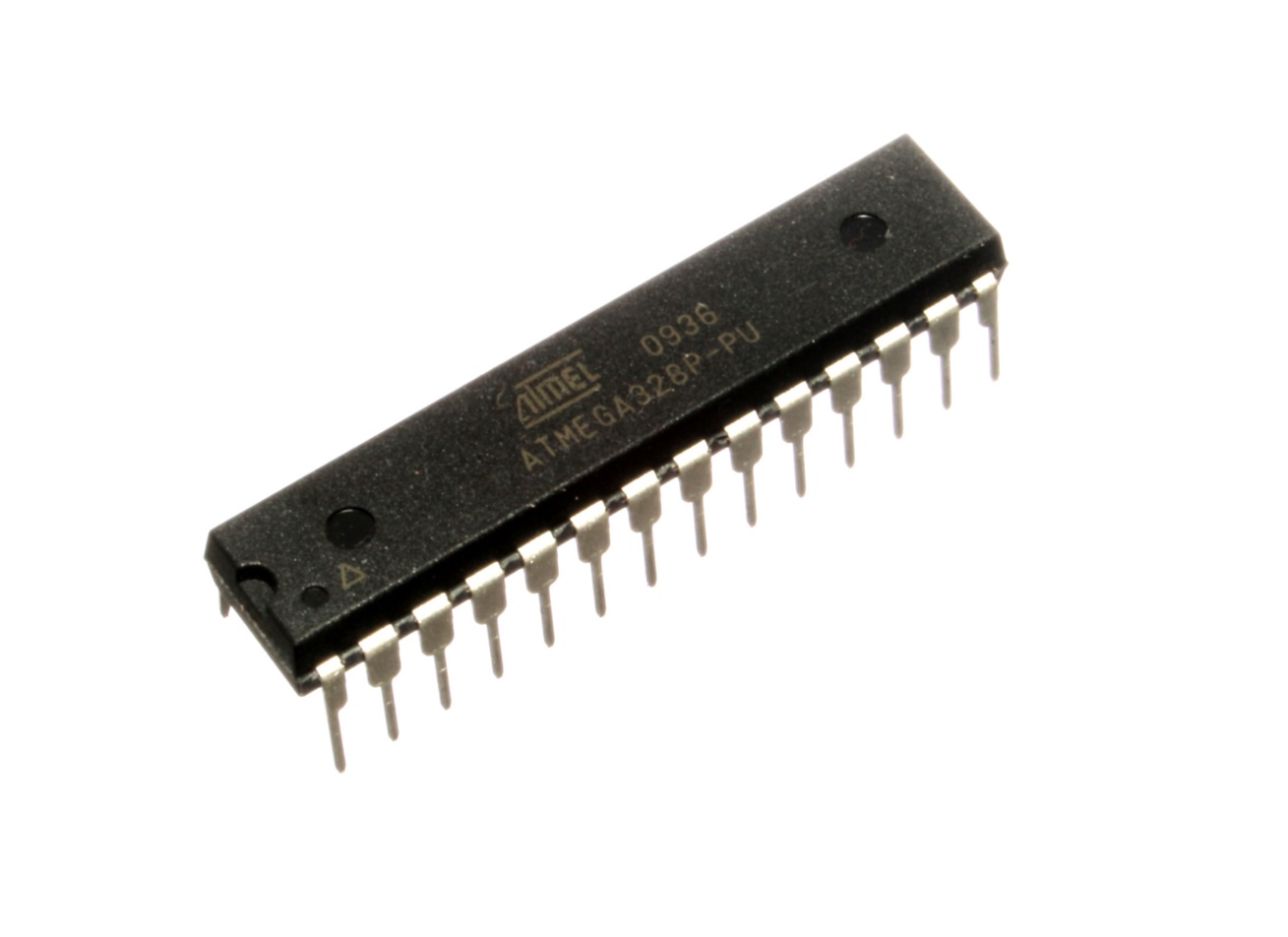


Рисунок 2.4 – Пример микроконтроллера ATmega328

ATmega328 - это однокристальный микроконтроллер, созданный Atmel в семействе megaAVR (позже Microchip Technology приобрела Atmel в 2016 году). Он имеет модифицированное ядро 8-битного RISC-процессора с гарвардской архитектурой. Микроконтроллер Atmega328 используется в базовых платах Arduino, то есть в Arduino UNO, Arduino Pro Mini и Arduino Nano.

Характеристики:

8-битный микроконтроллер AVR на базе RISC Atmel объединяет 32 КБ флэш-памяти ISP с возможностями чтения во время записи, 1 КБ EEPROM, 2 КБ SRAM, 23 линии ввода-вывода общего назначения, 32 рабочих регистра общего назначения, три гибких таймера / счетчики с режимами сравнения, внутренними и внешними прерываниями, последовательным программируемым USART, байтовым 2-проводным последовательным интерфейсом, последовательным портом SPI, 6-канальным 10-битным аналого-цифровым преобразователем (8 каналов в пакетах TQFP и QFN / MLF), программируемый сторожевой таймер с внутренним генератором и пять программно выбираемых режимов энергосбережения. Устройство работает в диапазоне 1,8-5,5 вольт. Пропускная способность устройства приближается к 1 MIPS на МГц.

Безусловно, без знания того, какой пин за что отвечает, невозможно написать и код для микроконтроллера. В этом нам поможет официальный даташит.

Техническая спецификация, карта данных, даташит ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Datasheet, data sheet) — документ, который объединяет в себе технические характеристики продукта, материала, компонента (например, электронного) или подсистемы и предназначен для использования [инженером-конструктором](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80-%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80). Создается компанией-разработчиком (производителем) и начинается со вводной страницы, за которой следуют списки конкретных характеристик. В случае наличия [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) его располагают в конце документа или в отдельном файле.

На рисунке 2.4 вы можете видеть техническую спецификацию нашего микроконтроллера.

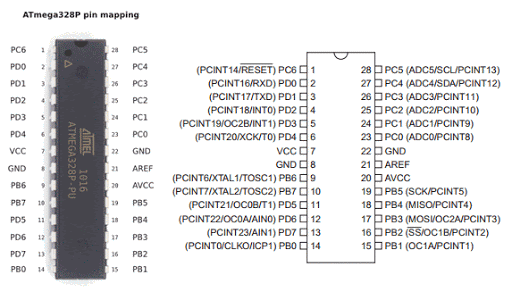


Рисунок 2.5 – Техническая спецификация микроконтроллера

cемейства Atmega328

Исходя из всех этих данных, можно написать код работы для микроконтроллера.

Разберём логически код микроконтроллера благодаря функциональноый схеме. Функциональную схему микроконтроллера вы можете наблюдать на рисунке 2.5.

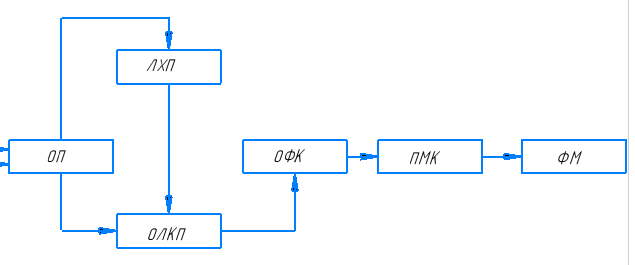


Рисунок 2.6 --- Функциональная схема блока микроконтроллера

Данная схема содержит:

ОП --- Обработчик прерывания. Блок ISR. Логически представляет из себя функцию обработчик прерывания, вызванного передачей байт из внешнего источника по соединению. Функция кладёт пришедший байт в конец локального хранилища, после чего вызывает

ЛХП --- Локальное хранилище пакетов. Блок локального хранилища. Представляет из себя массив байт с полным доступом по индексу. Блок ISR кладёт в конец массива один байт

ОЛХП --- Обработчик локального хранилища пакетов. Данный блок вызывается из блока ISR. Он проверяет, хватает ли в хранилища байт для начала обработки пакета. Если в хранилище достаточно байт, оно начинает верифицировать пакет

ОФК --- Обработчик функций команд. Представляет из себя логическое хранилище команд. Обработчик вызывает исполнение нужной команды

ПМК --- Порты микроконтроллера. Хотя в логическом виде он у нас присутствует, в кодовом виде его нету. Этот блок просто выставляет значения на порты выхода микроконтроллера.

ФМ --- Фесто машины. Блок машин Festo’s Mechatronics. Представляет из себя машину, с которой будет работать МК и является конечном пунктом команд. Не находится в логическом поле микроконтроллера, но выделен на схеме, для лучшего объяснения принципа работы.

* 1. Общая функциональная схема всей системы

Общая функциональная схема представлена в приложении А1. На ней представлены вся система и принцип её взаимодействия.

Первые логические блоки представляют из себя либо консольное приложение, либо встраиваемый C++ модуль.

Консольное приложение представляет из себя простую консоль для работы в режиме передачи набора данных или в режиме отладки. Пользователь записывает данные в консоль путём вызова соответствующих команд и программа генерирует пакеты протокола, которые позже будут отправлены микроконтроллеру. Стоить отметить что в подобном режиме пользователь все равно будет иметь возможность отправить произвольный набор байт, упакованный в пакет, попросту вызывая отдельную команду вида write(command, ByteArray).

Встраиваемый модуль выглядит как обычная статическая библиотека C++ с аналоговым вариантом название “FestoController>. Для того что бы была возможность подключить данную библиотеку к своей программе необходимо поместить исполняемый файлы и хедеры библиотеки в каталог с программой пользователя. Использовав функцию include мы можем подключить наш модуль и использовать абсолютно всего его функции как единицы кода.

Вне зависимости от вида использования программного обеспечения, пакеты данных передаются в микроконтроллер по USB ISP. В микроконтроллере генерируется прерывание, обрабатываемое функцией ISR. В течении одного прерывания нам необходимо записать принимаемый байт данных в локальное хранилище. Стоит отметить что объём хранилища не превышает 256 байт. Этого вполне достаточно для задач проекта. После чего сдвигается счётчик хранилища, который в сути своей хранит длину массива этого хранилища. Дальше вызывается обработчик пакетов. Этот блок функции обрабатывает все данные находящиеся в хранилище. Он просматривает, достаточно ли нам байт для проверки. То есть принято ли достаточное количество байт для полноценного пакета. После этого идёт проверка на начальный байт, так же называемый байтом инициализации. С него начинается считывание пакета, он является по сути стартовым байтом протокола. После идёт считывание команды, данных и байта контроля целостности. Если контрольная сумма верная, обработчик выбирает из списка функцию, которая должна работать с данными.

Микроконтроллер связан по портам с машинами Festo's Mechatronics. То есть машина реагирует на комбинацию выстановленных портов контроллера.

Как работает каждый блок в отдельности рассмотрено в предыдущих подкунктах.

3 Разработка приложения для ПК

При разработке программного обеспечение был выбран язык С++.

C++ (читается си-плюс-плюс)[[2]](https://ru.wikipedia.org/wiki/C%2B%2B) — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения.

Поддерживает такие [парадигмы программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как

— [процедурное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

— [объектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

— [обобщённое программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). В сравнении с его предшественником — языком [C[2]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе [x86](https://ru.wikipedia.org/wiki/X86) это [GCC](https://ru.wikipedia.org/wiki/GNU_Compiler_Collection), [Visual C++](https://ru.wikipedia.org/wiki/Visual_C%2B%2B), [Intel C++ Compiler](https://ru.wikipedia.org/wiki/Intel_C%2B%2B_Compiler), [Embarcadero (Borland) C++ Builder](https://ru.wikipedia.org/wiki/Embarcadero_C%2B%2B_Builder) и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на [Java](https://ru.wikipedia.org/wiki/Java) и [C#](https://ru.wikipedia.org/wiki/C_Sharp).

Синтаксис C++ унаследован от языка [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Одним из принципов разработки было сохранение совместимости с C. Тем не менее C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как [компиляторами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

Язык возник в начале [1980-х годов](https://ru.wikipedia.org/wiki/1980-%D0%B5_%D0%B3%D0%BE%D0%B4%D1%8B), когда сотрудник фирмы [Bell Labs](https://ru.wikipedia.org/wiki/Bell_Labs) [Бьёрн Страуструп](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%83%D1%81%D1%82%D1%80%D1%83%D0%BF,_%D0%91%D1%8C%D1%91%D1%80%D0%BD) придумал ряд усовершенствований к языку [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) под собственные нужды. Когда в конце [1970-х годов](https://ru.wikipedia.org/wiki/1970-%D0%B5_%D0%B3%D0%BE%D0%B4%D1%8B) Страуструп начал работать в Bell Labs над задачами теории очередей (в приложении к моделированию телефонных вызовов), он обнаружил, что попытки применения существующих в то время языков моделирования оказываются неэффективными, а применение высокоэффективных машинных языков слишком сложно из-за их ограниченной выразительности. Так, язык [Симула](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D0%B0) имеет такие возможности, которые были бы очень полезны для разработки большого программного обеспечения, но работает слишком медленно, а язык [BCPL](https://ru.wikipedia.org/wiki/BCPL) достаточно быстр, но слишком близок к языкам низкого уровня и не подходит для разработки большого программного обеспечения.

В книге «Дизайн и эволюция C++»  [Бьёрн Страуструп](https://ru.wikipedia.org/wiki/%D0%91%D1%8C%D1%91%D1%80%D0%BD_%D0%A1%D1%82%D1%80%D0%B0%D1%83%D1%81%D1%82%D1%80%D1%83%D0%BF)[[2]](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%83%D1%81%D1%82%D1%80%D1%83%D0%BF,_%D0%91%D1%8C%D1%91%D1%80%D0%BD) описывает принципы, которых он придерживался при проектировании C++. Эти принципы объясняют, почему C++ именно такой, какой он есть. Некоторые из них:

Получить универсальный язык со статическими типами данных, эффективностью и переносимостью языка C.

Непосредственно и всесторонне поддерживать множество стилей программирования, в том числе [процедурное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [абстракцию данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [объектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённое программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Дать программисту свободу выбора, даже если это даст ему возможность выбирать неправильно.

Максимально сохранить совместимость с C, тем самым делая возможным лёгкий переход от программирования на C.

Избежать разночтений между C и C++: любая конструкция, допустимая в обоих языках, должна в каждом из них обозначать одно и то же и приводить к одному и тому же поведению программы.

Избегать особенностей, которые зависят от платформы или не являются универсальными.

«Не платить за то, что не используется» — никакое языковое средство не должно приводить к снижению производительности программ, не использующих его.

Не требовать слишком усложнённой среды программирования.

Стандарт C++ состоит из двух основных частей: описание ядра языка и описание стандартной библиотеки.

Первое время язык развивался вне формальных рамок, спонтанно, по мере встававших перед ним задач. Развитию языка сопутствовало развитие [кросс-компилятора](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) [cfront](https://ru.wikipedia.org/w/index.php?title=Cfront&action=edit&redlink=1). Новшества в языке отражались в изменении номера версии кросс-компилятора. Эти номера версий кросс-компилятора распространялись и на сам язык, но применительно к настоящему времени речь о версиях языка C++ не ведут. Лишь в 1998 году язык стал стандартизированным.

• C++ поддерживает как комментарии в стиле C (/\* комментарий \*/), так и однострочные (// вся оставшаяся часть строки является комментарием), где // обозначает начало комментария, а ближайший последующий символ новой строки, который не предварён символом \ (либо эквивалентным ему обозначением ??/), считается окончанием комментария. Плюс этого комментария в том, что его не обязательно заканчивать, то есть обозначать окончание комментария.

• Спецификатор inline для функций. Функция, определённая внутри тела класса, является inline по умолчанию. Данный спецификатор является подсказкой компилятору и может встроить тело функции в код вместо её непосредственного вызова.

• Квалификаторы const и volatile. В отличие от С, где const обозначает только доступ на чтение, в C++ переменная с квалификатором const должна быть инициализирована. volatile используется в описании переменных и информирует компилятор, что значение данной переменной может быть изменено способом, который компилятор не в состоянии отследить. Для переменных, объявленных volatile, компилятор не должен применять средства

оптимизации, изменяющие положение переменной в памяти (например, помещающие её в регистр) или полагающиеся на неизменность значения переменной в промежутке между двумя присваиваниями ей значения. В многоядерной системе volatile помогает избегать [барьеров памяти](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D1%80%D1%8C%D0%B5%D1%80_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8) 2-го типа.

• [Пространства имён](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)#C++) (namespace).

Специальным случаем является безымянное пространство имён. Все имена, описанные в нём, доступны только в текущей единице трансляции и имеют локальное связывание. Пространство имён std содержит в себе стандартные библиотеки C++.

Для работы с памятью введены операторы new, new[], delete и delete[]. В отличие от библиотечных malloc и free, пришедших из C, данные операторы производят инициализацию объекта. Для классов это вызов конструктора, для [POD](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) типов инициализацию можно либо не проводить(new Pod;), либо провести инициализацию нулевыми значениями (new Pod(); new Pod{};).

3.1 Разработка архитектуры

Архитектура программного обеспечения ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) software architecture)[[2]](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) — совокупность важнейших решений об организации программной системы. Архитектура включает:

• выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;

• соединение выбранных элементов структуры и поведения во всё более крупные системы;

• архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

[Документирование](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) архитектуры [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) упрощает

процесс коммуникации между разработчиками, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы, [повторно использовать](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0) [компоненты](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) и [шаблоны](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) проекта в других.

Общепринятого определения «архитектуры» программного обеспечения» не существует. Так, сайт [Software Engineering Institute](https://en.wikipedia.org/wiki/Software_Engineering_Institute) приводит более 150 определений этого понятия.

Область компьютерных наук с момента своего образования столкнулась с проблемами, связанными со сложностью программных систем. Ранее проблемы сложности решались разработчиками путём правильного выбора структур данных, разработки алгоритмов и применения концепции разграничения полномочий. Хотя термин «архитектура программного обеспечения» является относительно новым для индустрии разработки ПО, фундаментальные принципы этой области неупорядоченно применялись пионерами разработки ПО начиная с середины 1980-х. Первые попытки осознать и объяснить программную архитектуру системы были полны неточностей и страдали от недостатка организованности, часто это была просто диаграмма из блоков, соединенных линиями. В 1990-е годы наблюдается попытка определить и систематизировать основные аспекты данной дисциплины. Первоначальный набор шаблонов проектирования, стилей проектирования, передового опыта, языков описания и формальная логика были разработаны в течение этого времени.

Основополагающей идеей дисциплины программной архитектуры является идея снижения сложности системы путём абстракции и разграничения полномочий. На сегодняшний день до сих пор нет согласия в отношении чёткого определения термина «архитектура программного обеспечения».

Являясь в настоящий момент своего развития дисциплиной без четких правил о «правильном» пути создания системы, проектирование архитектуры ПО все ещё является смесью науки и искусства. Аспект «искусства» заключается в том, что любая коммерческая система подразумевает наличие применения или миссии. С точки зрения пользователя программной архитектуры, программная архитектура дает направление для движения и решения задач, связанных со специальностью каждого такого пользователя, например, заинтересованного лица, разработчика ПО, группы поддержки ПО, специалиста по сопровождению ПО, специалиста по развертыванию ПО, тестера, а также конечных пользователей. В этом смысле архитектура программного обеспечения на самом деле объединяет различные точки зрения на систему. Тот факт, что эти несколько различных точек зрения могут быть объединены в архитектуре программного обеспечения, является аргументом в защиту необходимости и целесообразности создания архитектуры ПО ещё до этапа разработки ПО.

Начало архитектуре программного обеспечения как концепции было положено в научно-исследовательской работе Эдсгера Дейкстры[[2]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0,_%D0%AD%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%92%D0%B8%D0%B1%D0%B5) в 1968 году и Дэвида Парнаса в начале 1970-х. Эти ученые подчеркнули, что структура системы ПО имеет важное значение, и что построение правильной структуры — критически важно. Популярность изучения этой области возросла с начала 1990-х годов вместе с научно-исследовательской работой по исследованию архитектурных стилей (шаблонов), языков описания архитектуры, документирования архитектуры, и формальных методов.

Языки описания архитектуры (ADLS)[[2]](https://en.wikipedia.org/wiki/Activities_of_daily_living) используются для описания архитектуры программного обеспечения. Различными организациями было разработано несколько различных ADLS, в том числе AADL (стандарт SAE), Wright (разработан в университете Carnegie Mellon), Acme (разработан в университете Carnegie Mellon), xADL (разработан в UCI), Darwin (разработан в Imperial College в Лондоне), DAOP-ADL (разработан в Университете Малаги), а также ByADL (Университет L’Aquila, Италия). Общими элементами для всех этих языков являются понятия компонента, коннектора и конфигурации. Также, помимо специализированных языков, для описания архитектуры часто используется унифицированный язык моделирования [UML](https://ru.wikipedia.org/wiki/UML).

Развертывание (англ. deployment view) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов

Внедрение (англ. implementation view) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т.д.)

Распределение работы (англ. work assignment view) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). Каждый шаблон имеет свои задачи и свои недостатки.

Примеры архитектурных шаблонов:

Многоуровневый шаблон (Layered pattern)[[2]](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0). Система разбивается на уровни, которые на диаграмме изображаются один над другим. Каждый уровень может вызывать только уровень на 1 ниже него. Таким образом разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Недостатками данного подхода являются усложнение системы и снижение производительности.

Шаблон посредника (Broker pattern)[[2]](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D0%BA_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.

Шаблон [«Модель-Представление-Контроллер»](https://ru.wikipedia.org/wiki/Model-View-Controller) (Model-View-Controller pattern). Т.к. требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне Model-View-Controller (MVC) интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты.

В данном дипломном проекте используется модульный вид архитектуры. Он показывает систему как структуру из различных программных блоков. А именно вид уровней.

3.2 Выбор и описание необходимых наборов библиотек Qt Framework

QTextEdit[[3]](https://doc.qt.io/qt-5/qtextedit.html) будет отвечать за консоль.

Класс QTextEdit предоставляет виджет, который используется для редактирования и отображения как простого, так и форматированного текста.

QTextEdit - это расширенная программа просмотра / редактирования, поддерживающая форматирование текста с использованием тегов в стиле HTML. Он оптимизирован для обработки больших документов и быстрого реагирования на ввод данных пользователем.

QTextEdit работает с абзацами и символами. Абзац - это отформатированная строка, которая переносится по словам, чтобы соответствовать ширине виджета. По умолчанию при чтении обычного текста одна новая строка обозначает абзац. Документ состоит из нуля или более абзацев. Слова в абзаце выравниваются в соответствии с выравниванием абзаца. Абзацы разделяются жестким переносом строки. Каждый символ в абзаце имеет свои собственные атрибуты, например шрифт и цвет.

QTextEdit может отображать изображения, списки и таблицы. Если текст слишком велик для просмотра в области просмотра редактора текста, появятся полосы прокрутки. Редактор текста может загружать как обычный текст, так и файлы HTML[[3]](https://wiki.qt.io/Handling_HTML) (подмножество HTML 3.2 и 4).

Поддержка форматированного текста в Qt разработана, чтобы обеспечить быстрый, переносимый и эффективный способ добавления разумных средств интерактивной справки к приложениям и обеспечить основу для редакторов форматированного текста.

На рисунке 3.1 представлен ранний вид приложения. Область текстового редактора создана благодаря QTextEdit.

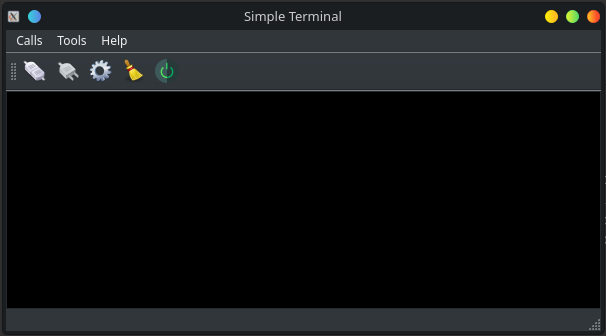


Рисунок 3.1 – Ранний вид программы

QString[[3]](https://doc.qt.io/qt-5/qstring.html) был выбран для удобной работы со строковыми типами данных.

Класс QString предоставляет строку символов Unicode. QString хранит строку 16-битных [QChar](http://doc.crossplatform.ru/qt/4.6.x/qchar.html)[3], где каждому [QChar](http://doc.crossplatform.ru/qt/4.6.x/qchar.html)(библиотека Qt) соответствует один символ Unicode 4.0. (Символы Unicode со значениями кодов больше 65535 хранятся с использованием суррогатных пар, т.е. двух последовательных [QChar](http://doc.crossplatform.ru/qt/4.6.x/qchar.html).)

[Unicode](http://doc.crossplatform.ru/qt/4.6.x/unicode.html#unicode)[[3]](https://wiki.qt.io/Strings_and_encodings_in_Qt) - это международный стандарт, который поддерживает большинство использующихся сегодня систем письменности. Это расширение ASCII и Latin-1, где все символы ASCII/Latin-1 доступны на позициях с тем же кодом.

Внутри QString использует [неявное совместное использование данных](http://doc.crossplatform.ru/qt/4.6.x/implicit-sharing.html) (копирование-при-записи), чтобы уменьшить использование памяти и избежать ненужного копирования данных. Это также позволяет снизить накладные расходы, свойственные хранению 16-битных символов вместо 8-битных.

В дополнение к QString Qt также предоставляет класс [QByteArray](http://doc.crossplatform.ru/qt/4.6.x/qbytearray.html) для хранения сырых байт и традиционных нультерминальных строк. В большинстве случаев QString - необходимый для использования класс. Он используется во всем API Qt, а поддержка Unicode гарантирует, что ваши приложения можно будет легко перевести на другой язык, если в какой-то момент вы захотите увеличить их рынок распространения. Два основных случая, когда уместно использование [QByteArray](http://doc.crossplatform.ru/qt/4.6.x/qbytearray.html): когда вам необходимо хранить сырые двоичные данные и когда критично использование памяти (например, в [Qt для встраиваемых Linux-систем](http://doc.crossplatform.ru/qt/4.6.x/qt-embedded-linux.html)).

Именно поэтому там так же необходим класс QByteArray для удобства работы с пакетами байт.

Класс QByteArray[[3]](https://doc.qt.io/qt-5/qbytearray.html) предоставляет массив байт.

QByteArray может быть использован для хранения как сырых байт (включая и '\0'), так и традиционных 8-битных нуль-терминированых строк. Использование QByteArray более удобно, чем использование const char \*. Здесь всегда гарантировано, что данные завершаются '\0' и используется [неявное совместное использование данных](http://doc.crossplatform.ru/qt/4.5.0/shared.html#implicit-sharing) (copy-on-write) для экономии памяти и избегания ненужного копирования данных.

Два основных случая, где QByteArray подходит - это, когда вам нужно сохранить сырые двоичные данные, и когда критично сохранение памяти (например, в Qt для Embedded Linux). Причина выбора данной библиотеки хранение двоичных данных.

QByteArray делает полную копию данных const char \*, так что вы можете модифицировать их, не получив в дальнейшем побочных эффектов.

Класс QMainWindow предоставляет главное окно приложения. Главное

окно предоставляет структуру для создания пользовательского интерфейса приложения. Qt имеет класс QMainWindow[[3]](https://doc.qt.io/qt-5/qmainwindow.html) и [связанные с ним классы](http://doc.crossplatform.ru/qt/4.7.x/mainwindow-classes.html) для управления главным окном. QMainWindow имеет собственный компоновщик, в который вы можете добавлять [QToolBar](http://doc.crossplatform.ru/qt/4.7.x/qtoolbar.html)'ы, [QDockWidget](http://doc.crossplatform.ru/qt/4.7.x/qdockwidget.html)'ы, [QMenuBar](http://doc.crossplatform.ru/qt/4.7.x/qmenubar.html), и [QStatusBar](http://doc.crossplatform.ru/qt/4.7.x/qstatusbar.html). Компоновщик имеет центральную область, которая может быть занята любым виджетом.

QSerialPort[[3]](https://doc.qt.io/qt-5/qserialport.html) используется для передачи данных по порту связи. Он принимает на вход блоки данных в виде байт, хранимых нами в QByteArray.

QObject[[3]](https://doc.qt.io/qt-5/qobject.html) используется как общий класс для работы с другими классами Qt.

3.3 Разработка и обоснование стркутуры отсылаемых пакетов

На рисунке 3.2 представлен пример пакета протокола.

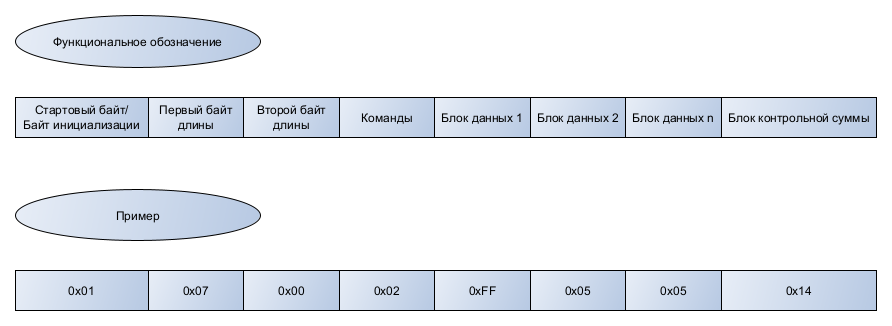


Рисунок 3.2 – Пример пакета протокола

Разберём структуру этого пакета. Первый байт является стартовым байтом, 0х01. Он сигнализирует о том, что этот байт является началом пакета и отсюда нам следует его считывать. Дальше идут 2 байта, отвечающие за размер. Отсюда узнаём размер пришедшего пакета. Узнаём длину пакеты без стартового байта. Таким образом узнаём точку старта и точку конца пакета.

Дальше идёт блок команды. Он говорит о том, каким образом следует обработать

данные и что с ними делать. Дальше до последнего блока идут блоки данных. Самый последний блок, это CRC – контрольная сумма. По ней определяем конечную целостность пакета.

Ниже представлен код обработчика данных.

Код обработчика данных:

*void QSerialProtocol::slotReadyRead()*

*{*

*qDebug() << "slotReadyRead " << debugValue;*

*debugValue++;*

*QByteArray Buf = serial->readAll();*

*data.push\_back(Buf);*

*while (data.length() > 0)*

*{*

*if (data[0]==0xFF)*

*{*

*if (data.length()>3)*

*{ quint16 length = DeSerializationQuint16(data.mid(1,2));//data[2]<<8 + data[1];*

*if (data.length() == length+1)*

*{*

*QByteArray forCRC;*

*forCRC = data.mid(0,length);*

*if (data[length+1] == getControlSumm(&forCRC))*

*{*

*qDebug() << "govno";*

*emit signalSerialRawData(data.mid(0,length));*

*handlerPacket(data.mid(3,1));*

*data.remove(0,length); //Удаляем пакет из буфера*

*//break;*

*} else*

*{*

*data.remove(0,1); //Удаляем заголовок неверного пакета*

*break;*

*}*

*} else break;*

*} else break;*

*} else data.remove(0,1);*

*}*

*}*

4 Разработка функциональной схемы модуля работы с датчиками и органами управления Festo's Mechatronics

Функциональную схему модуля вы можете найти на рисунке 4.1

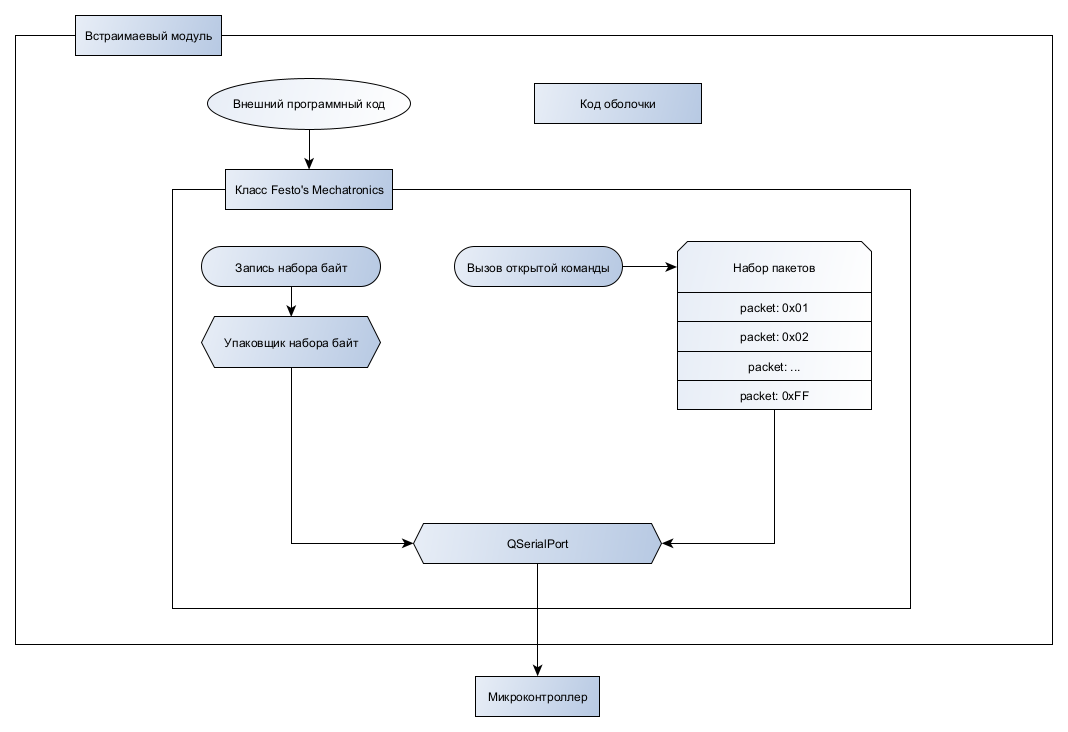


Рисунок 4.1 – Функциональная схема программного модуля

Данный модуль представлен в виде функциональной схемы на рисунке 4.1 Он подключается к программе пользователя посредством простой директивы include.

Рассмотрим каждый из логических блоков в отдельности.

1. Внешний программный код. Это часть блока отвечает за вызов методов класса FestoMashine
2. Код оболочки. Логически представлен, т.к. необходим для пояснения, что данный модуль будет в первую очередь служить как дополнение к другому рабочему коду.
3. Вызов открытой команды. Один из способов взаимодействия с модулем. Вызываем открытые методы класса, выбирая нужную команду, в соответствии с принципами инкапсуляции. Пакет команды отсылается через QSerialPort микроконтроллеру
4. Блок команд. Представляет из себя библиотеку упакованных и пронумерованных пакетов
5. Запись набора байт. Для расширямости приложения оставляем возможность пользователю записать любой выбранный им набор байт.
6. Упаковщик занимается тем, что преобразует входной набор байт в соответствии с правилами протокола
7. QSerialPort отвечает за пересылку байт через соединению микроконтроллеру. Представляет из себя простой класс QT Framework

Логика модуля была продумана для того что бы соответствовать следующим требованиям:

• расширяемость

• инкапсуляция интерфейса

• удобочитаемость

Именно исходя из этих требований было разработано наше программное обеспечение в виду статического модуля.

5 Разработка электрической принципиальной схемы модуля работы с датчиками и органами управления

На рисунке 5.1 представлена электрическая принципиальная схема микроконтроллера ATmega328P-AU

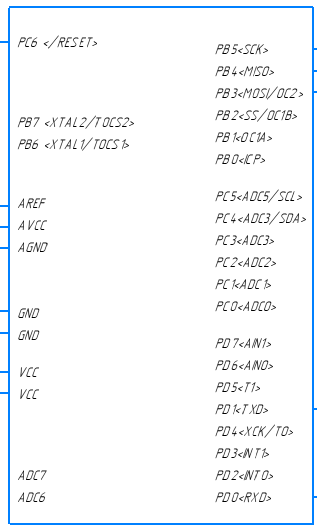


Рисунок 5.1 – Элемент ATmega328P-AU на электрическое

принципиальной схеме

Это 8-битный микроконтроллер который удовлетворяем требованием проекта. Полная техническая документация на официальном ресурсе компании ATmega, там можем посмотреть распиновку, рабочую частоту и т.д.

В приложении D находится полная электрическая принципиальная схема системы. На ней микроконтроллер соединён с ISP блоком. Этот блок является программатором. Через него меняем прошивку микроконтроллера.

Стоит отметить, что так как прошивка происходит через пин RESET, существует возможность производить перепрошивку и отладку одновременно, что существенно снижает объём работы.

Блок USB отвечает за логическое соединение ПК с микроконтроллером. Разъёмы RXD TXD соединяются с противоположными на микроконтроллере, по принципу RXD/TXD TXD/RXD.

При отсутствии подключения внешнего питания, питание может производиться через пин VCC USB разъёма, т.к. его мощности хватает для подпитки мироконтроллера.

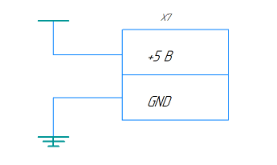


Рисунок 5.2 --- Питание системы

Присутствует на схеме, в проекте подключается с помощью разобранного USB кабеля характеристиками +5В/2А.

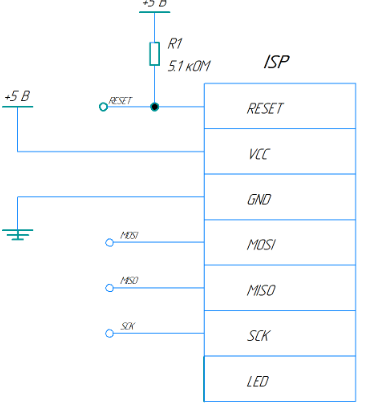


Рисунок 5.3 --- Схема подключения программатора к системе

Схема подключения программатора. В данной схеме порт RESET связан с портом RESET микроконтроллера. VCC подключено к питанию.

MOSI/MISO/SCK подключены к одноимённым портам микроконтроллера.

Порт LED получает питание через порт VCC контроллера, диод показывает режим работы программатора.

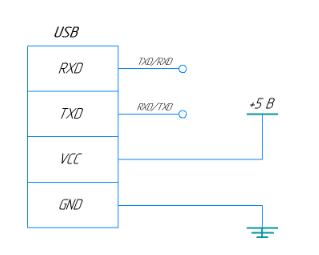


Рисунок 5.4 --- Схема подключения USB

На рисунке 5.4 схема подключения USB кабеля к микроконтроллеру.

Порты VCC и GND подключены к одноимённым линиям.

Порты RXD TXD подключены к противоположным портам микроконтроллера по принципу RXD/TXD TXD/RXD.

6 Технико-экономическое обоснование целесообразности разработки программного продукта и оценка его конкурентоспособности

6.1 Технико-экономическое обоснование целесообразности разработки программного продукта

Разработка данной системы необходима для упрощения процесса работы с пневмо-машинами(далее ПМ) компании Festo Mechantronics, в частности тем ПМ, который закуплены кафедрой «Промышленная электроника». Данные машины предназначены для обучения студентов кафедры, они симулируют работы реальных аппаратов, в том числе на заводе, предприятии, компании и т.л.

Данный программный продукт (далее ПП), благодаря своему удобству, позволяет гораздо проще и быстрее организовывать учебный процесс. Также, благодаря своей внутренней организации, он позволяет с наименьшим количеством проблем добавить в модуль основные принципы работы с теоретически новой машиной, добавляя “команды общения” с этой машиной в уже существующий API(Application Programming Interface).

Отдельные функции данного ПП не являются новыми, однако их объединение в данном продукте обладает новизной. Самым главным отличием данного ПП является то, что приложение способно работать абсолютно на разных операционных системах(далее ОС). Однако, не поддерживаются системы, которые официально не включает в себя QT 5.15 (Qt Company).

Для разработки данного ПП используется только ПО распространяемое по свободной лицензии. Это необходимо для того, чтобы в случае, если один из компонентов системы, по каким либо причинам станет платным, у кафедры не возникли дополнительные издержки. Используются такие технологии и ПП, как Qt, C++, AtmelStudio, openClibs. Данный ПП предназначен для различного рода работы с участием компьютером или микроконтроллером(хотя нельзя отрицать, что при доработке данного ПП, его функциональность может быть расширена, а работа ещё более автоматизирован).

Таблица 6.1 – Основные показатели сравнительного анализа вариантов ПП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель | Единицы измерения | Варианты | | Результаты сравнения: (+/-) |
| Festo’s Mechtronics programm. | Проектируемый |
| Работа на ОС Windows XP | Наличие | Да | Да | 0 |
| Работа в оффлайн режиме | Наличие | Да | Да | 0 |
| Работы на ОС Windows 8 | Наличие | Нет | Да | + |
| Работы на ОС Windows 10 | Наличие | Нет | Да | + |
| Переносимость как модуля | Наличие | Нет | Да | + |
| Работа на ОС Linux GNU | Наличие | Нет | Да | + |
| Обработчик граф. схемы | Наличие | Да | Нет | - |
| Возможность добавления новых машин | Наличие | Нет | Да | + |

6.2 Оценка конкурентности программного продукта

Техническая прогрессивность разрабатываемого ПП определяется коэффициентом эквивалентности (Кэк). Расчет этого коэффициента осуществляется путем сравнения технического уровня товара-конкурента и разрабатываемого ПП по отношению к эталонному уровню ПП данного

направления с использованием формулы (6.1):

(6.1)

где

Кт.н, Кт.б – коэффициенты технического уровня нового и базисного ПП, которые можно рассчитать по формуле (6.2):

(6.2)

где

*β* – коэффициент весомости *i*-го технического параметра;

*n* – число параметров;

Pi, Pэ –численное значение *i*-го технического параметра сравниваемого ПП и эталона.

Расчет данного коэффициента приведен в таблице 6.2.

Таблица 6.2 – Расчет коэффициентов эквивалентности

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Наименование параметра | Вес параметра, β | Значение параметра | | | Pб/Pэ | Pн/Pэ | β·Pб/Pэ | β·Pн/Pэ |
| Pб | Pн | Pэ |
| Объём памяти, необходимый для приложения, Мб | 0.7 | 400 | 500 | 450 | 0.88 | 1.11 | 0.616 | 0.777 |
| Скорость обращения команды. мс | 0.2 | 0.3 | 0.2 | 0.15 | 2 | 1.33 | 0.4 | 0.266 |
| Отказы | 0.1 | 2 | 1 | 1 | 0.5 | 1 | 0.05 | 0.1 |
| Итого | | | | | | | 1.066 | 1.143 |
| Коэффициент эквивалентности | | | | | | | 1.07 | |

Далее рассчитывается коэффициент изменения функциональных возможностей (*Кф.в*) нового ПП по формуле (6.3):

(6.3)

где

*Кф.в.н, Кф.в.б* – балльная оценка неизмеримых показателей нового и базового изделия соответственно.

Расчет коэффициента изменения функциональных возможностей нового ПП приведен в таблице 6.3.

Таблица 6.3 – Расчет коэффициента изменения функциональных возможностей

|  |  |  |
| --- | --- | --- |
| Наименование параметра | Бальная оценка базового ПП | Бальная оценка нового ПП |
| Работа на ОС Windows XP | 4 | 4 |
| Работа в оффлайн режиме | 5 | 5 |
| Работа на ОС Windows 8 | 0 | 4 |
| Работа на ОС Windows 10 | 0 | 5 |
| Переносимость как модуля | 0 | 5 |
| Работа на ОС Linux GNU | 0 | 5 |
| Обработчик граф. схемы | 5 | 0 |
| Возможность добавления новых машин | 0 | 4 |
| Итого | 14 | 32 |
| Коэффициент функциональных возможностей | 2.29 | |

Конкурентоспособность нового ПП по отношению к базовому можно оценить с помощью интегрального коэффициента конкурентоспособности, учитывающего все ранее рассчитанные показатели. Он рассчитывается по формуле (6.4):

(6.4)

где

*Кн* – коэффициент соответствия нового ПП нормативам (*Кн* = 1);

*Кц* – коэффициент цены потребления.

Коэффициент цены потребления рассчитывается как отношение договорной цены нового ПП к договорной цене базового.

Расчет данного коэффициента приведен в таблице 6.4.

Таблица 6.4 – Расчет уровня конкурентоспособности нового ПП

|  |  |
| --- | --- |
| Коэффициенты | Значение |
| Коэффициент эквивалентности (Кэ) | 1.07 |
| Коэффициент изменения функциональных возможностей (Кф.в) | 2.29 |
| Коэффициент соответствия нормотивам (Кн) | 1 |
| Коэффициент цены потребления (Кц) | 1 |
| Интегральный коэффициент конкурентоспособности | 2.45 |

Интегральный коэффициент конкурентоспособности (*Ки*) больше единицы, следовательно новый ПП является более конкурентоспособным по сравнению с базовым.

6.3 Определение общего объема функций программного обеспечения

В качестве единицы измерения объема программного обеспечения (ПО) используется строка исходного кода (LOC). Общий объем ПО (*Vo*) определяется исходя из количества и объема функций, реализуемых программой, по формуле (6.5):

(6.5)

где

*Vi* – объем отдельной функции ПО;

*n* – общее число функций.

Анализируя разработанный ПП, уточненный объем ПО (*Vу*) определяем по формуле (6.6):

(6.6)

где

*Vу*i – уточненный объем отдельной функции ПО в строках исходного кода.

Сравнение исходного и уточненных объемов строк исходного кода представлен в таблице 6.5.

Таблица 6.5 – Перечень и объем функций программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Код функций | Наименование (содержание) функций | Объём строк исходного кода (LOC) | |
| С использованием среды разработки приложений QtCreator | |
| По каталогу (Vo) | Уточненный (Vу) |
| 1. Ввод, анализ входной информации, генерация кодов и процессор входного языка | | | |
| 101 | Организация ввода информации | 150 | 20 |
| 102 | Контроль, предварительная обработка и ввод информации | 550 | 50 |
| 105 | Преобразование входного языка в машинные команды (транслятор, препроцессор, макрогенератор) | 4200 | 145 |
| 109 | Управление вводом-выводом | 2400 | 30 |
| 2. Генерация программ и ПО, а также настройка программного обеспечения | | | |
| 401 | Генерация рабочих программ | 3360 | 230 |
| 3. Управление ПО, компонентами ПО и внешними устройствами | | | |
| 506 | Обработка ошибочных сбойных ситуаций | 1720 | 190 |
| 507 | Обеспечения интерфейса между компонентами | 1820 | 40 |

Окончание таблицы 6.5

|  |  |  |  |
| --- | --- | --- | --- |
| 4. Тестирование, проведение тестовых испытаний прикладных программ, вспомогательные программные функции | | | |
| 602 | Вспомогательные и сервисные программы | 580 | 1010 |
| 5. Расчетные задачи | | | |
| 703 | Расчет показателей | 460 | 670 |
| Итого | | 18740 | 3225 |

В связи с использованием более совершенных средств разработки, объемы функций были уменьшены и уточненный объем ПП составил 3225 строк кода вместо 18740.

6.4 Расчет поправочных коэффициентов, учитывающих организационно-технические условия разработки программного обеспечения

Все ПО в зависимости от их характеристик подразделяется на три категории сложности (таблица 6.6).

Таблица 6.6 – Характеристика категорий сложности ПО

|  |  |
| --- | --- |
| Категория сложности | Характеристика программного обеспечения |
| 1 | ПО, обладающее одно или несколькими из следующих характеристик:  1) Наличие сложного интеллектуального языкового интерфейса с пользователем  2) Обеспечение телекоммуникационной обработки данных и управление удаленными объектами  3) Обеспечение существенного распараллеливания вычислений  4) Криптография и другие методы защиты информации |
| 2 | ПО, обладающее одной или несколькими из следующих характеристик:  1) Моделирование объектов и процессов  2) Обеспечение настройки ПО на изменение структур входных и выходных данных  3) Обеспечение переносимости ПО  4) Реализации особо сложных инженерных и научных расчетов |
| 3 | ПО, не обладающее перечисленными характеристиками |

Разрабатываемое программное обеспечение можно отнести ко второй категории сложности, так как обеспечивается переносимость разрабатываемого ПО на различные программные и аппаратные платформы.

На основании принятого к расчету (уточненного) объема и категории сложности ПО определяем нормативную трудоемкость ПО (*Тн*) выполняемых работ (таблица 6.7).

Таблица 6.7 – Нормативная трудоемкость на разработку ПО (*Тн*)

|  |  |  |
| --- | --- | --- |
| Уточненный объем ПО | 2-ая категория сложности | Номер нормы |
| 3225 | 175 | 41 |

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (*Кс*), который рассчитывается по формуле (6.7):

(6.7)

где

*Кi* – коэффициент, соответствующий степени повышения сложности ПО (таблица 6.8);

*n* – количество учитываемых характеристик.

Таблица 6.8 – Коэффициент повышения сложности ПО (*Кi*)

|  |  |  |
| --- | --- | --- |
| № п/п | Характеристика повышения сложности ПО | Значение Кс |
| 1 | Функционирование ПО в расширенной опрационной среде (связь с другим ПО) | 0.08 |
| 2 | Интерактивный доступ | 0.06 |
| 3 | Обеспечение хранение, ведения, поиска данных в сложных структурах | 0.07 |
| 4 | Наличие у ПО одновременно нескольких характеристик в таблице 6.6 |  |

Окончание таблицы 6.8

|  |  |  |
| --- | --- | --- |
| 4.1 | Две характеристики | 0.12 |
| 4.2 | Три характеристики | 0.18 |
| 4.3 | Свыше трех характеристик | 0.26 |

Новизна разрабатываемого ПО определяется путем экспертной оценки данных, полученных при сравнении характеристик разрабатываемого ПО с имеющимися аналогами.

Влияние фактора новизны на трудоемкость учтем путем умножения нормативной трудоемкости на коэффициент, учитывающий новизну ПО (*Кн*). Коэффициенты, учитывающие новизну ПО представлены в таблице 6.9.

Таблица 6.9 – Коэффициенты, учитывающие новизну ПО (*Кн*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория новизны ПО | Степень новизны | Использование | | Значение Кн |
| На основе нового типа ПК | В среде новой ОС |
| А | Принципиально новое ПО, не имеющее аналогов | + | + | 1.58 |
| - | + | 1.44 |
| + | - | 1.1 |
| - | - | 1 |

Окончание таблицы 6.9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Б | ПО, являющееся развитием определенного параметрического ряда ПО | + | + | 1 |
| - | + | 0.81 |
| + | - | 0.72 |
| В | ПО, являющееся развитием определенного параметрического ряда ПО, разработанных для ранее освоенных типов конфигурации ПК и ОС | - | - | 0.63 |

Разрабатываемое ПО можно отнести к категории Б, а значение *Кн* = 1.

Современные технологии разработки компьютерных программ предусматривают широкое использование коробочных продуктов (пакетов, модулей, объектов). Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО. Коэффициенты, учитывающие степень использования стандартных модулей представлены в таблице 6.10.

Таблица 6.10 – Коэффициенты, учитывающие степень использования стандартных модулей (*Кт*)

|  |  |
| --- | --- |
| Степень охвата реализуемых функциями разрабатываемого ПО стандартными модулями | Значение Кт |
| От 60% и выше | 0.55 |
| От 40 до 60% | 0.65 |
| От 20 до 40% | 0.77 |

Окончание таблицы 6.10

|  |  |
| --- | --- |
| До 20% | 0.9 |
| Не используются стандартные модули для реализации функций разрабатываемого ПО | 1 |

В разрабатываемом ПО объем стандартных модулей составляет от 40 до 60 процентов, что соответствует значению коэффициента *Кт* = 0.65.

Коэффициент (*Ку.р*) определяется средством разработки ПО и характером ОС (таблица 6.11)

Таблица 6.11 – Коэффициенты, учитывающие средства разработки ПО (*Кур*)

|  |  |  |  |
| --- | --- | --- | --- |
| Средства разработки ПО | Значения Кур в зависимости от характера операционной среды | | |
| Стационарные  ОС | Функционирование  ПО в сетях | |
| локальных | глобальных |
| Процедурные языки высокого уровня | 1 | 1.2 | 1.3 |

Программный комплекс разработан на языке высокого уровня для стационарной ОС, что соответствует значению коэффициента *Ку.р* = 1

Значение коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО, определяется с учетом установленной категории новизны ПО. Коэффициенты удельных весов трудоемкости стадий разработки ПО приведены в таблице 6.12

Таблица 6.12 – Коэффициент удельных весов трудоемкости стадий разработки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория новизны ПО | Без применения CASE-технологий | | | | |
| Стадии разработки ПО | | | | |
| ТЗ | ЭП | ТП | РП | ВН |
| Значения коэффициентов | | | | |
| Ктз | Кэп | Ктп | Крп | Квн |
| Б | 0.1 | 0.2 | 0.3 | 0.3 | 0.1 |

6.5 Расчет трудоемкости выполняемых работ по стадиям разработки программного обеспечения

Нормативная трудоемкость ПО (*Тн*) выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: повышения сложности ПО (*Кс*), учитывающих новизну ПО (*Кн*), учитывающих степень использования стандартных модулей (*Кт*), средства разработки ПО (*Ку.р*) и определяются по формулам:

для стадии ТЗ по формуле (6.8):

(6.8)

для стадии ЭП по формуле (6.9):

(6.9)

для стадии ТП по формуле (6.10):

(6.10)

для стадии РП по формуле (6.11):

(6.11)

для стадии ВН по формуле (6.12):

(6.12)

где

*Кт.з*, *Кэ.п*, *Кт.п*, *Кр.п* и *Кв.н* – значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО.

Коэффициенты *Кс*, *Кн*, *Ку.р* вводятся на всех стадиях разработки, а коэффициент *Кт* вводится только на стадии РП.

6.6 Расчет общей трудоемкости разработки программного обеспечения

Общая трудоемкость разработки ПО (То) определяется суммированием нормативной (скорректированной) трудоемкости ПО по стадиям разработки по формуле (6.13):

(6.13)

где

*Тyi* – нормативная (скорректированная) трудоемкость разработки ПО на *i*-й стадии, чел.-дн.;

*n* – количество стадий разработки.

Результаты расчетов нормативной и скорректированной трудоемкости ПО по стадиям разработки и общая трудоемкость разработки ПО представлены в таблице 6.13

Таблица 6.13 – Расчет общей трудоемкости разработки

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № П/П | Показатели | Стадии разработки | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| 1 | Общий объем ПО (Vо), количество с строк LOC | - | - | - | - | - | 18740 |
| 2 | Общий уточненный объем ПО (Vу), количество строк LOC | - | - | - | - | - | 3225 |
| 3 | Категория сложности разрабатываемого ПО | - | - | - | - | - | 2 |
| 4 | Нормативная трудоемкость разработки ПО (Тн), чел.-дн. | - | - | - | - | - | 175 |
| 5 | Коэффициент повышения сложности ПО (Кс) | 1.33 | 1.33 | 1.33 | 1.33 | 1.33 | - |
| 6 | Коэффициент, учитывающий новизну ПО (Кн) | 1 | 1 | 1 | 1 | 1 | - |

Окончание таблицы 6.13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | Коэффициент, учитывающий степень использования стандартных модулей (Кт) | - | - | - | 0.65 | - | - |
| 8 | Коэффициент, учитывающий средства разработки (Ку.р) | 1 | 1 | 1 | 1 | 1 | - |
| 9 | Коэффициенты удельных весов  трудоемкости стадий разработки ПО (Кт.з, Кэ.п, Кт.п, Кр.п, Кв.н) | 0.1 | 0.2 | 0.3 | 0.3 | 0.1 | 1 |
| 10 | Распределение нормативной  трудоемкости ПО по стадиям, чел.-дн. | 18 | 35 | 53 | 53 | 18 | 177 |
| 11 | Распределение скорректированной (с учетом Кс, Кн, Кт, Ку.р) трудоемкости ПО по стадиям, чел.-дн. | 24 | 56 | 83 | 54 | 28 | 211 |
| 12 | Общая трудоемкость разработки ПО (То), чел.-дн. | - | - | - | - | - | 211 |

6.7 Расчет затрат на разработку (себестоимости) программного продукта

В состав затрат на разработку ПП входят следующие статьи расходов:

– затраты труда на создание ПП (затраты по основной, дополнительной заработной плате и соответствующие отчисления) (*Зтр*);

– затраты на изготовление эталонного экземпляра (*Зэт*);

– затраты на технологию (затраты на приобретение и освоение программных средств, используемых при разработке ПП; затраты на ПО, используемое как эталон) (*Зтех*);

– затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) (*Зм.в*);

– затраты на материалы (информационные носители) (*Змат*);

– затраты на энергию, на использование каналов связи (для отдельных видов);

– общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) (*Зобщ.пр*);

– непроизводственные (коммерческие) расходы (затраты, связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) (*Знепр*).

Основные отличия в расчете затрат на программную продукцию от традиционных продуктов: большая динамичность и большая неопределенность результата в заданные сроки, особенно на ранних стадиях разработки; отсутствие сложившейся технологической базы для создания программной продукции, что приводит к разнообразию приемов и методов разработки при создании схожей продукции различными разработчиками; разнообразие предметной области.

Суммарные затраты на разработку программного обеспечения (*Зр*) определяются по формуле (6.14):

(6.14)

### ***6.7.1 Расчет затрат на оплату труда разработчиков***

Расходы на оплату труда разработчиков с отчислениями (*Зтр*) определяются с использованием формул (6.15)–(6.18) :

(6.15)

где

*ЗПосн* – основная заработная плата разработчиков, руб.;

*ЗПдоп* – дополнительная заработная плата разработчиков, руб.;

*ОТЧзп* – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

(6.16)

где

*Сср.час* – средняя часовая тарифная ставка, руб./ч;

*То* – общая трудоемкость разработки, чел.-ч;

*Кув* – коэффициент, учитывающий доплаты стимулирующего характера (*Кув* = 1,5–2,0).

(6.17)

где

*Счi* – часовая тарифная ставка разработчика i-й категории, руб./ч;

*ni* – количество разработчиков i-й категории.

Расчет данной величины представлен в таблице 6.14

Таблица 6.14 – Расчет средней часовой тарифной ставки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Категория  исполнитель-  разработчик (i) | Кол-во чел(ni) | Тарифный  коэф(Тki) | Месячная тарифная ставка (Сmi) | Часовая тарифная ставка (Счi) | Средняя часовая тарифная ставка (Сср.ч) |
| 1 | Инженер-программист 1 категории | 1 | 2.81 | 116.44 | 0.69 | - |
| 2 | Итого | 1 | - | - | - | 0.69 |

Для определения основной заработной платы (*ЗПосн*) исполнителей необходим расчет месячных (*Cмi*) и часовых (*Cчi*) тарифных ставок. Месячная тарифная ставка каждого исполнителя определяется путем умножения действующей месячной тарифной ставки первого разряда (*Cм1*) на тарифный коэффициент (*Ткi*), который соответствует установленному тарифному разряду[[4]](http://mintrud.gov.by/ru/min_zar_plata_2019/).

Часовая тарифная ставка (*Счi*) вычисляется путем деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне фонд рабочего времени 168 часов (*Fмес*) по формуле (6.18):

(6.18)

Рассчитаем *ЗПосн*, учитывая, что действующая месячная тарифная ставка первого разряда *Cм1* = 41 рублей [[2](https://myfin.by/wiki/term/tarifnaya-stavka-pervogo-razryada)], а тарифный коэффициент *Ткi* = 2.81.

Дополнительная заработная плата *ЗПдоп* разработчиков включает выплаты, предусмотренные законодательством о труде, и определяется по формуле (6.19).

(6.19)

где

*Ндоп* – норматив на дополнительную заработную плату разработчиков.

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле (6.20).

(6.20)

где

*Hзп* – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы (*Hз.п* = 34 %).

Таким образом можно рассчитать затраты на оплату труда (*Зтр*) согласно формуле (6.14)

### ***6.7.2 Расчет затрат на машинное время***

Затраты машинного времени (*Зм.в*) определяются по формуле (6.21):

(6.21)

где

*Сч* – стоимость одного часа машинного времени, руб./ч.;

*Кт* – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ; *Кт* = 1;

*tэвм* – машинное время ЭВМ, необходимое для разработки и отладки проекта, ч.

Стоимость одного машино-часа определяется по формуле (6.22):

(6.22)

где

*ЗПоб* – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, руб./год;

*Зар* – стоимость аренды помещения под размещение вычислительной техники, руб./год;

*Зам* – амортизационные отчисления за год, руб./год;

*Зэ.п* – затраты на электроэнергию, руб./год;

*Зв.м* – затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), руб./год;

*Зт.р* – затраты на текущий и профилактический ремонт ЭВМ, руб./год;

*Зпр* – прочие затраты, связанные с эксплуатацией ПЭВМ, руб./год;

*Fэвм* – действительный фонд времени работы ЭВМ, час/год.

Так как в процессе разработки необходимо только две единицы вычислительной техники, то обслуживающий персонал наниматься не будет *ЗПоб* = 0 руб.

Годовые затраты на аренду помещения (*Зар*) определяются по формуле (6.23):

(6.23)

где

*Cар* – средняя годовая ставка арендных платежей, руб./м2 ; [[3](https://myfin.by/wiki/term/arendnaya-stavka)]

*S* – площадь помещения, м2.

Сумма годовых амортизационных отчислений (*Зам*) определяется по формуле (6.24):

(6.24)

где

*Зпрi* – затраты на приобретение (стоимость) *i*-го вида основных

фондов, руб.;

*Кдоп* – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, *Кдоп* = 12–13 % от *Зпр*;

*Зпрi*/(1 + *Кдоп*) – балансовая стоимость ЭВМ, руб.;

*mi* – количество оборудования *i*-го вида;

*Намi* – норма амортизации для *i*-го вида ЭВМ, %.

Расчет расходов на амортизацию производиться с использованием таблицы 6.15.

Таблица 6.15 – Расчетов расходов на амортизацию

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Наименование | Кол-во | Стоимость приобретения | Балансовая стоимость единицы ЭВМ | Суммарная стоимость ЭВМ, | Норма амортизации | Сумма амортизационных отчислений (), руб. |
| 1 | Ноутбук Lenovo Ideapad 3 15ADA05 [[3](https://5element.by/products/682221-noutbuk-lenovo-ideapad-3-15ada05-81w100apre?utm_source=google_shopping&utm_medium=cpc&utm_campaign=newstrategy&utm_content=site_SDC&utm_term=_u_&gclid=CjwKCAjwy42FBhB2EiwAJY0yQiLV9oQGqho7KCNhUgrNj1vPjEZ8xwrfQdKqf9zz4akJau3PKzBpoxoCaHsQAvD_BwE)] | 1 | 1197 | 1340,64 | 1340,64 | 20 | 268,12 |
| 2 | Микроконтроллер ATmega328P-PU [[4](https://www.chipdip.by/product/atmega328p-pu)] | 3 | 6.7 | 7.5 | 22.5 | 20 | 4.5 |
| Итого | | 4 | - | - | 1362,5 | - | 272,62 |

Стоимость электроэнергиих[[5]](https://www.energosbyt.by/ru/info-potrebitelyam/fiz-l/schetchiki/ustanovka-i-ekspluatatsiya), потребляемой за год, (*Зэ.п*) определяется по формуле (6.25):

(6.25)

где

*Мсум* – суммарная мощность всей применяемой для разработки проекта техники (ПЭВМ и периферийной техники), кВт;

*Сэл* – стоимость одного кВт · ч электроэнергии, руб.; [[6](https://myfin.by/wiki/term/tarify-na-elektroenergiyu-dlya-naseleniya-v-belarusi)]

*А* – коэффициент интенсивного использования мощности, А = 0,98...0,9.

Расчет суммарной мощности всей применяемой техники проводится с использованием таблицы 6.16.

Таблица 6.16 – Расчет суммарной мощности ПЭВМ и периферийной техники

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Наименование оборудования | Количество | Мощность 1 ед, кВт | Суммарная мощность, кВт |
| 1 | Ноутбук Lenovo Ideapad 3 15ADA05 | 1 | 0.09 | 0.09 |
| 2 | Микроконтроллер ATmega328P | 3 | 0.01 | 0.03 |
| Итого | | 4 | - | 0.12 |

Затраты на материалы (*Зв.м*), необходимые для обеспечения нормальной работы ПЭВМ, составляют 1% от балансовой стоимости ЭВМ и определяются по формуле (6.26):

(6.26)

где

*Зпр* – затраты на приобретение (стоимость) ЭВМ, руб.;

*Кдоп* – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, *Кдоп* = 12–13 %;

*Км.з* – коэффициент, характеризующий затраты на вспомогательные материалы (*Км.з* = 0,01).

Затраты на текущий и профилактический ремонт (*Зт.р*) принимаются равными 5-9% от балансовой стоимости ЭВМ и рассчитывается по формуле (6.27):

(6.27)

где

*Кт.р* – коэффициент, характеризующий затраты на текущий и профилактический ремонт (*Кт.р* = 0,05–0,09).

Прочие затраты, связанные с эксплуатацией ЭВМ (*Зпр*), состоят из амортизированных отчислений на здания, стоимости услуг сторонних организаций и составляют 5% от балансовой стоимости и рассчитывается по формуле (6.28):

(6.28)

где

*Кпр* – коэффициент, характеризующий размет прочих затрат, связанных с эксплуатацией ЭВМ (К пр = 0,05).

Для расчета машинного времени ЭВМ (*tэвм*), необходимого для разработки и отладки проекта, следует использовать формулу (6.29):

(6.29)

где

*tрп* – срок реализации стадии «Рабочий проект» (РП), дн.;

*tвн* – срок реализации стадии «Ввод в действие» (ВП), дн.;

*Fсм* – продолжительность рабочей смены, ч; F см = 8 ч;

*Ксм* – количество рабочих смен, *Ксм* = 1.

### ***6.7.3 Расчет затрат на изготовление эталонного экземпляра***

Затраты на изготовление эталонного экземпляра (*Зэт*) определяются по формуле (6.30):

(6.30)

где

*Кэт* – коэффициент, учитывающий размер затрат на изготовление эталонного экземпляра, *Кэт* = 0,05.

При написании дипломной работы были использованы только бесплатные и свободно распространяемое ПО, поэтому затраты на технологию (*Зтех*) будут нулевыми.

### ***6.7.4 Расчет затрат на материалы***

Затраты на материалы (носители информации и прочее) (*Змат*), необходимые для обеспечения нормальной работы ПЭВМ, рассчитываются по формуле (6.31):

(6.31)

где

*Цi* – цена *i*-го наименования материала полуфабриката, комплектующего, руб.;

*Ni* – потребность в *i*-м материале, полуфабрикате, комплектующем, натур. ед.;

*Кт.з* – коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров, *Кт.з* = 0,1–1,3;

*Цоi* – цена возвратных отходов *i*-го наименования материала, руб.;

*Nоi* – количество возвратных отходов *i*-го наименования, натур. ед.;

*n* – количество наименований материалов, полуфабрикатов, комплектующих.

Расчет по данной на материалы проводится с помощью таблицы 6.17.

Таблица 6.17 – Расходы на материалы, покупные изделия и полуфабрикаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Наименование материала | Единица измерения | Количество | Цена с НДС за ед. изд., руб | Сумма с НДС, руб. |
| 1 | Флюс ФИМ ПЭТ 100мл[[6]](https://slesarka.by/catalog/ruchnoy-instrument/elektromontazhnyy-instrument/vse-dlya-payki/pripoy-kanifol) | шт. | 1 | 12.60 | 5.9 |
| Всего расходов (Змат) | | | | | 5.9 |

### ***6.7.5 Расчет общепроизводственных затрат***

Общепроизводственные затраты (*Зобщ.пр*) рассчитываются по формуле (6.32):

(6.32)

где

Ндоп – норматив общепроизводственных затрат.

### ***6.7.6 Затраты на непроизводственные затраты***

Непроизводственные затраты (*Знепр*) рассчитываются по формуле (6.33):

(6.33)

где

*Ннепр* – норматив непроизводственных затрат.

Таблица 6.18 – Результаты расчета суммарных затрат на разработку ПО, руб.

|  |  |  |
| --- | --- | --- |
| № п/п | Статья затрат | Итого |
| 1 | Затраты на оплату труда разработчиков (Зтр) | 2575.18 |
| 1.1 | Основная заработная плата разработчиков | 1747.08 |
| 1.2 | Дополнительная заработная плата разработчиков | 174.7 |
| 1.3 | Отчисления от основной и дополнительной заработной платы | 653.4 |
| 2 | Затраты машинного времени (Зм.в) | 924.96 |
| 3 | Затраты на изготовление эталонного экземпляра (Зэт) | 144.76 |
| 4 | Затраты на технологию (Зтех) | 0 |
| 5 | Затраты на материалы (Змат) | 6.49 |
| 6 | Общепроизводственные затраты (Зобщ.пр) | 87.35 |
| 7 | Непроизводственные затраты (Знепр) | 87.35 |
| 8 | Суммарные затраты на разработку ПО (Зр) | 3826.09 |

Заключительным этап работы над технико-экономической частью дипломного проекта является составление по результатам проведенных расчетов таблицы технико-экономических показателей проекта (таблица 6.21).

Таблица 6.21 – Технико-экономические показатели проекта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Наименование показателя | | Единица измерения | Базовый вариант | Проектируемый вариант |  |
| Показатели конкурентоспособности | | | | | |  |
| 1 | Интегральный коэффициент конкурентоспособности | | – | х | 2.45 |  |
| 1.1 | Коэффициент эквивалентности | | – | х | 1.07 |  |
| 1.2 | Коэффициент изменения функциональных возможностей | | – | х | 2.29 |  |
| 1.3 | Коэффициент соответствия нормативам | | – | х | 1 |  |
| 1.4 | Коэффициент потребления | | – | х | 1 |  |
| Показатели затрат на разработку | | | | | |  |
| 2 | Общая трудоемкость разработки ПО | | чел.-дн. | х | 211 |  |
| 3 | Затраты на разработку программы | | руб. | х | 3826.09 |  |
| 3.1 | Затраты на оплату труда разработчика | | руб. | х | 2575.18 |  |
| 3.2 | Затраты машинного времени | | руб. | х | 924.96 |  |
| 3.3 | Затраты на изготовление эталонного экземпляра | | руб. | х | 144.76 |  |
| 3.4 | Затраты на технологию | | руб. | х | 0 |  |
| 3.5 | Затраты на материалы | | руб. | х | 6.49 |  |
| 3.6 | Общепроизводственные затраты | | руб. | х | 87.35 |  |
| 3.7 | Непроизводственные затраты | | руб. | х | 87.35 |  |
|  | |

7 Раздел по энерго- и ресурсоснабжению

Энергосбережение означает переход к энергоэффективным технологиям во всех отраслях экономики, включая топливно-энергетический комплекс, и, прежде всего, энергоемкие отрасли, а также коммунально-бытовой сектор.

В Республике Беларусь в одной из первых из бывших республик Советского Союза была осознана необходимость энергосбережения и возведена в ранг государственной политики. Сегодня перед человечеством острейшим образом стоит три главные взаимосвязанные проблемы – питание, энергия и экология. В конечном итоге от их решения во многом зависит и состояние экономики, и степень воздействия на окружающую среду.

Как известно, Республика Беларусь относится к числу государств, которые недостаточно обеспечены собственными ТЭР, вынуждены импортировать 85 % потребляемых ТЭР. Так что потенциал энергосбережения, оцениваемый в 30-40 %, является важнейшим резервом, существенным источником энергии в топливно-энергетическом балансе экономики страны. Необходимо понять, что чем больше внимания уделяется вопросам энергоэффективности, тем выше жизненный уровень нации.

В этой связи реализация энергосберегающей политики должна стать общегосударственной программой, т.к. в условиях роста цен на энергоносители, в условиях сложной экологической обстановки, данная стратегия является приоритетной в сфере социально-экономического развития Республики Беларусь.

Республика Беларусь относится к странам, геологическая структура которых характеризуется крайне бедными топливно-энергетическими ресурсами (ТЭР). Беларусь, не располагая собственной сырьевой базой, импортирует примерно 85 % потребляемых ТЭР. Данное обстоятельство влияет на сферу экономической и экологической безопасности государства, национальная экономика становится зависимой от внешних поставщиков и уязвимой по отношению к резким колебаниям цен на энергоресурсы.

Снижение потребления энергоресурсов относительно динамики ВВП обусловлено тем, что субъекты хозяйствования в условиях рыночных отношений стали активно применять меры по экономному и рациональному использованию ТЭР, а также усилением государственного регулирования процессами энергосбережения в стране.

Интегральным показателем эффективности использования ТЭР в целом по государству является энергоемкость ВВП. Динамика изменения энергоемкости ВВП отражает степень внедрения достижений НТП в области энергосбережения, происходящие структурные сдвиги в отраслях народного хозяйства и интенсивность проведения организационно-технических мероприятий по экономии энергоресурсов.

Для получения положительных результатов, в будущем можно предложить следующий комплекс мер позволяющих реализовывать энергосберегающую политику: замещение импортируемых энергоресурсов местными видами топлива, позволяющее экономить денежные средства, валюту, которая идет на их закупку, и направлять их на реализацию новых инновационных проектов. использование вторичных энергоресурсов, позволяющих создавать новые рабочие места, улучшать экологическую обстановку. внедрение современных технологий. Данные мероприятия позволят выполнить директивы по снижению потребляемой электроэнергии на 10 % и более, но главное – позволит сэкономить деньги, топливо, эффективно использовать энергию, удешевить себестоимость производимой продукции на предприятиях республики и, как следствие, повысить ее конкурентоспособность на внутреннем и внешнем рынках.

Экономия энергии будет происходить благодаря установке импульсного блока питания.

[Блоки питания (БП)](https://promair.by/bloki-pitaniya-i-tverdotelnye-rele/) предназначены для реализации вторичной мощности в электрических цепях, а также для преобразования напряжения до необходимых значений. Элементы могут быть встроены в оборудование или подключаться самостоятельным звеном.

Существует два принципа преобразования электроэнергии в устройствах: на основе аналогового трансформатора и на импульсных блоках питания (ИБП).

Трансформаторные БП. Особенность блоков питания такого типа заключается в использовании силового трансформатора для изменения напряжения в сети. Устройства понижают амплитуду синусоидальной гармоники и направляют ее в выпрямитель, состоящий из силовых диодов. Сглаживание происходит за счет параллельно подключенной емкости. Окончательная стабилизация питающего напряжения осуществляется в полупроводниковой схеме с резисторами.

Трансформаторные преобразователи до недавнего времени были единственными в своем роде, но имели недостатки:

* большой вес и крупные габариты;
* высокую стоимость, зачастую многократно превосходящую цену остальных компонентов сети.

Импульсные БП. В конструкции устройства нет понижающего трансформатора. Почти во всей современной аппаратуре установлены именно импульсные блоки питания как наиболее компактные и эффективные.

Основные преимущества ИБП:

* Малый вес и компактные размеры. Уменьшение габаритов устройств обусловлено переходом от использования тяжелых силовых трансформаторов. В ИБП нет линейных управляющих систем, которые требуют установки больших охлаждающих радиаторов. Повышение частоты обрабатываемых сигналов также позволило уменьшить размеры конденсаторов.
* Высокий КПД. Низкочастотные трансформаторы характеризуются значительными потерями энергии в виде тепла, которое образуется в результате электромагнитных преобразований. В ИБП максимальные потери происходят в каскаде силовых ключей во время переходных процессов, а все остальное время транзисторы устойчивы. Потери энергии сведены к минимуму. КПД устройств достигает 98 %.
* Широкий диапазон входных напряжений. Область применения устройств значительно расширена. Импульсные технологии позволяют использовать блоки питания в сетях с различными стандартами электроэнергии.
* Встроенные системы защиты. Большинство моделей имеют автоматическую защиту от токов короткого кроткого замыкания, системы аварийного отключения нагрузок и т. д. Защитные устройства надежно встраиваются в конструкцию блоков благодаря применению миниатюрных цифровых полупроводниковых модулей.
* Доступная стоимость. Элементная база ИБП постоянно унифицируется. Снижается стоимость на основные компоненты устройств, которые выпускаются серийно на автоматических станках. Дополнительное сокращение затрат достигается за счет использования менее мощных полупроводников.

Недостатками ИБП являются:

* Ограничения по мощности. Существуют противопоказания, как при высоких, так и при низких нагрузках. Если в выходной цепи ток упадет ниже критического значения, то блок начинает генерировать напряжение с искаженными характеристиками, либо полностью отказывает схема запуска.
* Наличие высокочастотных помех. Блоки вырабатывают их в любом исполнении. Высокочастотные помехи транслируются в окружающую среду, поэтому необходимо дополнительно решать вопрос об их подавлении. В некоторых видах чувствительной цифровой аппаратуры использование ИБП по этой причине невозможно.

Устройство работает по принципу инвертора. Сначала переменное напряжение в блоке преобразуется в постоянное, а затем снова в переменное, но уже с необходимой частотой.

Схематически устройство можно представить как совокупность трех цепей:

* ШИМ-контроллера, который регулирует преобразование широтно-импульсной модуляции;
* каскада силовых ключей, подключенных по мостовой, полумостовой схеме или по схеме со средней точкой;
* импульсного трансформатора.

Взаимодействие элементов импульсного БП происходит по следующей схеме:

* напряжение 220В поступает на выпрямитель. Амплитуда сглаживается за счет работы конденсаторов емкостного фильтра;
* проходящие синусоиды выпрямляются диодным мостом;
* транзисторная схема преобразует ток в импульсы прямоугольной формы и высокой частоты.

Преобразование синусоид в импульсы может выполняться с гальваническим отделением питающей сети от выходных сетей или без нее.

8 Раздел по охране труда и технике безопасности

8.1 Общие сведения

Молниезащита (громозащита)— это комплекс технических решений и специальных приспособлений для обеспечения безопасности здания, а также имущества и людей, находящихся в нём. На земном шаре ежегодно происходит до 16 миллионов [гроз](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%BE%D0%B7%D0%B0), то есть около 44 тысяч за день. Опасность для зданий (сооружений) в результате прямого удара молнии может привести к:

• повреждению здания (сооружения) и его частей;

• отказу находящихся внутри электрических и электронных частей;

• гибели и [травмированию](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%B2%D0%BC%D0%B0) живых существ, находящихся непосредственно в здании (сооружении) или вблизи него.  
 Молния — особый вид прохождения электрического тока через огромные воздушные промежутки, источник которого — атмосферный заряд, накопленный грозовым облаком. Условия образования таких облаков большая влажность й быстрое изменение температуры. В результате возникновения восходящих потоков воздуха и быстрой конденсации водяных паров, содержащихся в воздухе, образуется большое количество водяной пыли, которая заряжается отрицательно.  
   Воздействие тока молнии возможно трех типов.  
   1) Прямой удар при разряде молнии в объект оказывает тепловое и механическое воздействие. При этом ток молнии может вызвать нагревание токоотвода до температуры каления, плавления и даже испарения. Быстрое разогревание вызывает нарастание электродинамических напряжений в конструкциях. Это вызывает механические разрушения, часто происходящие в виде взрыва.  
   2) Вторичное воздействие разряда молнии сопровождается появлением в пространстве изменяющетося во времени магнитного поля, которое индуцирует в контурах, образованных из различных протяженных металлических предметов (трубопроводов, электропроводок и т. д.), всегда имеющихся в здании, электродвижущую силу. В замкнутых контурах электродвижущая сила вызывает появление наведенных токов. В тех контурах, в которых контакты недостаточно надежны в местах соединения, эти токи могут вызвать искрение или сильное нагревание, что очень опасно для помещений, где могут образовываться опасные концентрации горючих или взрывоопасных веществ.

Внешняя молниезащита представляет собой систему, обеспечивающую перехват молнии и отвод её в землю, тем самым, защищая здание (сооружение) от повреждения и пожара. В момент прямого удара молнии в строительный объект правильно спроектированное и сооружённое молниезащитное устройство должно принять на себя ток молнии и отвести его по токоотводам в [систему заземления](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B7%D0%B5%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5), где энергия разряда должна безопасно рассеяться. Прохождение тока молнии должно произойти без ущерба для защищаемого объекта и быть безопасным для людей, находящихся как внутри, так и снаружи этого объекта.

Существуют следующие виды внешней молниезащиты:

• молниеприемная сеть;

• натянутый молниеприемный трос;

• молниеприемный стержень.

Помимо вышеупомянутых традиционных решений (приведенных как в международном стандарте, так и в российских нормативных документах с середины 2000-х годов получает распространение [молниезащита с системой ранней стримерной эмиссии](https://ru.wikipedia.org/wiki/%D0%90%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%BB%D0%BD%D0%B8%D0%B5%D0%B7%D0%B0%D1%89%D0%B8%D1%82%D0%B0), также именуемая активной молниезащитой. Однако нет никаких надёжных доказательств того, что активная молниезащита работает эффективнее, чем традиционная молниезащита тех же размеров.

В общем случае внешняя молниезащита состоит из следующих элементов:

[Молниеотвод](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BB%D0%BD%D0%B8%D0%B5%D0%BE%D1%82%D0%B2%D0%BE%D0%B4) (молниеприёмник, громоотвод) — устройство, перехватывающее разряд молнии. Выполняется из металла (нержавеющая либо оцинкованная сталь, алюминий, медь)

Токоотводы (спуски) — часть молниеотвода, предназначенная для отвода тока молнии от молниеприемника к заземлителю.

[Заземлитель](https://ru.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%B7%D0%B5%D0%BC%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C&action=edit&redlink=1) — проводящая часть или совокупность соединенных между собой проводящих частей, находящихся в электрическом контакте с землей непосредственно или через проводящую среду.

8.2 Общие сведение о точках ударов

   Занос высоких потенциалов в здания может происходить по любым металлоконструкциям, рельсовым путям, эстакадам, проводам ЛЭП, трубопроводам и т. д. Эти заносы сопровождаются электрическими разрядами, которые могут явиться источником взрыва или пожара.  
   Защита от поражения молнией зависит от типа производства, расположенного в здании, и от среднегодовой грозовой деятельности атмосферы. Грозовая деятельность может быть оценена ожидаемым количеством поражений молнией в год зданий и сооружений:

(8.1)

   где

l, b — длина и ширина защищаемого сооружения (или наименьшего описанного прямоугольника для зданий сложной конфигурации), м; h — наибольшая высота сооружения, м; n — среднегодовое число ударов молнии в 1 км2 поверхности земли (в данном географизическом месте).  
   Все сооружения по необходимости устройства молниезащиты разделены на три категории.  
   В зданиях и сооружениях I категории длительное время сохраняются или систематически возникают взрывоопасные смеси газов, паров и пыли с воздухом или другими окислителями; перерабатываются или хранятся взрывчатые вещества в неметаллических упаковках или в открытом виде. Взрыв таких зданий и сооружений сопровождается значительными разрушениями и человеческими жертвами.  
   В зданиях и сооружениях II категории взрывоопасные смеси газов, паров и пыли с воздухом или другими окислителями возникают только в момент производственных аварий или неисправностей; взрывчатые вещества хранятся в прочной металлической упаковке. Взрыв в таких помещениях сопровождается, как правило, незначительными разрушениями без человеческих жертв.  
   В зданиях и сооружениях III категории прямой удар молнии может вызвать пожар, механические разрушения и поражения людей. К. этой категории можно отнести жилые и общественные здания, дымовые трубы, водонапорные башни, газгольдеры, резервуары.

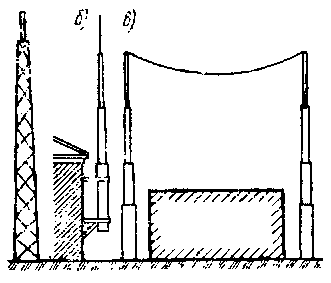


Рисунок 8.1 ---- Молниеотводы:  
а — стержневой отдельно стоящий; 6 — то же, укрепленный иа здании; в — тросовый

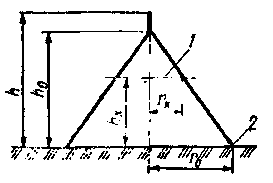


Рисунок 8.2 --- Зона защиты одиночного стержневого молниеотвода:  
1 — граница зоны защиты иа уровне защищаемого объекта; 2 — граница зоны защиты на уровне земли

   В соответствии с инструкцией СН 305—77 здания и сооружения I и II категорий подлежат молниезащите от прямых ударов молнии, вторичных воздействий и заноса высоких потенциалов.  
   Здания и сооружения III категории должны иметь защиту от прямых ударов молнии и от заноса высоких потенциалов по надземным проводящим коммуникациям (за исключением наружных емкостей со взрыво- и пожароопасными жидкостями и газами, а также вертикальных наружных труб).

8.3 Общие сведения о защите от ударов молний

   Для защиты зданий и промышленных сооружений от тока молнии устраивают молниеотводы (громоотводы). Они воспринимают молнию и отводят ее ток в землю. Молниеотводы делят на стержневые и тросовые, которые подразделяют на отдельно стоящие, изолированные и не изолированные от защищаемого здания.  
   Наиболее часто применяются стержневые молниеотводы. Тросовые используются для защиты длинных и узких сооружений, а также, когда из-за густой сети подземных коммуникаций нельзя установить большое число стержневых молниеотводов.  
   Защитное действие молниеотвода основано на свойстве молнии поражать наиболее высокие и хорошо заземленные металлические конструкции и характеризуется зоной защиты, под которой понимается часть пространства, защищенного от прямых ударов молнии с определенной степенью надежности.

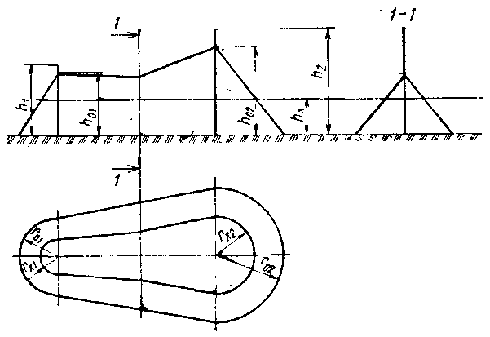


Рисунок 8.3 --- Зона защиты двойного стержневого молниеотвода

Зона защиты двойного стержневого молниеотвода, состоящего из двух стержневых молниеотводов разной высоты. Торцовые части сечения — это зоны защиты одиночного стержневого молниеотвода.  
   Для защиты больших площадей и объектов применяют многократные стержневые молниеотводы. Для определения внешней границы зоны защиты трех-четырех взаимодействующих молниеотводов используют те же приемы, что и для одиночного или двойного стержневого молниеотвода.  
   Конструктивно молниеотвод представляет собой молниеприемник, токоотводящий спуск и заземлитель. Опоры молниеотводов могут выполняться из стали в виде стоек из труб одного диаметра и железобетонных колонн или дерева. Там, где это возможно, в качестве опор для крепления токоведущих частей молниеотвода следует использовать конструкции самих защищаемых зданий. Молниеприемники стержневых молниеотводов изготавливаются из стальных стержней и имеют высоту не менее 200 мм.  
   Высокие объекты, как правило, имеют каркас из металла или железобетона, который может служить токоотводом. Следует только предусмотреть надежное соединение во время строительства стальной арматуры железобетонных деталей каркаса. В качестве таких токоотводов можно использовать конструктивные элементы (перила балконов, пожарные лестницы и т, д.) или специально проложенные стальные проводки. К каркасу объекта, являющемуся токоотводом, подсоединяют все металлические элементы здания (трубопроводы, каркасы лифтов и т. д.). Каркас объекта через каждые 20... 30 м по его периметру присоединяют к заземляющему контуру.  
   Защита от заноса высоких потенциалов осуществляется следующим образом. Для сооружений I категории ввод воздушных линий любого назначения запрещается. Вместо них применяют подземные кабели. Ввод в объект трубопроводов разрешается только от цехов, представляющих общую технологическую линию. В месте ввода трубопроводы соединяют с заземлителей. Для II категории линии любого назначения, подключаемые к объекту, должны иметь кабельный ввод протяженностью не менее 50 м. Для сооружений III категории разрешается ввод воздушных линий.  
   Тип заземлителя для молниеотвода выбирают исходя из удельного сопротивления грунта и импульсного сопротивления Ra: Ru=aR (где а — импульсный коэффициент; R — сопротивление растеканию тока промышленной частоты, Ом).  
   Пострадавшим от электрического тока должна быть немедленно оказана первая помощь. Если пострадавший окажется в соприкосновении с токоведущими частями, необходимо немедленно освободить его от действия тока.  
   При поражении электрическим током даже при отсутствии у потерпевшего дыхания, сердцебиения пульса не следует отказываться от первой помощи — необходимо срочно вызвать врача и сделать искусственное дыхание. Бывали случаи, когда мнимо умершие после поражения электрическим током в результате принятых быстрых мер были возвращены к жизни через несколько часов.  
   Каждый цех и каждая смена должны иметь шкафчик первой помощи с необходимыми средствами для временной остановки кровотечения, перевязки ран и проведения мероприятий по оживлению.  
   В установках напряжением до 1000 В для отделения пострадавшего от токоведущих частей можно воспользоваться сухими подручными предметами, не проводящими электрический ток: одеждой, канатом, доской и др. Можно также взяться за полы сухой одежды пострадавшего, избегая касания к металлическим окружающим предметам и частям тела пострадавшего, не покрытым одеждой. Для изоляции рук можно надеть резиновые перчатки, суконную фуражку или накинуть на пострадавшего прорезиненную материю или просто сухую одежду. Действовать следует быстро и решительно, но осторожно.  
   Меры первой помощи зависят от того, в каком состоянии находится пострадавший после освобождения от тока: если он не потерял сознание, необходимо обеспечить полный покой до прибытия врача. Если же быстро вызвать врача невозможно, то пострадавшего нужно срочно доставить в лечебное учреждение. Если пострадавший потерял сознание, но дыхание его сохранилось, его нужно удобно уложить, обеспечить покой, создать приток свежего воздуха, дать понюхать нашатырный спирт, растереть и согреть тело. Вместе с тем надо срочно вызвать врача.

ЗАКЛЮЧЕНИЕ

Таким образом создадим продукт, способный практически напрямую и на всех современных устройствах подключаться и работать с машинами Festo’s Mechatronics. Он куда более гибкий и функциональный чем предыдущее решение, а также является кроссплатформенным, что весьма важный фактор в условиях сегодняшнего развития операционных систем.

Другой возможной реализацией данного программного продукта является использование его в качестве модуля или библиотеки C++. При желании пользователя данную библиотеку можно будет использовать и на других популярных языках программирования, таких как Jave, C#, Python, Go и т.д.

Так же подобный модуль является легко масштабрируемым, так как подходит под все признаки объектно-ориентированного программирования. Пользователю необходимо будет только отнаследоваться от наших базовых классов и дополнять их по своему желанию.

Наши логические блоки, такие как приложение на ПК и микроконтроллер связаны между собой протоколом общение, который определён в коде диплома. Это теоретически позволит каждому из них работать отдельно друг от друга с другим абстрактным блоком, если соблюдается протокол общение.

Список использованных источников

1. Компания Festo [Электронный ресурс], Festo Didactic Company. ‒ Режим доступа: https://www.festo.com/us/en/ ‒ Дата доступа: май 2021г.
2. Электронная библиотека Wikipedia [Электронный ресурс], Wikipedia Company. ‒ Режим доступа: https://ru.wikipedia.org/wiki/ ‒ Дата доступа: май 2021г.
3. Документация Qt Framework [Электронный ресурс], QT Company. ‒ Режим доступа: https://doc.qt.io/‒ Дата доступа: май 2021г.
4. Минимальная заработная плата [Электронный ресурс]. Режим доступа: – http://mintrud.gov.by/ru/min\_zar\_plata\_2019/. - Дата доступа 10.05.2021.
5. Тарифы на электрическую энергию для юридических лиц и индивидуальных предпринимателей [Электронный ресурс]. Режим доступа: – https://www.energosbyt.by/ru/info-potrebitelyam/fiz-l/schetchiki/ustanovka-i-ekspluatatsiya - Дата доступа 10.05.2021.
6. Припой, канифоль, флюс [Электронный ресурс]. Режим доступа: – [https://slesarka.by/catalog/ruchnoy-instrument/elektromontazhnyy-instrument/vse-dlya-payki/pripoy-kanifol. - Дата доступа 12.05.2021](https://slesarka.by/catalog/ruchnoy-instrument/elektromontazhnyy-instrument/vse-dlya-payki/pripoy-kanifol.%20-%20Дата%20доступа%2012.05.2021).
7. Документация microship [Электронный ресурс], MicroShip Company. ‒ Режим доступа: ww1.microship.com Дата доступа: май 2021г.
8. Статьи по Arduino [Электронный ресурс], Режим доступа: www.amperka.ru Дата доступа: май 2021г.
9. Документация Компас3Д [Электронный ресурс], АСКОН Company. ‒ Режим доступа: www.kompas.ru Дата доступа: май 2021г.
10. Документация chidip [Электронный ресурс], ChipDip Company. ‒ Режим доступа: www.chipdip.ru Дата доступа: май 2021г.
11. Форум stackoverflow [Электронный ресурс], StackOverflow. ‒ Режим доступа: www.stackoverflow.com Дата доступа: апрель 2021г.
12. Документация С++ [Электронный ресурс], CppReference. ‒ Режим доступа: www.cppreference.com Дата доступа: апрель 2021г.
13. Форум разработчиков Habr [Электронный ресурс], Habr Company. ‒ Режим доступа: www.habr.com Дата доступа: апрель 2021г.
14. Облачное хранилище github [Электронный ресурс], Git Company. ‒ Режим доступа: www.github.com Дата доступа: апрель 2021г.
15. Учебник по коду Codewars [Электронный ресурс], CodeWars Company. ‒ Режим доступа: www.codewars.com Дата доступа: апрель 2021г.
16. Статьи AtmelStudio [Электронный ресурс], Atmel Company. ‒ Режим доступа: www.atmel.com Дата доступа: апрель 2021г.
17. Форум DevBy [Электронный ресурс], DevBY Company. ‒ Режим доступа: www.dev.by Дата доступа: апрель 2021г.
18. Статьи CyberForum [Электронный ресурс], Cyberforum. ‒ Режим доступа: www.cyberforum.ru Дата доступа: апрель 2021г.
19. Библиотека технической документации [Электронный ресурс], Data Sheet. ‒ Режим доступа: www.alldatasheet.comДата доступа: апрель 2021г.
20. Курс SQL [Электронный ресурс], SqlAlchemy. ‒ Режим доступа: www.sqlalchemy.com Дата доступа: апрель 2021г.