МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Разработка программного модуля для управления электроприводом тягового двигателя на базе отечественного микроконтроллера

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.406 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Д.В. Топольский
«___»_____2025 г.

Автор работы,
студент группы КЭ-406
_____В.Д. Савченко
«___»_____2025 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____С.В. Сяськов
«___»_____2025 г.

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Савченко Виктор Денисович
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка программного модуля для управления электроприводом тягового двигателя на базе отечественного микроконтроллера» утверждена приказом по университету от «21» апреля 2025 г. №648-13/12.

2. **Срок сдачи студентом законченной работы:** 1 июня 2025 г.

3. **Исходные данные к работе:**

3.1. Отладочная плата микроконтроллера MIK32:

- архитектура RISC-V;
- рабочая частота до 32 МГц;
- ОЗУ 16 КБ;
- ПЗУ (EEPROM) 8 КБ;
- АЦП 12 бит, 8 каналов, частота преобразования 0,8 МГц;
- ЦАП 12 бит, 2 канала, частота преобразования 1 МГц;

- интерфейсы SPI, I2C, UART.

3.2. Двигатель вентильный моментный ДВМ100.22:

- номинальное напряжение 24 В;
- номинальный ток 5 А;
- номинальный момент 2,5 Нм;
- номинальная частота вращения 350 об/мин.

4. **Перечень подлежащих разработке вопросов:**

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.
2. Разработка алгоритма управления электроприводом тягового двигателя.
3. Написание программного кода для управления работой электропривода
4. Проведение тестирования программного модуля.

5. **Дата выдачи задания:** 2 декабря 2024 г.

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*В.Д. Савченко* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	03.03.2025	
Разработка алгоритма управления электроприводом тягового двигателя	22.03.2025	
Написание программного кода для управления работой электропривода	12.04.2025	
Проведение тестирования программного модуля	26.04.2023	
Компоновка текста работы и сдача на нормоконтроль	22.05.2025	
Подготовка презентации и доклада	30.05.2023	

Руководитель работы _____/*Д.В. Топольский*/

Студент _____/*В.Д. Савченко*/

Аннотация

В.Д. Савченко. Разработка программного модуля для управления электроприводом тягового двигателя на базе отечественного микроконтроллера. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 62 с., 9 ил., библиогр. список – 21 наим.

В выпускной квалификационной работе представлена разработка программного модуля для управления электроприводом тягового двигателя на базе отечественного микроконтроллера MIK32 «Амур». Основной целью работы являлось создание эффективного и надежного решения, соответствующего требованиям импортозамещения и технологической независимости. В ходе исследования проведен анализ современных методов управления электроприводами, выполнена разработка алгоритма управления на основе широтно-импульсной модуляции (ШИМ) с электронной коммутацией фаз, а также реализован программный код для микроконтроллера. Проведенные тесты подтвердили корректность работы системы, включая управление ШИМ, обработку сигналов с датчиков Холла и защиту от перегрузки по току. Результаты работы нацелены на импортозамещение систем управления электроприводами на основе отечественных микроконтроллеров и могут быть применены на транспорте, промышленности и других областях.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные системы управления электроприводами находят широкое применение в транспортной отрасли, промышленной автоматизации и энергетике. Тяговые двигатели, являясь ключевым элементом электротранспорта, требуют надёжного и эффективного управления, что возможно благодаря использованию программируемых микроконтроллеров. В условиях текущих вызовов, таких как импортозамещение и стремление к технологической независимости, особое значение приобретает разработка программных решений для отечественных микроконтроллеров.

Использование отечественных микроконтроллеров позволяет не только минимизировать зависимость от зарубежных технологий, но и сократить затраты на производство, адаптируя решения под специфику российских условий. Однако создание программного обеспечения для управления тяговыми электроприводами требует глубокого понимания архитектуры микроконтроллеров, особенностей их периферии и возможностей обработки сигналов в реальном времени.

1. АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ, МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ, ЗАТРАГИВАЮЩЕЙ ИССЛЕДУЕМУЮ НАУЧНО-ТЕХНИЧЕСКУЮ ПРОБЛЕМУ

1.1. ОБЩАЯ ИНФОРМАЦИЯ ОБ ЭЛЕКТРОПРИВОДЕ И УПРАВЛЕНИИ ЭЛЕКТРОПРИВОДАМИ

Электропривод определяется как электромеханическая система, которая состоит из электродвигателя, преобразовательных и управляющих устройств. Основная задача электропривода заключается в преобразовании электрической энергии в механическую и передаче этой энергии исполнительным органам рабочих машин. Примерами таких машин являются транспортёры, токарные станки, насосы, лифты и другие устройства, выполняющие производственные и технологические операции [1].

Электропривод выполняет две важные функции – силовую и информационную. Силовой канал отвечает за преобразование и передачу энергии, а информационный – за управление этим процессом, т. е. он выполняет сбор и обработку данных о состоянии системы. Эти два канала тесно переплетены между собой и вместе формируют основу функциональной схемы электропривода (рисунок 1).

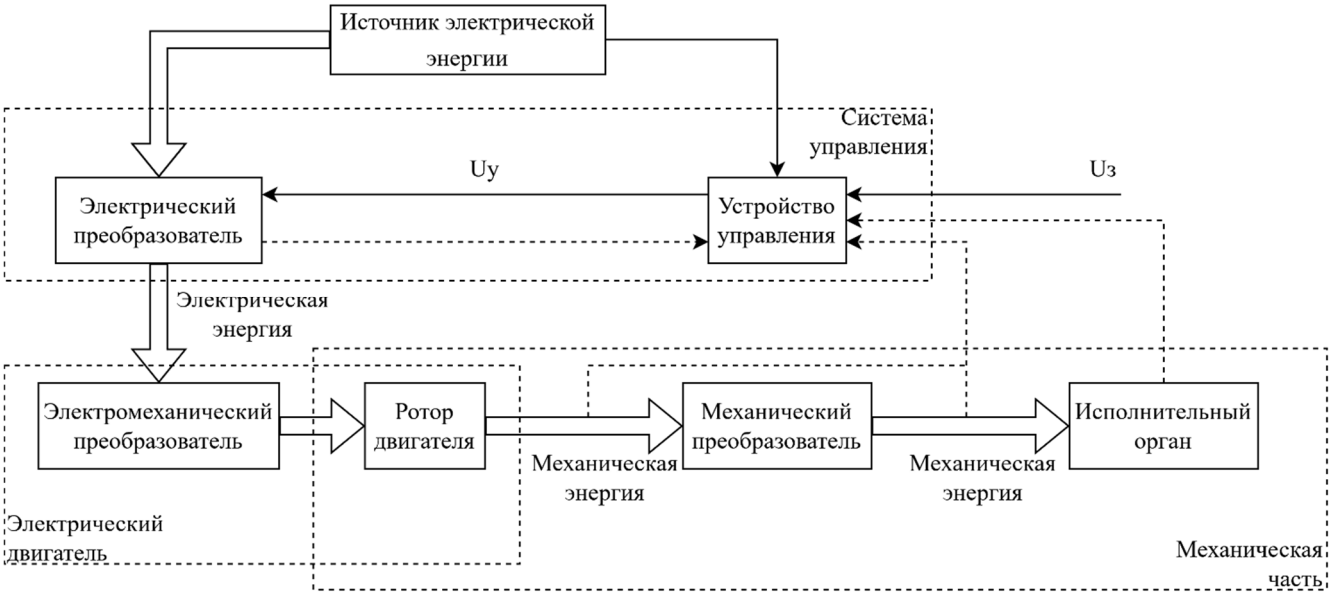

Рисунок 1 – Функциональная схема электропривода [1]

Электропривод включает электрическую и механическую части силового канала.

1. Электрическая часть: обеспечивает передачу энергии от источника и, при необходимости, её преобразование для соблюдения требованиям системы.
2. Механическая часть: включает ротор двигателя, механические передачи, редуктор или вариатор и исполнительный орган рабочей машины. Эти элементы обеспечивают передачу и преобразование энергии в механическое движение.

Электропривод является частью более крупных систем, таких как система электроснабжения, технологическая установка или информационная система. Его эффективность в таких системах определяется оптимальным взаимодействием всех компонентов.

Электропривод стоит рассматривать как компонент интегрированной системы, поскольку системы элементы связаны между собой. Например, использование преобразователей энергии на основе полупроводниковых приборов расширяет функциональные возможности, однако приводит к проблемам электромагнитной совместимости. Для поддержания стабильности

функционирования систем электроснабжения требуется использовать такие вспомогательные устройства, как фильтры и компенсаторы.

Конструктивная интеграция, например мотор-колесо в транспортных средствах или электрошпиндель в станках, улучшает функциональные характеристики привода и позволяет добиться новых преимуществ. В свою очередь, использование микропроцессорных технологий в информационном канале повышает качество управления, обеспечивая единство энергетических и информационных процессов в системе. [1].

1.2. ПРИНЦИПЫ УПРАВЛЕНИЯ В ЭЛЕКТРОПРИВОДЕ

Управление в электроприводе представляет собой процесс преобразования электрической энергии в механическую таким образом, чтобы обеспечить выполнение заданного алгоритма работы технологической установки. Управление может быть одномерным (по одной регулируемой координате) или многомерным (по нескольким координатам).

Объектами управления выступают элементы силового канала электропривода: электрические, электромеханические и механические преобразователи, а также рабочий орган. Входные воздействия разделяются на две группы:

1. Управляющие воздействия – определяются системой управления.
2. Возмущающие воздействия – зависят от внешних факторов, таких как момент нагрузки или напряжение питающей сети.

Процесс управления направлен на формирование управляющих воздействий, позволяющих реализовать заданный алгоритм работы системы. Совокупность элементов, участвующих в этом процессе, называется системой управления электропривода.

В зависимости от количества каналов передачи информации между устройством управления и объектом управления все системы управления делятся на два больших класса:

1. Разомкнутые системы:

- алгоритм управления реализуется управляющий устройством без наличия какой-либо информации о текущем состоянии объекта управления;
- применяются для регулирования параметрами рабочего органа при невысоких требованиях к точности и диапазону регулирования параметров.

2. Замкнутые системы:

- алгоритм управления реализуется с использованием каналов обратной связи, которые позволяют корректировать управляющее воздействие на объект управления;
- применяются в случаях, когда необходима высокая точность и широкой диапазон регулирования параметров.

В последние годы в связи с бурным развитием микропроцессорной техники получили развитие адаптивные системы управления, среди которых есть и самонастраивающиеся, то есть такими, в которых автоматически изменяются и структура (алгоритм), и параметры регуляторов. Реализация алгоритмов таких систем стала возможна благодаря использованию в их составе микроконтроллеров, играющих главную роль в управлении [1].

1.3. ОБЩАЯ ИНФОРМАЦИЯ О МИКРОКОНТРОЛЛЕРАХ

Микроконтроллер – это микроэлектронное программируемое логическое устройство, предназначенное для обработки информации и управления процесса обмена этой информацией в составе микропроцессорных систем (компьютера) Так, микроконтроллеры широко используются, например, в бытовой технике, автомобильной электронике, аэрокосмической и военной отраслях, а также в промышленном производстве [2].

Базовая структура микроконтроллера включает в себя несколько различных компонентов:

- центральный вычислительное устройство (процессор) – отвечает за выполнение вычислений и логических операций;
- блок постоянной памяти для хранения программ и начальных настроек микроконтроллера. Его главная задача заключается в том, чтобы сделать устройство независимым от внешнего питания;
- порты ввода/вывода, которые используются для организации связи с внешними устройствами;
- аналого-цифровые и цифро-аналоговые преобразователи (АЦП/ЦАП) применяются для преобразования аналоговых сигналов в цифровые и наоборот;
- таймеры, которые используются для задач, связанных с измерением времени и подсчетом внешних импульсов.

Структура микроконтроллера изображена на рисунке 2, на которой отражены основные компоненты.

Рисунок 2 – Структура микроконтроллера

Микроконтроллеры обычно классифицируют по разрядности, типу используемой памяти, архитектуре памяти и набору команд. Рассмотрим детальнее каждую из этих классификаций.

1. По разрядности:

Традиционно разрядность современной вычислительной техники кратна восьми. Другие варианты встречаются крайне редко, поскольку нестандартная

разрядность приводит к сложностям использования стандартного программного обеспечения. Наиболее распространены следующие типы:

- 8-битные – самые простые и наименее мощные. Хорошо подходят для выполнения базовых задач, таких как простое управление устройствами или считывание данных с датчиков. Они дешевые, потребляют минимальное количество энергии;
- 16-битные – предлагают уже больше вычислительных ресурсов и больший объем памяти, чем восьмибитные, что делает их подходящими для более сложных задач, например, для управления моторами, для проектов, которые требуют обработку большого объема данных;
- 32-битные – еще более мощные устройства с большим объемом памяти и высокой скоростью обработки данных. Подходят для сложных приложений, например, смартфоны, мультимедийные устройства и любых других систем, в которых требуется высокая производительность.

2. По типу используемой памяти:

- с встроенной (внутренней) памятью – все функциональные блоки находятся на одном кристалле;
- с внешней памятью – для хранения данных требуется подключение внешних компонентов.

3. По архитектуре памяти:

- гарвардская архитектура – данные и программа хранятся в разных адресах пространства памяти;
- архитектура фон-Неймона – данные и программа имеют общий адрес памяти.

4. По набору команд:

- complex instruction set computer (CISC) – характеризуется большим числом различных по формату и длине команд, а также сложной кодировкой инструкции;

- reduced instruction set computer (RISC) – наоборот, включает простые инструкции одинакового размера.

Использование микроконтроллеров в составе различных устройств позволяет значительно снизить их размер, стоимость, одновременно повышая производительность и функциональные возможности. Микроконтроллеры, благодаря своей универсальности, размерам и энергоэффективности становятся незаменимыми в условиях стремительного развития цифровых технологий.

1.4. ТЕКУЩЕЕ СОСТОЯНИЕ И ПЕРСПЕКТИВЫ ПРИМЕНЕНИЯ ОТЕЧЕСТВЕННЫХ МИКРОКОНТРОЛЛЕРОВ

В статье Заботнова П. В. [3] приводится детальный анализ состояния микроэлектронной отрасли в России, включая сравнение ее положения относительно мирового рынка. Акцентируется внимание на таких ключевых проблемах, которые препятствуют развитию отрасли, как устаревшие технологии, дефицит квалифицированных кадров и влияние санкционного давления.

Значительная часть российских предприятий ориентирована на нужды военного комплекса и аэрокосмической отрасли. Это приводит к узкой специализации и мелкосерийному производству, часто на уже устаревшем оборудовании, использующих проектные нормы 130, 90 и 65 нм. Эти ограничения затрудняют создание конкурентоспособных изделий, включая микроконтроллеры, энергонезависимую память, приемо-передающие устройства и аналого-цифровые преобразователи. Также подчеркивается, что российская микроэлектроника не обладает рядом ключевых этапов цепочки производства, таких как собственное производство технологического оборудования, разработка IP-блоков и полноценное тестирование продукции.

Серьезной проблемой является и дефицит кадров, недостаток таких высококвалифицированных специалистов, как инженеры-технологи и разработчики программного обеспечения. По мнению автора, решение этой

проблемы должно стать одним из приоритетов отрасли. Немаловажно развитие международного сотрудничества с дружественными странами, такими как Китай, для восполнения недостающих компетенций и доступа к передовым технологиям.

Основным выводом работы является необходимость комплексного подхода к развитию микроэлектронной отрасли. Автор рекомендует сосредоточить усилия на модернизации производственных мощностей, развитии науки, подготовке кадров и локализации производства микроэлектронных компонентов. На этом фоне деятельность ключевых российских компаний демонстрирует, как в условиях существующих ограничений отрасль пытается находить решения и развивать современные технологии.

Важной частью реализации этих задач выступают такие компании, как АО «Микрон» [4] и АО «Ангстрем» [5], играющие ключевую роль в обеспечении внутреннего рынка полупроводниковой продукции. Их деятельность охватывает разработку и производство интегральных схем, микросхем управления питанием, устройств интернета вещей (IoT) и других электронных компонентов, применяемых в транспорте, энергетике и цифровой экономике [6].

АО «Микрон», основанное в 1964 году, является крупнейшим отечественным производителем микроэлектроники. Компания изготавливает микросхемы с проектными нормами от 250 до 65 нм. Среди ключевых направлений её деятельности:

- выпуск платёжных карт, сертифицированных для работы в системе «Мир»;
- производство транспортных карт и пропусков;
- разработка IoT-устройств, включая трекеры для логистики и датчики для контроля температуры и состояния грузов;

- создание микроконтроллеров на архитектуре RISC-V для IoT-приложений. Планируется занять до 30% российского рынка контроллеров к 2025 году.

АО «Ангстрем», основанное в 1963 году, специализируется на производстве микросхем с проектными нормами от 1 мкм до 0,6 мкм. Компания активно работает в следующих областях:

- разработка силовых модулей для транспортных систем, таких как трамваи и электромобили;
- производство компонентов для систем освещения;
- создание радиочастотных идентификаторов (RFID) для транспортной и складской логистики.

Обе компании ведут разработки для высокотехнологичных сфер, включая телекоммуникации, космос и сельское хозяйство. АО «Ангстрем» выпускает микросхемы, которые способны выдержать радиационные условия пространства космоса, а «Микрон» разрабатывает умные устройства, помогающие городским службам – например, автоматизировать сбор и переработку отходов.

Несмотря на такие существующие проблемы, как технологическое отставание, санкции и нехватку квалифицированных специалистов, деятельность компаний «Микрон» и «Ангстрем» демонстрирует способность российских производителей развивать современные решения и занимать важные позиции на рынке. Их работа направлена на снижение импортозависимости и укрепить технологическую независимость страны. Это подтверждается запуском новых проектов и постоянной модернизацией производств.

Еще одной компанией, внесшей значительный вклад в развитие отечественной микроэлектроники, является АО «ПКК Миландр» [7]. Основанная в 1993 году в Зеленограде, компания с момента своего создания

активно занимается организацией поставок электронных компонентов российским предприятиям.

В 2003 году «Миландр» расширил свою деятельность, открыв Центр проектирования, специализирующийся на разработке микроэлектронной элементной базы. Среди разработок центра – микроконтроллеры, микропроцессоры, схемы памяти и другие компоненты, которые могут поставляться как с приемкой отдела технического контроля, так и с приемкой заказчика, соответствуя высоким требованиям надежности.

Для обеспечения качества продукции в 2004 году был создан Испытательный технический центр микроприборов (ООО «ИТЦ МП»), а в 2005 году запущено сборочно-измерительное производство. Это позволило компании выстроить полный цикл разработки и тестирования интегральных схем.

География компании охватывает ключевые регионы России: филиалы «Миландра» работают в Нижнем Новгороде, Воронеже и Екатеринбурге. Сегодня продукция компании используется более чем 700 предприятиями в России и странах СНГ, включая разработки для авиационной и специальной техники.

Основные направления деятельности компании включают:

- разработку интегральных микросхем с проектными нормами от 2 мкм до 0,04 мкм;
- организацию испытаний и измерений, включая тестирование импортных микросхем;
- комплексные поставки электронных компонентов для радиоэлектронной аппаратуры гражданского и специального назначения.

Инфраструктура компании включает 4000 м² офисных и производственных площадей, более 250 высококвалифицированных специалистов и современное аппаратно-программное оборудование для проектирования и тестирования.

Следующим примером успешного отечественного предприятия в области микроэлектроники является АО «Научно-исследовательский институт электронной техники» (НИИЭТ) [8]. Эта компания занимает ключевые позиции в разработке сложной микроэлектронной продукции, включая микроконтроллеры и силовые транзисторы, активно способствуя достижению технологической независимости России.

История НИИЭТ началась в 1959 году с основания Особого конструкторского бюро (ОКБ), которое в 1961 году было преобразовано в Центральное конструкторское бюро (ЦКБ) при Воронежском заводе полупроводниковых приборов (ВЗПП).

Важной частью дальнейшего развития микроэлектронной отрасли является стратегическое направление, закрепленное в "Распоряжении Правительства РФ" [9]. Это документ, представляющий собой стратегический план развития электронной промышленности России до 2030 года. В нем отражены вызовы, с которыми сегодня сталкивается отечественная микроэлектроника, включая важную задачу импортозамещения и создания на рынке конкурентоспособных продуктов.

Особое внимание уделено внедрению современных бизнес-моделей и технологий, которые помогут стимулировать производство российской электронной продукции. Правительство ставит цель на значительное расширение использования производственных, научных и инженерных ресурсов через сотрудничество как внутри страны, так и с зарубежными партнерами. В частности, с такими как Китай, что поможет открыть доступ к новым технологиям и поспособствует глубокому включению России в глобальные цепочки создания стоимости.

Подчеркивается и важность создания и поддержки отечественных микроконтроллеров – особенно с учетом растущей зависимости технологий от электронных компонентов. Создание собственных решений является главным шагом в повышении технологической независимости страны. Отдельно

говорится о необходимости подготовки кадров. Планируется внедрение механизмов для прогноза и отслеживания потребностей в специалистах.

В документ содержатся конкретные шаги, которые направлены на внедрение новых технологий, включая создание линейки оборудования, обеспечение безопасности и устойчивости процессов производства. Данные меры не только повысят эффективность текущих разработок, но и зададут фундамент для появления новых направлений для разработки инноваций.

Данный стратегический план служит фундаментом для формирования программ и инициатив в микроэлектронной области, акцентируя внимание на необходимости интеграции отечественных ресурсов, технологий и международного сотрудничества. Такой подход создает устойчивую базу для дальнейшего роста отрасли и укрепления ее роли в экономике страны.

1.5. ТРЕБОВАНИЯ К МИКРОКОНТРОЛЛЕРУ ДЛЯ УПРАВЛЕНИЯ ЭЛЕКТРОПРИВОДАМИ

Для качественного управления электроприводами микроконтроллеры должны обладать определенным набором характеристик, которые позволят обеспечить выполнение сложных алгоритмов и надежность работы системы. Ключевые требования можно разделить на следующие категории:

1. Высокопроизводительный процессор: должен обеспечивать выполнение сложных расчетов и обработку сигналов в режиме реального времени с производительностью составляющей не менее 100 миллионов операций в секунду [10].
2. Периферийные устройства:

- встроенные таймеры-счетчики с генерацией ШИМ с частотой до 50 кГц и высоким разрешением для управления силовыми ключами [11];
- аналого-цифровой преобразователь с 12-битной разрядностью и выше и временем преобразования до 100 нс для точного измерения токов и напряжений [10];

- поддержка квадратурных декодеров (энкодеров) для обработки сигналов с датчиков положения и реализации обратной связи [11].

3. Надежность и устойчивость: устойчивость к электромагнитным помехам, механизмы защиты от перегрузок и аварийных ситуаций, включая отключение ШИМ при критических параметрах [10].
4. Интерфейсы для связи: поддержка интерфейсов, таких как controller area network (CAN), serial peripheral interface (SPI), inter-integrated circuit (I2C) и universal asynchronous receiver/transmitter (UART) для интегрирования в системы автоматизации распределенного управления [11].
5. Энергоэффективность: оптимизация энергопотребления для мобильных и автономных систем, а также компактные размеры микроконтроллеров позволяют интегрировать их в устройства с ограниченным пространством (робототехнике и IoT) [10].
6. Гибкость и программируемость: поддержка обновления прошивки и универсальность в реализации различных алгоритмов управления, включая векторное управление, фаззи-логика и прямое управление моментом [11].

1.6. ПРИМЕРЫ ОТЛАДОЧНЫХ ПЛАТ МИКРОКОНТРОЛЛЕРОВ ДЛЯ УПРАВЛЕНИЯ ЭЛЕКТРОПРИВОДАМИ

После анализа общих требований к микроконтроллерам для управления электроприводами важно рассмотреть конкретные примеры устройств, которые соответствуют этим требованиям и используются в различных сферах, от промышленной автоматизации до бытовых систем.

Начинать работу с микроконтроллерами значительно удобнее, когда они входят в состав готового отладочного комплекта. Такие комплекты позволяют быстрее приступить к проектированию, отладке и тестированию системы. Многие производители предлагают отладочные платы, которые включают все необходимые интерфейсы и элементы для разработки.

Из четырёх ранее рассмотренных отечественных производителей микроконтроллеров – АО «Микрон», АО «Ангстрем», «Миландр» и АО «НИИЭТ», готовые отладочные решения были найдены у трёх: «Микрон», «Миландр» и «НИИЭТ». Далее будут рассмотрены примеры таких отладочных плат.

1. Отладочная плата СТАРТ-MIK32 [12]:

Отладочная плата «Старт» – программируемое устройство на основе RISC-V-микроконтроллера MIK32 «Амур» (К1948ВК018) [13]. Плата оснащена микроконтроллером с минимальной обвязкой, внешней flash-памятью и встроенным программатором с поддержкой в различных средах разработки, включая Arduino, PlatformIO и другое программное обеспечение.

Это первый отечественный микроконтроллер, основанный на архитектуре RISC-V. Он отличается своими компактными размерами, низким расходом электроэнергии и универсальностью, благодаря чему отлично подходит для промышленных систем автоматизации и IoT-устройств. Устройство оснащено встроенными ШИМ, АЦП с разрядностью 12 бит, а также интерфейсами UART, SPI и I²C, обеспечивающими взаимодействие с периферийными устройствами.

2. Отладочный комплект для микросхемы К1986ВЕ92FI [14]:

Этот комплект предназначен для того, чтобы познакомить пользователей с возможностью микроконтроллера К1986ВЕ92F1I [15], а также для получения практических навыков его программирования.

Сам микроконтроллер К1986ВЕ92F1I, построен на базе архитектуры ARM Cortex-M3, и отлично подходит для управления встроенными системами, в том числе электроприводами. Работает на частоте до 80 МГц, оснащен 128 КБ Flash-памяти и 32 КБ ОЗУ, что позволяет эффективно решать задачи в реальном времени. Имеет такие периферийные модули, как 12-битный АЦП, 16-канальный ШИМ и интерфейсы CAN, UART, SPI, обеспечивающими гибкость интеграции.

3. КФДЛ.441461.029 [16]:

Макетно-отладочная плата для микроконтроллера К1921ВГ015 [17], предназначена для изучения архитектуры 32-разрядного микроконтроллера, а также для макетирования и отладки систем пользователя на ее основе.

Микроконтроллер, специально разработанный для энергосберегающих приложений, включая IoT-устройства, приборы учета и управление электроприводами. Поддерживает тактовую частоту до 50 МГц. Среди периферийных возможностей – ШИМ, многоканальный АЦП и интерфейсы связи CAN, UART, SPI.

4. КФДЛ.441461.018 [18]:

Макетно-отладочная плата для микроконтроллера 1921ВК035 [19], с возможностью подключения ко всем портам микроконтроллера, подходит для отладки систем.

Микроконтроллер рассчитан на применение в промышленности и измерительных приборах. Работает на частоте до 100 МГц, имеет в составе интерфейсы связи CAN, UART и SPI, а также встроенные модули ШИМ и АЦП.

5. КФДЛ.441461.024 [20]:

Макетно-отладочная плата, предназначенная для работы с микроконтроллером 1921ВК028 [21]. Обеспечивает возможность отладки и тестирования в составе систем.

Сам микроконтроллер отличается высокой производительностью с ядром, обеспечивающим работу на тактовой частоте до 200 МГц. Среди Его особенностей – поддержка современных интерфейсов связи, а также встроенные модули ШИМ и АЦП. Большой объем памяти 2 МБ Flash и 704 КБ ОЗУ, которые позволяют решать сложные ресурсоемкие задачи, например управление промышленными электроприводами, медицинским оборудованием и автоматизированными производственными линиями.

Таким образом, рассмотренные микроконтроллеры демонстрируют широкое разнообразие характеристик и возможностей, необходимых для

реализации современных систем управлениями. Однако, чтобы выбрать наиболее подходящее решение для конкретных задач, требуется провести сравнительный анализ этих устройств, оценив их преимущества и ограничения по ключевым параметрам.

1.7. СРАВНИТЕЛЬНЫЙ АНАЛИЗ МИКРОКОНТРОЛЛЕРОВ

В данном разделе представлен сравнительный анализ микроконтроллеров, основанный на ключевых характеристиках, которые позволят выбрать наиболее подходящее решение. Для удобства восприятия все данные сведены в таблицу 1.

Таблица 1 – Сравнение характеристик микроконтроллеров

Хар-ка	MIK32 «Амур»	K1986BE92F1I	K1921ВГ015	K1921ВК035	K1921ВК028
Архитектура	RISC-V	ARM CortexM3	RISC-V	RISC-V	RISC-V
Рабочая частота, МГц	До 32	До 80	До 50	До 100	До 200
Память (Flash/ОЗУ), Кб	Внешняя / 16	128 / 32	1024 / 320	64 / 16	2048 / 704
АЦП, бит/каналов	1 * (12/8)	2 * (12/16)	1 * (16/8)	1 * (12/4)	4 * (12/12)
ШИМ	Да, без комплементарных выходов	Да, с комплементарными выходами	Да, без комплементарных выходов	Да, с комплементарными выходами	Да, с комплементарными выходами
Интерфейсы	UART, SPI, I2C	CAN, UART, SPI, I2C	CAN, USB, UART, SPI, I2C, QSPI	CAN, SPI, I2C	CAN, UART, SPI, I2C
Таймеры	3 * Таймер32 3 * Таймер16	3 * Таймер16	1 * Таймер32 3 * Таймер16	4 * Таймер32	8 * Таймер32
Порты ввода-вывода	2 * 16, 1 * 8-разрядных	6 * 16-разрядных	3 * 16-разрядных	2 * 16-разрядных	12 * 16-разрядных

ВЫВОДЫ ПО ПЕРВОЙ ГЛАВЕ

Сравнение представленных микроконтроллеров показало, что все рассмотренные модели соответствуют требованиям для управления электроприводами и могут использоваться в автоматизированных системах различного уровня сложности: от управления маломощными двигателями до реализации векторного управления синхронными и асинхронными электродвигателями.

Более простые микроконтроллеры, такие как MIK32 «Амур» и К1921ВГ015, подходят для управления шаговыми или вентильными двигателями без использования векторных алгоритмов. В то же время К1986ВЕ92F1I, К1921ВК035 и К1921ВК028 обладают ресурсами, необходимыми для построения сложных систем управления с использованием векторного и скалярного методов.

В рамках работы, выполняемой в региональной молодежной лаборатории электромеханических, электронных и электрохимических систем, целью которой является разработка электронного блока управления транспортного средства на отечественной элементной базе, было поручено изучить возможности микроконтроллера MIK32 «Амур для применения в системах управления. Этот микроконтроллер выбран для дальнейшей работы с учётом доступности его отладочного комплекта и достаточности его характеристик для решения задач.

2. РАЗРАБОТКА АЛГОРИТМА УПРАВЛЕНИЯ ЭЛЕКТРОПРИВОДОМ ТЯГОВОГО ДВИГАТЕЛЯ

2.1. ПОСТАНОВКА ЗАДАЧИ И ВЫБОР МЕТОДА УПРАВЛЕНИЯ

В современных системах к электроприводам предъявляются высокие требования по точности регулирования, энергоэффективности, надёжности и устойчивости к внешним воздействиям. Эти требования особенно актуальны для тяговых приводов, где важны как динамические характеристики, так и надёжная работа в различных режимах – включая реверс, пуск с нагрузкой, аварийное торможение и т. д.

Для обеспечения этих характеристик в качестве исполнительного механизма был выбран вентильный двигатель постоянного тока (BLDC). Его выбор обусловлен рядом преимуществ по сравнению с классическими коллекторными или асинхронными двигателями:

- высокая энергоэффективность за счёт отсутствия щёток и низких потерь на трение;
- высокая удельная мощность и компактность;
- стабильная работа при высоких скоростях вращения;
- низкие эксплуатационные требования и повышенный ресурс.

Еще одной особенностью BLDC-двигателя является отсутствие механической коммутации – переключение обмоток осуществляется электронным способом, на основе информации о положении ротора. Это требует наличия системы управления, которая способна:

- определять положение ротора;
- формировать сигналы коммутации;
- регулировать напряжение на обмотках, управляя скоростью вращения.

Эффективность работы такой системы в большей степени зависит от выбранного алгоритма управления. Разные подходы обеспечивают различный

уровень точности, плавности хода и быстродействия системы. Существуют несколько распространённых методов управления BLDC-двигателями:

1. Шестишаговая (трапецеидальная) коммутация.

Это один из самых распространенных и лёгких способов управления. Он заключается в том, что фазы включаются и выключаются по очереди в шести разных фиксированных положениях за один оборот ротора. Чтобы определить, когда именно нужно менять фазы, чаще всего используют датчики Холла или бездатчиковый метод с анализом обратной ЭДС. Скорость двигателя меняется путём изменения скважности широтно-импульсной модуляции. Этот метод прост и надёжен, но даёт пульсации в крутящем моменте, особенно при малых оборотах.

2. Векторное (поле-ориентированное) управление.

Метод, представляющий более трудный путь. Он строится на изменении не только величины и частоты подаваемого напряжения, но и угла подачи тока в обмотки двигателя относительно положения ротора. При этом ток формируется так, чтобы его вектор всё время находился под углом 90° к магнитному полю ротора, что позволяет достигать максимального крутящего момента при минимальных потерях. Для реализации этого метода требуется точное знание положения ротора и выполнение математических преобразований, которые позволяют задавать токи в нужных координатах.

3. Управление с помощью пропорционально-интегрально-дифференцирующего (ПИД) регулятора.

В этом методе используется ПИД-регулятор для управления координатой – например, положением или скоростью ротора. Регулируемая координата изменяется во времени в соответствии с заданной траекторией, такую переменную называют бегущей позицией. Разница между текущей и бегущей позицией служит входом для регулятора, а его выходом является модуль вектора тока, подаваемого на двигатель. Этот токовой вектор удерживается перпендикулярно ротору, обеспечивая формирование крутящего момента. Метод позволяет добиться высокой точности позиционирования и

гибкости управления, однако требует настройки трёх коэффициентов – пропорционального, интегрального и дифференциального, которые и будут определять поведение системы.

4. Бездатчиковое управление.

Этот метод не использует физические датчики положения ротора. Вместо этого положение определяют косвенно – по изменениям обратной электродвижущей силы (ЭДС), возникающей в неактивной фазе двигателя. Когда двигатель вращается, в его обмотках наводится ЭДС, и по её анализу можно определить момент для переключения фаз. Данное управление удешевляет конструкцию и повышает надёжность за счёт отсутствия уязвимых датчиков, но усложняет схему управления и требует точных алгоритмов фильтрации и обработки сигналов, особенно при низких оборотах, где ЭДС мала.

Для реализации надёжной системы управления в данной работе был выбран метод 6-шаговой (трапецеидальной) коммутации с использованием встроенных датчиков Холла и ШИМ для регулирования скорости. Он метод обладает достаточной простотой и надёжностью, что облегчает аппаратную реализацию и отладку, а датчики Холла обеспечивают точное определение положения ротора без сложных вычислительных алгоритмов.

В работе предполагается реализовать такую систему на базе отечественного микроконтроллера, что обеспечит гибкость, компактность и возможность модификации алгоритма.

2.2. ОПИСАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ

В состав тягового электропривода входят следующие компоненты:

- вентильный двигатель постоянного тока – он преобразует электрическую энергию в механическую. Оснащён встроенными датчиками Холла, расположенными с угловым сдвигом 120°. Датчики формируют цифровые сигналы, соответствующие текущему положению ротора;

- трехфазный инвертор – силовой модуль, он преобразует управляющие сигналы (ШИМ) в напряжения, подаваемые на обмотки двигателя;
- датчики тока – служат для измерения тока в фазах двигателя. Аналоговые выходы сигналов с датчиков подаются на входы АЦП микроконтроллера, что позволяет контролировать нагрузку в обмотках двигателя, реализовывать защиту от перегрузки и короткого замыкания, а также реализовать обратную связь в замкнутом контуре управления;
- микроконтроллер – выполняет такие функции как обработку сигналов от датчиков положения и тока, генерацию ШИМ-сигналов для инвертора, реализует алгоритм коммутации, прием управляющий команд от системы верхнего уровня по интерфейсу связи, отправку информации о состоянии в верхний уровень.

Для наглядности на рисунке 3 представлена структурная схема, включающая в себя микроконтроллер, инвертор, вентильный двигатель с датчиками Холла и датчики тока.

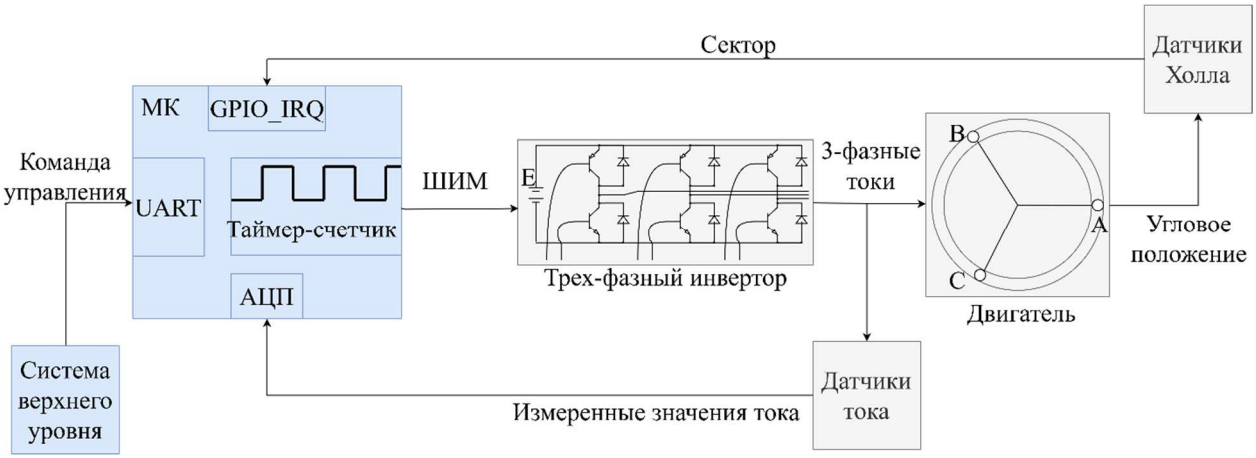

Рисунок 3 – Структурная схема системы управления электроприводом

В качестве взаимодействие микроконтроллера с внешней системой был выбран интерфейс UART, благодаря его:

- простоте интеграции с ПК, другими контроллерами и управляющими устройствами;
- достаточной пропускной способности для передачи параметров (скважность, направление, аварийные сигналы);

- распространённости и доступности в большинстве микроконтроллеров.

В рамках разработки программного модуля системы требуется реализовать алгоритм, обеспечивающий:

- приём внешних команд управления;
- определение положения ротора по сигналам с датчиков Холла;
- коммутацию фаз двигателя в соответствии с положением ротора;
- формирование ШИМ-сигналов заданной скважности;
- считывание и анализ данных с датчиков тока;
- реализацию защитных механизмов.

Алгоритм должен быть устойчивым, легко модифицируемым и пригодным для применения в составе тягового электропривода.

2.3. ОПИСАНИЕ АЛГОРИТМА УПРАВЛЕНИЯ

На рисунке 4 представлена блок-схема алгоритма системы управления.

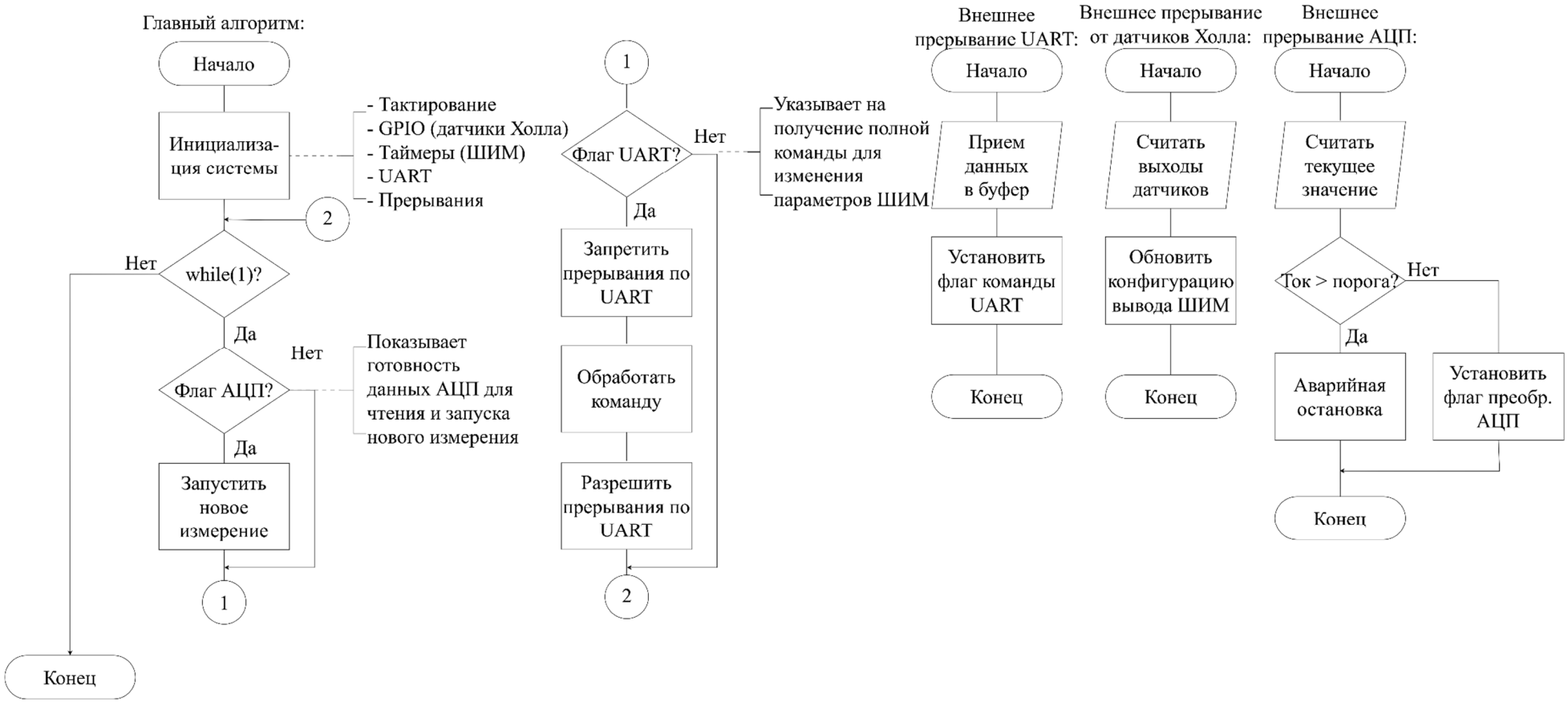

Рисунок 4 – Блок-схема алгоритма управления

Программа работает с использованием прерываний: получения сигналов от датчиков Холла, результатов измерения тока через АЦП и получения управляющий команд по интерфейсу связи. Основной цикл находится в режиме ожидания, что позволяет экономить ресурсы микроконтроллера и обеспечить быстрое реагирование на внешние события. Такой подход присущ встроенным системам реального времени, где важна мгновенная реакция на изменения параметров. Архитектура программы ориентирована на модульность, что позволит упростить отладку и возможную модернизацию системы.

Основные составляющие алгоритма включают в себя:

- обработку прерываний от сигналов с датчиков Холла для определения положения ротора и формирования управляющих сигналов коммутации фаз двигателя;
- обработку данных с датчиков тока, поступающих на АЦП, для контроля нагрузки;
- приём и обработку управляющих команд, поступающих по UART, позволяющих задавать параметры работы двигателя;
- генерацию ШИМ сигналов для управления инвертором, обеспечивая регулирование напряжения и скорости вращения двигателя.

Каждый из этих процессов реализован с учётом приоритетов и времени реакции, что важно при работе с тяговыми электроприводами.

Данный подход обеспечивает баланс между быстродействием и эффективным использованием вычислительных ресурсов, что особенно важно для работы в условиях ограниченных возможностей микроконтроллера и жестких требований к надежности и стабильности работы. Кроме того, система обладает легкой масштабируемостью, что позволит дать возможности для её применения в других типах электроприводов.

ВЫВОДЫ ПО ВТОРОЙ ГЛАВЕ

Во второй главе была сформулирована задача управления тяговым электроприводом на базе вентильного двигателя постоянного тока, а также определена структурная схема системы управления, включающая в себя основные компоненты системы и связи между ними. Были определены требования к алгоритму управления, такие как быстродействие, точность регулирования и устойчивость к внешним воздействиями.

Проведён сравнительный анализ различных методов управления электродвигателем, в ходе которого были рассмотрены их преимущества и недостатки. По результатам анализа был выбран метод шестишаговой (трапецеидальной) коммутации с использованием встроенных датчиков Холла и ШИМ для регулирования скорости.

На основе выбранного метода разработан общий алгоритм системы управления, структура которого реализована с использованием системы прерываний. Такой подход позволяет обеспечить быструю реакцию на внешние воздействия и одновременно эффективное использование ресурсов микроконтроллера. Для наглядного представления логики работы системы управления разработана подробная блок-схема алгоритма, демонстрирующая все этапы процесса управления и их взаимосвязи.

3. НАПИСАНИЕ ПРОГРАММНОГО КОДА ДЛЯ УПРАВЛЕНИЯ РАБОТОЙ ЭЛЕКТРОПРИВОДА

3.1. АППАРАТНАЯ ПЛАТФОРМА И СРЕДСТВА РАЗРАБОТКИ

В качестве аппаратной платформы использовалась отладочная плата «Старт», основанная на отечественном микроконтроллере MIK32 «Амур» (К1948ВК018). Микроконтроллер реализован на архитектуре RISC-V и оснащён встроенными периферийными модулями, включая ШИМ-генераторы, АЦП, таймеры и интерфейс UART, что позволяет реализовать необходимые функции управления двигателем.

Плата включает в себя:

- микроконтроллер MIK32;
- внешнюю flash-память объёмом 4 МБ;
- встроенный стабилизатор напряжения 3,3 В;
- разъём Micro-USB для питания и обмена данными;
- встроенный отладчик, совместимый с OpenOCD и поддерживающий режим USB-UART;
- кварцевые резонаторы на 32 МГц и 32,768 кГц;
- базовый набор кнопок и индикаторов (reset, wakeup, пользовательская кнопка, светодиоды);
- цифровые входы/выходы для подключения внешних модулей.

Программирование микроконтроллера осуществляется на языке C с использованием среды PlatformIO (на базе Visual Studio Code). Эта среда поддерживает автоматическую установку необходимых инструментов, включая компилятор RISC-V, отладчик и средства сборки, а также предоставляет удобный механизм организации проекта, компиляции, загрузки прошивки и отладки.

3.2. СТРУКТУРА ПРОГРАММНОГО МОДУЛЯ

Программа разделена на отдельные модули для удобства разработки, отладки и последующей модификации. Каждый функциональный блок реализован в виде самостоятельного модуля, состоящего из файла реализации (.c) и соответствующего заголовочного файла (.h).

Основные модули программы:

- main.c (листинг А.1 приложения А) – основной файл, содержащий функцию main(), где производится инициализация модулей, настройка системы и переход к основному циклу;
- hall.h / hall.c (листинги А.2 и А.3 приложения А) – модуль обработки сигналов с датчиков Холла. Он отвечает за определение положения ротора и вызов функции обновления состояния коммутации;
- pwm.h / pwm.c (листинги А.4 и А.5 приложения А) – модуль управления широтно-импульсной модуляцией. Помимо генерации ШИМ-сигналов, реализована функция коммутации фаз двигателя;
- adc.h / adc.c (листинги А.6 и А.7 приложения А) – модуль взаимодействия с аналогово-цифровым преобразователем. Он используется для измерения токов от сигналов датчиков;
- usart.h / usart.c (листинги А.8 и А.9 приложения А)– модуль взаимодействия с интерфейсом UART. Он обеспечивает приём команд и передачу данных о состоянии системы.

Функция main() отвечает за последовательный вызов функций инициализации этих модулей, запуск таймеров и переход к основному циклу. В цикле реализована логика приёма команд, а реакция на внешние события осуществляется через аппаратную систему прерываний.

Прерывания используются для:

- обработки изменения состояния датчиков Холла;
- фиксации завершения преобразования в АЦП;
- приёма данных через UART.

Такой подход обеспечивает стабильную работу системы в режиме реального времени и упрощает масштабирование проекта.

3.3. РЕАЛИЗАЦИЯ ОБРАБОТКИ СИГНАЛОВ С ДАТЧИКОВ ХОЛЛА

Для определения положения ротора в бесщёточном двигателе постоянного тока используются три датчика Холла. Каждый из них подключён к отдельному пину микроконтроллера и сконфигурирован как вход с прерыванием по изменению сигнала. Это позволяет точно и своевременно реагировать на переход ротора в следующую позицию, что критически важно для корректной коммутации фаз.

Инициализация пинов и настройка прерываний выполняется в функции GPIO_Init_Hall_Sensors(), которая определена в модуле hall.c. В этой функции включается тактирование GPIO, устанавливаются режимы входов и активируются прерывания на изменение состояния каждого из трёх входных сигналов.

Одной из особенностей MIK32 является его система обработки прерываний: все внешние события обрабатываются в рамках единого программного обработчика. Идентификация источника прерывания осуществляется путём программной проверки состояния входов с использованием условных конструкций. Вложенные прерывания не поддерживаются. Такой подход возлагает на программный уровень всю ответственность за определение источника прерывания и порядок их обработки, что усложняет разработку и не позволяет реализовать сложные системы с приоритетной обработкой.

При возникновении внешнего прерывания вызывается функция update_commutation(). Внутри неё вызывается get_hall_sensor_state(), которая формирует трёхбитную маску текущего состояния датчиков (A, B и C) и возвращает её. Полученный код состояния используется в функции

commutation(), определённой в модуле pwm.c, для выбора соответствующего шага коммутации на основе таблицы коммутации.

3.4. РЕАЛИЗАЦИЯ ОБРАБОТКИ ДАННЫХ С ДАТЧИКОВ ТОКА

Для контроля тока в фазах двигателя и обеспечения защиты инвертора в проекте реализована система обратной связи на основе встроенного АЦП микроконтроллера. Сигналы тока снимаются с шунтов и подаются на аналоговые входы, где преобразуются в цифровую форму.

Инициализация модуля осуществляется в файле adc.c функцией ADC_Init(). В ней задаётся канал измерения (ADC_CHANNEL5), внутреннее опорное напряжение и режим калибровки. В качестве опорного используется встроенный источник, а калибровка проводится по внутреннему стандарту.

Измеренное значение сохраняется в переменную adc_value. Проверка на превышение тока выполняется через функцию ADC_CheckOvercurrent(), которая сравнивает полученное значение с программно заданным порогом ADC_OVERCURRENT_THRESHOLD. При превышении порога будет вызвана функция безопасной остановки фаз safe_switch_phase() из модуля управления ШИМ.

Такой подход позволяет в реальном времени контролировать ток и реагировать на аварийные ситуации, повышая надёжность и безопасность системы.

3.5. УПРАВЛЕНИЕ ШИМ-СИГНАЛАМИ ДЛЯ ИНВЕРТОРА

Формирование ШИМ-сигналов является основой для управления фазами инвертора и, соответственно, двигателем. Они генерируются с помощью двух таймеров TIMER32_1 и TIMER32_2, каждый из которых управляет отдельной группой каналов. Конфигурация таймеров и каналов, а также реализация алгоритма коммутации, выполнены в модуле pwm.c.

В процессе инициализации, реализованной в функции pwm_init(), настраиваются параметры таймеров, такие как значение TOP-регистра

(pwm_top), режим счёта и источник тактирования. Каждый канал таймера конфигурируется в режим ШИМ.

Вся логика коммутации между фазами реализована в функции commutation(), которая управляет шестью возможными состояниями фаз (шагами коммутации) согласно таблице comm_table. В таблице указывается, какие каналы должны быть активны для каждой фазы в каждом состоянии, а функция safe_switch_phase() реализует переключение с учётом "мёртвого времени" (dead time) для предотвращения возникновения сквозного тока.

Кроме основной генерации сигналов, также реализована поддержка изменение направления вращения двигателя через переменную motor_direction, влияющую на порядок включения фаз.

3.6. ОБРАБОТКА КОМАНД ПО ИНТЕРФЕЙСУ UART

Для внешнего управления системой предусмотрена передача и приём управляющих команд через универсальный асинхронный приёмопередатчик. Это позволяет напрямую взаимодействовать с контроллером: задавать направление вращения двигателя, изменять параметры работы или инициировать действия, такие как запуск и остановка.

Настройка интерфейса UART реализована в модуле usart.c. В функции USART_Init() осуществляется полная конфигурация регистров структуры USART_HandleTypeDef, в том числе установка режима работы, битности, стоп-битов и скорости передачи данных. Вся инициализация производится через вызов функции HAL_USART_Init() из аппаратно-абстрактного уровня библиотеки HAL.

В рамках обработки приёма данных в буфере rx_buf накапливаются входящие байты. Управление индексом буфера осуществляется через переменную rx_index, а факт завершения приёма фиксируется флагом rx_complete. Дополнительно используется флаг start_char_received для отслеживания начала команды.

3.7. РЕАЛИЗАЦИЯ ОСНОВНОЙ ПРОГРАММЫ И ОБРАБОТКИ ПРЕРЫВАНИЙ

Файл main.c содержит основную управляющую логику микроконтроллера, объединяющую все подсистемы: генерацию ШИМ, обработку сигналов датчиков Холла и приём команд по UART. Здесь реализуется инициализация всех аппаратных модулей, запуск таймеров, установка начальных параметров и основной цикл ожидания команд.

В процессе инициализации компонентов последовательно вызываются:

- HAL_Init() – базовая инициализация системы;
- USART_Init() – настройка UART для приёма управляющих команд;
- SystemClock_Config() – конфигурация системного тактирования;
- GPIO_Init_Hall_Sensors() – инициализация GPIO для считывания с датчиков Холла;
- pwm_init() – запуск таймеров и ШИМ-каналов;
- HAL_Timer32_Start(...) – запуск двух таймеров;
- установка начального заполнения ШИМ в 0% и отключение всех фаз через safe_switch_phase().

Основной цикл (while(1)) реализован по схеме «ожидание события – обработка – возврат в ожидание». Обработка производится, когда установлен флаг rx_complete, который означает, что по UART поступила полная строка команды.

После получения строки:

1. Прерывания UART временно отключаются.
2. Вызовом ProcessInput() происходит преобразование строки в целочисленное значение и установка соответствующего уровня ШИМ.
3. Буферы и флаги сбрасываются, прерывания включаются обратно.

В функции trap_handler() реализуется обработка асинхронных событий:

- прерывания от GPIO (датчики Холла): вызывается update_commutation(), если произошло изменение на одной из линий датчиков;
- прерывание по приёму байта по UART: реализован простой протокол, при котором команда начинается с символа d, после чего считывается строка чисел до символа новой строки (\n или \r), и сохраняется в буфер rx_buf.

  Также реализована простая защита от переполнения буфера.

ВЫВОДЫ ПО ТРЕТЬЕЙ ГЛАВЕ

В данной главе была реализована программная часть управления бесщёточным двигателем с использованием микроконтроллера. Основные результаты включают:

- разделение функционала на отдельные модули – PWM, USART, обработка прерываний, что повысило читаемость и удобство сопровождения кода;
- настройку и инициализацию периферии (таймеры, UART, ADC);
- реализацию безопасной коммутация фаз с учётом «dead time» для предотвращения коротких замыканий;
- организацию цикла обработки пользовательских команд и внешних прерываний;
- реализацию обработчика прерываний, который позволяет быстро реагировать на внешние события.

Таким образом, программная архитектура обеспечивает надежное управление двигателем и удобный интерфейс настройки через UART, что является основой для дальнейшего развития системы.

4. ПРОВЕДЕНИЕ ТЕСТИРОВАНИЕ ПРОГРАММНОГО МОДУЛЯ

4.1. МЕТОДИКА ТЕСТИРОВАНИЯ

Для верификации работоспособности системы использовался комплексный функциональный подход, включающий:

1. Проверку корректности работы алгоритмов управления (коммутация, ШИМ).
2. Проверку защиты от перегрузки по току с помощью АЦП.
3. Верификацию обработки сигналов с датчиков Холла.

В качестве инструментов использовались**:**

- отладочная плата СТАРТ-MIK32;
- электродвигатель ДВМ100.22;
- измерительное оборудование: осциллограф Hantek DSO5102P;
- терминальная программа Terra Term для мониторинга UART-сообщений.

4.2. ТЕСТИРОВАНИЕ УПРАВЛЕНИЯ ШИМ

Основной задачей данного тестирование было подтверждение корректности задания коэффициента заполнения импульсов, которые формируются на выходе таймеров. Проверка проводилась путём подачи различных управляющих команд через UART и последующего измерения выходного сигнала на одном из каналов таймера с помощью осциллографа.

Процедура тестирования:

1. С помощью UART-интерфейса вручную подавались команды на изменение скважности сигнала. Формат команды – d <скважность> <направление движения>, где скважность указывается в процентах.

2. Были последовательно поданы команды:

- d 20 0;
- d 50 0;
- d 80 0.

3. Сигнал снимался с одного из выходных каналов таймера с помощью осциллографа.

В результате на выходе наблюдались сигналы, соответствующие заданным скважностям. Измеренные формы сигналов подтверждают корректную работу ШИМ-модуля, что представлено на осциллограммах (рисунки 5, 6, 7).

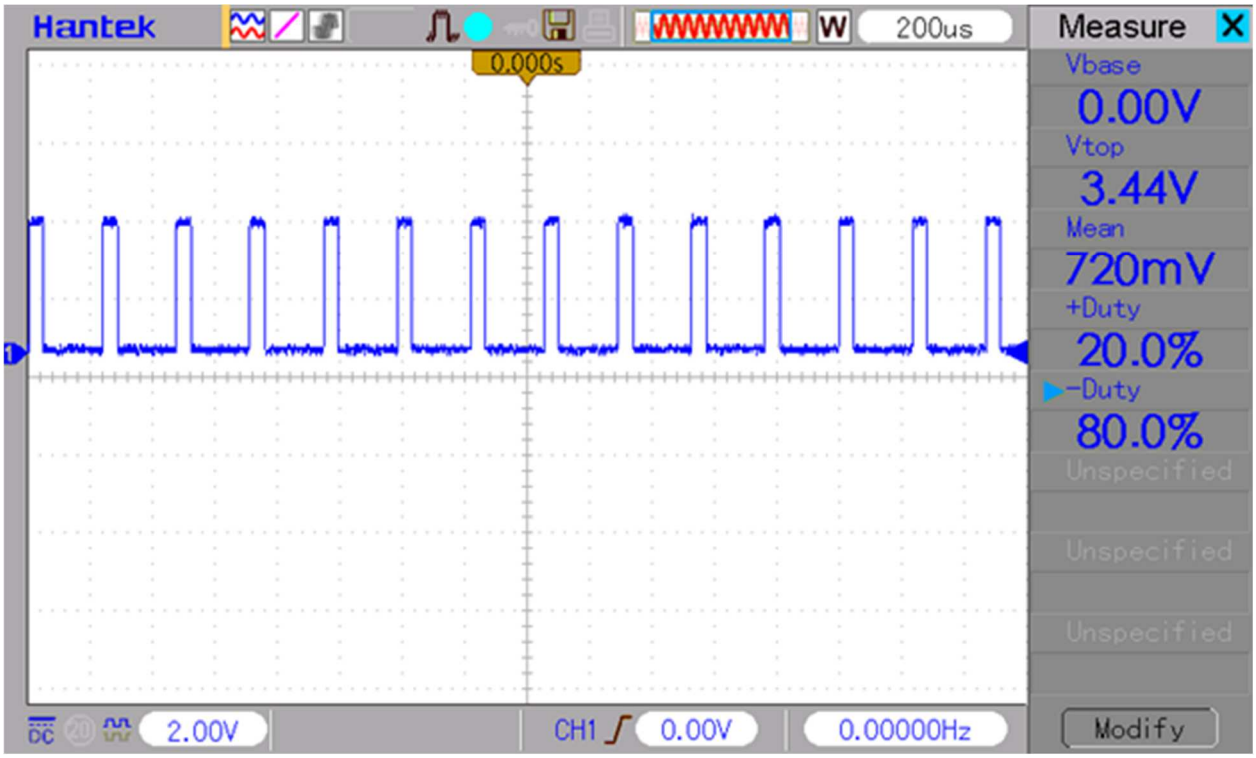

Рисунок 5 – Осциллограмма сигнала с установленной скважностью 20%

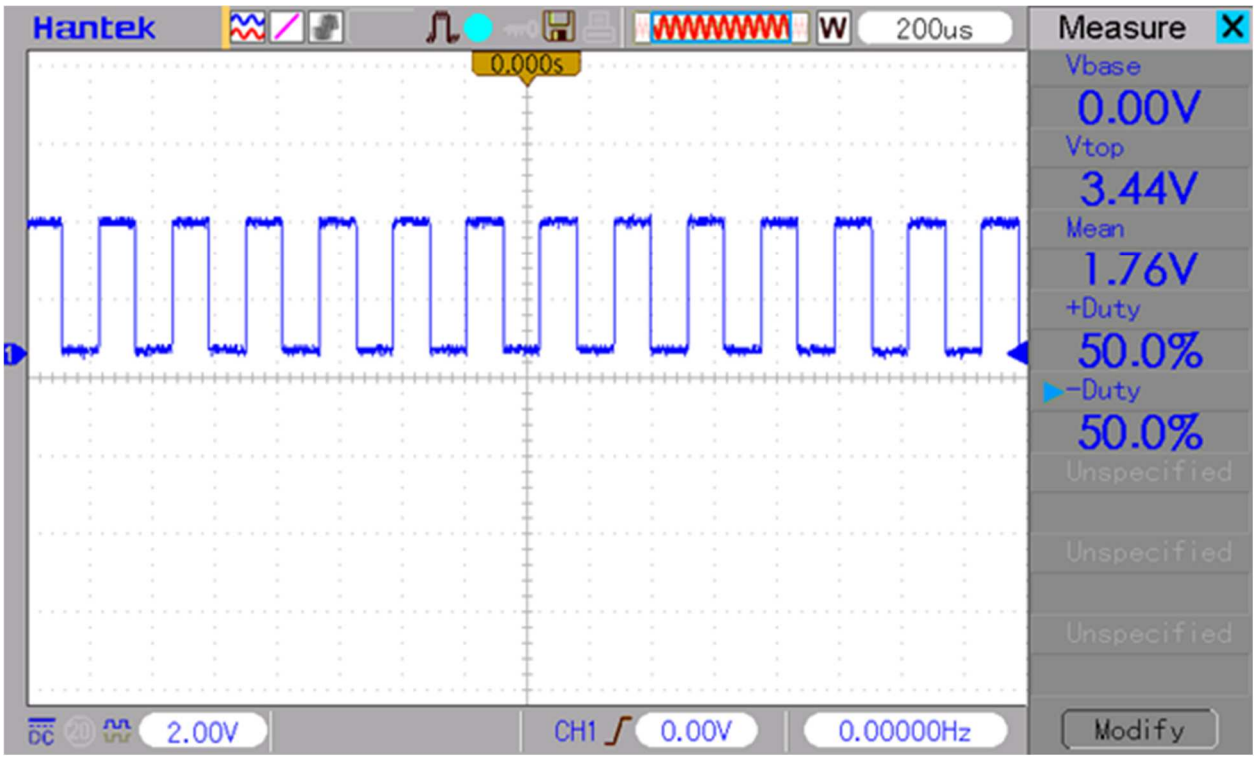

Рисунок 6 – Осциллограмма сигнала с установленной скважностью 50%

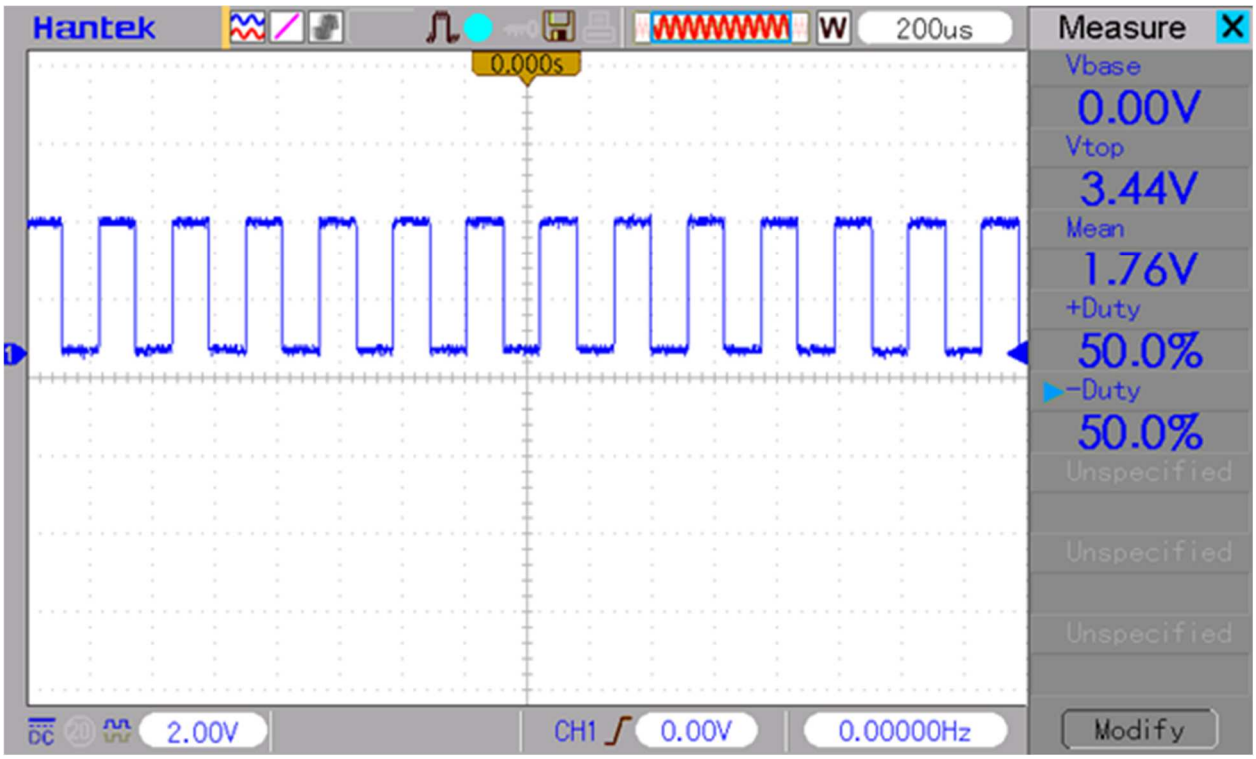

Рисунок 7 – Осциллограмма сигнала с установленной скважностью 80%

4.3. ТЕСТИРОВАНИЕ ОБРАБОТКИ СИГНАЛОВ С ДАТЧИКОВ ХОЛЛА

Для управления положением ротора в системе использовались три датчика Холла, подключённые к портам GPIO1_8, GPIO1_9 и GPIO1_10 микроконтроллера. Датчики формируют трехзначный код положения ротора, который используется в функции коммутации фаз двигателя.

Процедура тестирования:

1. Сигналы подавались посредством ручной прокрутки ротора.
2. В обработчике прерываний происходила проверка текущих состояний всех трёх датчиков с последующим вызовом функции update_commutation().
3. Для верификации работы алгоритма данные о состоянии датчиков и вызовах коммутации выводились в UART.

В результате все три датчика Холла корректно регистрировали изменения состояния и вызывали прерывания.

Система реагировала мгновенно, без пропусков или ложных срабатываний.

Коммутация фаз строго соответствовала заданной таблице.

В процессе тестирования в UART фиксировались следующие события (рисунок 8).

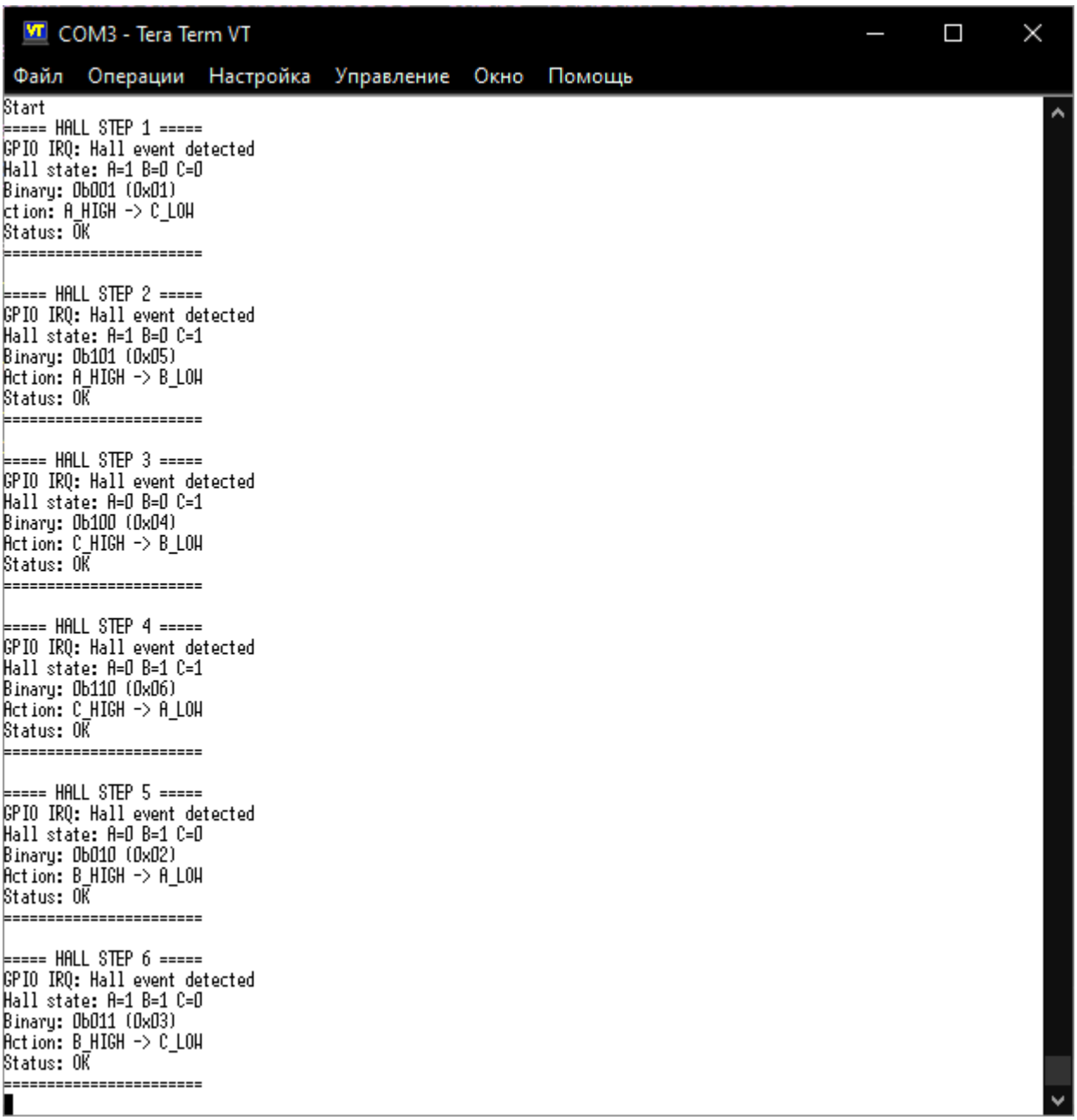

Рисунок 8 – Окно вывода UART с отображением состояний датчиков Холла и команд коммутации фаз

4.4. ТЕСТИРОВАНИЕ ЗАЩИТЫ ОТ ПЕРЕГРУЗКИ ПО ТОКУ

Для обеспечения безопасности работы двигателя была реализована защита от перегрузки по току, которая контролирует значение тока через показания аналогово-цифрового преобразователя (АЦП). Защитный порог установлен на уровне, который соответствует максимальному значению тока в двигателе.

Процедура тестирования:

1. Нагрузка на двигатель постепенно увеличивалась, при этом значения АЦП выводились в UART в режиме реального времени.
2. При достижении или превышении порогового значения система отключала фазы двигателя для предотвращения повреждений.
3. В UART выводились сообщения о текущих значениях АЦП и срабатывании защиты.

В процессе тестирования в UART фиксировались следующие события рисунок 9):

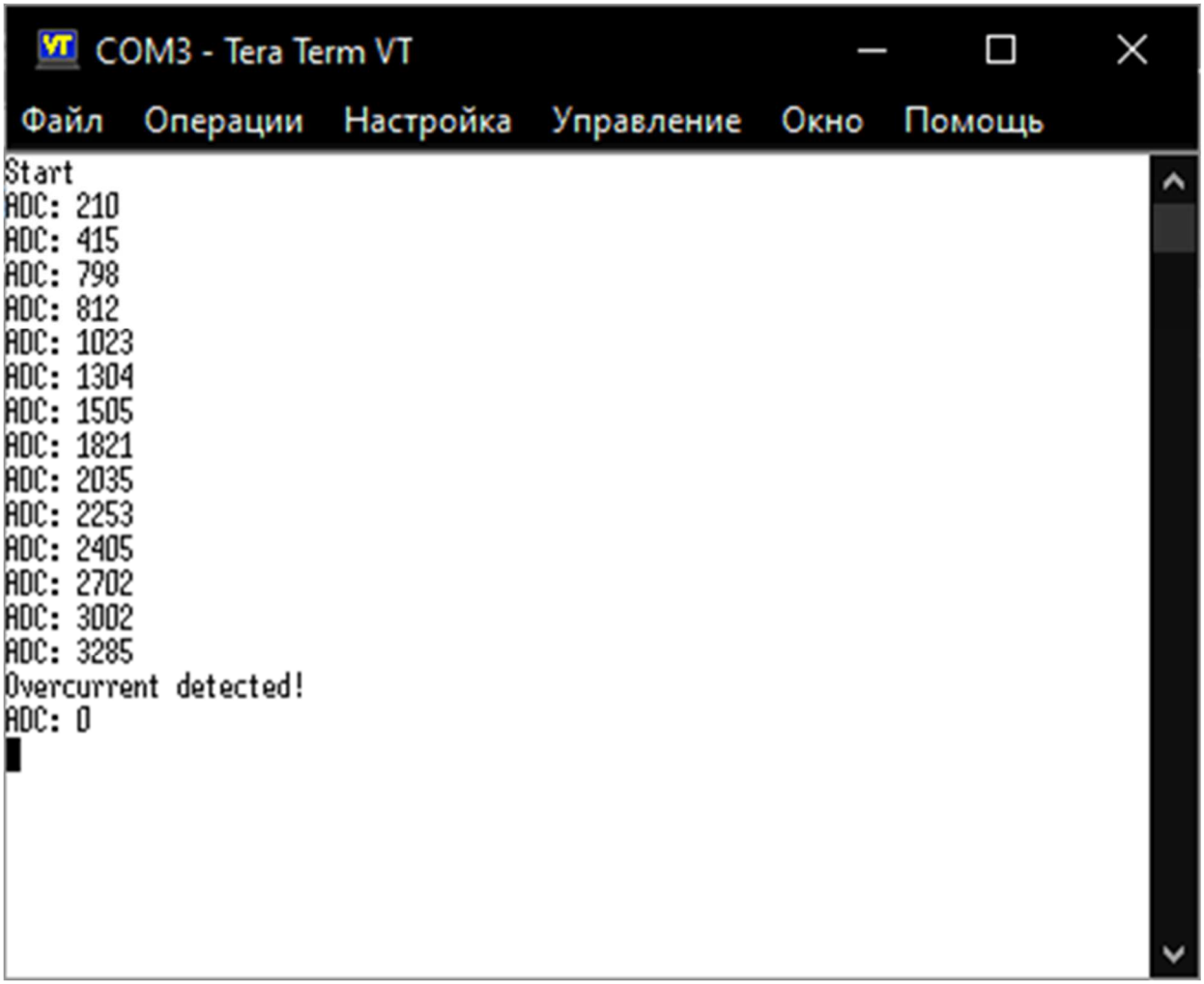

Рисунок 9 – Окно вывода UART с результатами тестирования защиты от перегрузки по току с показаниями АЦП и состоянием системы

В результате теста система своевременно фиксировала превышение порога тока. Защита отключала питание фаз двигателя, предотвращая

возможные повреждения. Логика работы АЦП и обработки данных подтверждена корректностью срабатывания в тестовых условиях

ВЫВОДЫ ПО ЧЕТВЕРТОЙ ГЛАВЕ

В ходе выполнения комплексного тестирования системы были подтверждены корректность и стабильность работы всех ключевых модулей:

Защита по току с использованием АЦП надёжно срабатывает при превышении пороговых значений и эффективно отключает фазы двигателя.

Обработка сигналов с датчиков Холла реализована корректно, обеспечивая мгновенную и стабильную коммутацию фаз в соответствии с положением ротора.

Управление ШИМ выполнено качественно, с точным формированием сигнала заданной скважности, что подтверждается осциллограммами.

Таким образом, проведённые испытания подтвердили соответствие системы заложенным функциональным требованиям и её готовность к эксплуатации в реальных условиях.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была успешно разработана система управления электроприводом тягового двигателя на базе отечественного микроконтроллера MIK32. Основные этапы работы включали:

1. Аналитический обзор литературы: были изучены современные методы управления электроприводами, определены требования к микроконтроллерам и рассмотрены несколько моделей из отечественной элементной базы. Проведен сравнительный анализ микроконтроллеров, в результате которого выбран MIK32 «Амур».
2. Разработку алгоритма: на основе метода 6-шаговой (трапецеидальной) коммутации с ШИМ и использованием сигналов с датчиков Холла разработан алгоритм, обеспечивающий управление двигателем. Алгоритм реализован с использованием системы прерываний для обеспечения быстрой реакции на изменения положения ротора и внешние управляющие команды.
3. Программная реализация: создан модульный программный код, включающий обработку сигналов с датчиков Холла, управление ШИМ, контроль тока через АЦП и взаимодействие по интерфейсу UART.
4. Тестирование: проведены комплексные испытания системы, подтвердившие корректность работы всех модулей. Особое внимание уделено проверке управления ШИМ, обработке сигналов датчиков Холла и защите от перегрузки по току.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бекишев, Р.Ф. Общий курс электропривода: учебное пособие / Р.Ф. Бекишев, Ю.Н. Дементьев. – 2-е изд. – Томск: ТПУ, 2014. – 302 с.
2. Microcontrollers Types & Their Applications – https://www.elprocus.com/microcontrollers-types-and-applications.
3. Заботнов, П.В. Состояние и возможности развития Российской микроэлектронной отрасли / П.В. Заботнов // Наука сегодня: глобальные вызовы, пути развития: Материалы XIV Всероссийской научно-практической конференции. Сер. «Гуманитарные исследования». – Рязань: Издательство ООО «Концепция», 2023. – С. 194-196.
4. «Микрон» – крупнейший российский производитель и экспортер микроэлектронной продукции – https://mikron.ru.
5. АО «Ангстрем» – ведущий производитель полупроводниковых изделий – https://www.angstrem.ru.
6. Чистова, А.А. Российская микроэлектроника на примере компаний Ао «Микрон» и АО «Ангстрем» / А.А. Чистова, О.А. Скрынская. // Актуальные аспекты модернизации российской экономики: Материалы IX Всероссийской заочной научно-практической конференции студентов, аспирантов и молодых ученых. Том 1. / под ред. О.А. Скрынская. – СПб.: СПбГЭТУ «ЛЭТИ», 2022. – С. 313-318.
7. Официальный сайт АО «ПКК Миландр» – https://milandr.ru.
8. АО «НИЭЭТ» – https://niiet.ru.
9. Распоряжение Правительства РФ от 17.01.2020 N 20-р (ред. от 21.10.2024) «Об утверждении Стратегии развития электронной промышленности Российской Федерации на период до 2030 года» – http://government.ru/docs/all/125857.
10. Микроконтроллерные системы управления электроприводами: современное состояние и перспективы развития – https://motorcontrol.ru/wp-content/uploads/2015/11/controllers.pdf.

11. Токарев, В.В. Процессорное управление электроприводами / В.В. Токарев // Современная электроника: электронный журнал. – https://www.cta.ru/articles/soel/2017/2017-8/114481/.
12. RISC-V микроконтроллер MIK32 АМУР – https://mikron.ru/products/mikrokontrollery/mk32-amur.
13. Техническое описание MIK32 – https://nc.mikron.ru/s/aXSRc8HdLAM2LLg/download.
14. Комплект отладочный для микросхемы К1986ВЕ92FI. Паспорт ТСКЯ.468998.145ПС – https://support.milandr.ru/upload/iblock/e0f/qvpnj5hbco3mgqdfwb4p5dnpfbe1bmcq/ТСКЯ.468998.145ПС.pdf.
15. Спецификация. Микросхема 32-разрядного однокристального микро-ЭВМ с памятью Flash-типа К1986ВЕ92FI, К1986ВЕ92F1I, К1986ВЕ94GI – https://support.milandr.ru/upload/iblock/45d/r521w55nr5gb4p0fec6is469aflxdslh/К1986ВЕ92FI, К1986ВЕ92F1I, К1986ВЕ94GI.pdf.
16. КФДЛ.441461.029. Макетно-отладочная плата для микроконтроллера К1921ВГ015 – https://niiet.ru/product/kdfl-441461-018.
17. Микросхема интегральная. К1921ВГ015. Руководство пользователя – https://niiet.ru/wp-content/uploads/2024/10/РП-К1921ВГ015-v3.pdf.
18. КФДЛ.441461.018. Макетно-отладочная плата для микроконтроллера 1921ВК035 – https://niiet.ru/product кфдл-441461-018.
19. Микросхема интегральная. 1921ВК035. Руководство пользователя. – https://niiet.ru/wp-content/uploads/2023/02/РП-1921ВК035.pdf.
20. КФДЛ.441461.024. Отладочная плата для микроконтроллера 1921ВК028 – https://niiet.ru/product/ кфдл-441461-024.
21. Микросхема интегральная. 1921ВК028. Руководство пользователя. – https://niiet.ru/wp-content/uploads/2023/02/РП-1921ВК028.pdf.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММНОГО МОДУЛЯ

Листинг А.1 – Исходный код файла main.c

```
#include <stdbool.h>
#include <stdint.h>
#include "mik32_hal.h"
#include "mik32_hal_irq.h"
#include "mik32_hal_usart.h"
#include "pwm.h"
#include "adc.h"
#include "hall.h"
#include "usart.h"

void SystemClock_Config(void);
void ProcessInput(char* input);
uint8_t simple_atoi(const char *s, bool *is_valid);

int main(void)
{
    HAL_Init();
    USART_Init();
    SystemClock_Config();

    GPIO_Init_Hall_Sensors();
    pwm_init();
    ADC_Init();

    HAL_EPIC_MaskEdgeSet(HAL_EPIC_ADC_MASK);
    HAL_EPIC_MaskLevelSet(HAL_EPIC_GPIO_IRQ_MASK | HAL_EPIC_TIMER16_0_MASK |
HAL_EPIC_TIMER32_1_MASK | HAL_EPIC_TIMER32_2_MASK | HAL_EPIC_UART_0_MASK);
    HAL_IRQ_EnableInterrupts();

    HAL_Timer32_Start(&htimer32_1);
    HAL_Timer32_Start(&htimer32_2);

    pwm_duty_cycle = 0;
    pwm_duty_percent = 0;

    safe_switch_phase(&A_HIGH, &A_LOW, 0, 0);
    safe_switch_phase(&B_HIGH, &B_LOW, 0, 0);
    safe_switch_phase(&C_HIGH, &C_LOW, 0, 0);

    HAL_USART_Print(&husart0, "Start\n", USART_TIMEOUT_DEFAULT);

    HAL_USART_RXNE_EnableInterrupt(&husart0);
    while (1)
    {
        if (conversion_ready_flag) {
            conversion_ready_flag = false;
            HAL_ADC_Single(&hadc);  // Запускаем следующее преобразование
        }
        // Ожидаем завершение приема
        if(rx_complete)
        {
            // Отключаем прерывания на время обработки
            HAL_USART_RXNE_DisableInterrupt(&husart0);
```

Продолжение листинга А.1

```
            // Обработка числа
            ProcessInput(rx_buf);

            // Сброс флагов и буфера
            rx_index = 0;
            rx_complete = false;
            start_char_received = false;
            rx_buf[0] = '\0';

            // Включаем прерывания обратно
            HAL_USART_RXNE_EnableInterrupt(&husart0);
        }
    }
}

void SystemClock_Config(void)
{
    PCC_InitTypeDef PCC_OscInit = {0};

    PCC_OscInit.OscillatorEnable = PCC_OSCILLATORTYPE_ALL;
    PCC_OscInit.FreqMon.OscillatorSystem = PCC_OSCILLATORTYPE_OSC32M;
    PCC_OscInit.FreqMon.ForceOscSys = PCC_FORCE_OSC_SYS_UNFIXED;
    PCC_OscInit.FreqMon.Force32KClk = PCC_FREQ_MONITOR_SOURCE_OSC32K;
    PCC_OscInit.AHBDivider = 0;
    PCC_OscInit.APBMDivider = 0;
    PCC_OscInit.APBPDivider = 0;
    PCC_OscInit.HSI32MCalibrationValue = 128;
    PCC_OscInit.LSI32KCalibrationValue = 8;
    PCC_OscInit.RTCClockSelection = PCC_RTC_CLOCK_SOURCE_AUTO;
    PCC_OscInit.RTCClockCPUSelection = PCC_CPU_RTC_CLOCK_SOURCE_OSC32K;
    HAL_PCC_Config(&PCC_OscInit);
}
bool parse_two_numbers(const char* input, uint8_t* duty, uint8_t* dir)
{
    uint16_t temp = 0;
    int stage = 0;

    *duty = 0;
    *dir = 0;

    while (*input != '\0')
    {
        if (stage == 0)
        {
            // Число до пробела (duty)
            if (*input >= '0' && *input <= '9')
            {
                temp = temp * 10 + (*input - '0');
                if (temp > 100) return false;
            }
            else if (*input == ' ')
            {
                *duty = (uint8_t)temp;
                temp = 0;
                stage = 1;
            }
```

Продолжение листинга А.1

```
                else return false;
            }
            else if (stage == 1)
            {
                // Второе число (dir), допускаем только 0 или 1
                if (*input == '0' || *input == '1')
                {
                    *dir = *input - '0';
                    input++;
                    if (*input != '\0') return false;  // ничего больше после
второго числа
                    return true;
                }
                else return false;
            }

            input++;
        }

        return false;
    }

    void ProcessInput(char* input)
    {
        uint8_t duty, dir;

        if (!parse_two_numbers(input, &duty, &dir))
        {
            HAL_USART_Print(&husart0, "E\n", USART_TIMEOUT_DEFAULT);
            return;
        }

        pwm_duty_percent = duty;
        pwm_duty_cycle = (duty * pwm_top) / 100;
        motor_direction = dir;
        HAL_USART_Print(&husart0, "OK\n", USART_TIMEOUT_DEFAULT);
    }

    void trap_handler(){
        if (EPIC_CHECK_GPIO_IRQ())
        {
            HAL_USART_Print(&husart0, "GPIO\n", USART_TIMEOUT_DEFAULT);
            // Обработка состояния датчика A
            if (HAL_GPIO_LineInterruptState(GPIO_MUX_PORT1_8_LINE_0))
update_commutation();
            // Обработка состояния датчика B
            if (HAL_GPIO_LineInterruptState(GPIO_MUX_PORT1_9_LINE_1))
update_commutation();
            // Обработка состояния датчика C
            if (HAL_GPIO_LineInterruptState(GPIO_MUX_PORT1_10_LINE_2))
update_commutation();
            // Сбрасываем флаг прерывания
            HAL_GPIO_ClearInterrupts();
        }
        if (EPIC_CHECK_ADC())
        {
```

Окончание листинга А.1

```
            adc_value = HAL_ADC_GetValue(&hadc);

            if (ADC_CheckOvercurrent()) {
                safe_switch_phase(&A_HIGH, &A_LOW, 0, 0);
                safe_switch_phase(&B_HIGH, &B_LOW, 0, 0);
                safe_switch_phase(&C_HIGH, &C_LOW, 0, 0);
                // HAL_USART_Print(&husart0, "Overcurrent detected!\n",
USART_TIMEOUT_DEFAULT);
            }
            conversion_ready_flag = true;
        }
        if (EPIC_CHECK_UART_0())
        {
            // Обработка приема данных
            if (HAL_USART_RXNE_ReadFlag(&husart0))
            {
                char c = HAL_USART_ReadByte(&husart0);

                // Ждем стартовый символ 'd'
                if(!start_char_received)
                {
                    if(c == 'd')
                    {
                        start_char_received = true;
                    }
                }
                else
                {
                    // Принимаем данные до символа новой строки или заполнения
буфера
                    if(c == '\n' || c == '\r')
                    {
                        rx_buf[rx_index] = '\0';
                        rx_complete = true;
                    }
                    else if(rx_index < (BUF_MAX - 1))
                    {
                        rx_buf[rx_index++] = c;
                    }
                    else
                    {
                        // Переполнение буфера
                        rx_buf[rx_index] = '\0';
                        rx_complete = true;
                    }
                }
                HAL_USART_RXNE_ClearFlag(&husart0);
            }
        }
        // Сбрасываем все прерывания в EPIC
        HAL_EPIC_Clear(HAL_EPIC_GPIO_IRQ_MASK | HAL_EPIC_UART_0_MASK |
HAL_EPIC_ADC_MASK);
    }
```

Листинг А.2 – Исходный файл hall.h

```
#ifndef HALL_H
#define HALL_H

#include <stdint.h>

void GPIO_Init_Hall_Sensors(void);
uint8_t get_hall_sensor_state(void);
void update_commutation();

#endif // HALL_H
```

Листинг А.3 - Исходный файл hall.c

```
#include "hall.h"
#include "pwm.h"
#include "mik32_hal_gpio.h"

#define HALL_A GPIO_PIN_8
#define HALL_B GPIO_PIN_9
#define HALL_C GPIO_PIN_10

void GPIO_Init_Hall_Sensors(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    // Включаем тактирование GPIO и прерываний
    __HAL_PCC_GPIO_0_CLK_ENABLE();
    __HAL_PCC_GPIO_1_CLK_ENABLE();
    __HAL_PCC_GPIO_IRQ_CLK_ENABLE();

    // Настройка пина для датчика A (PORT1_2)
    GPIO_InitStruct.Pin = HALL_A;  // Пин для датчика A
    GPIO_InitStruct.Mode = HAL_GPIO_MODE_GPIO_INPUT;  // Режим входа
    GPIO_InitStruct.Pull = HAL_GPIO_PULL_DOWN;  // Подтяжка к питанию
    HAL_GPIO_Init(GPIO_1, &GPIO_InitStruct);// Инициализация пина для датчика A

    // Настройка пина для датчика B (PORT1_6)
    GPIO_InitStruct.Pin = HALL_B;  // Пин для датчика B
    GPIO_InitStruct.Mode = HAL_GPIO_MODE_GPIO_INPUT;  // Режим входа
    GPIO_InitStruct.Pull = HAL_GPIO_PULL_DOWN;  // Подтяжка к питанию
    HAL_GPIO_Init(GPIO_1, &GPIO_InitStruct);// Инициализация пина для датчика B


    // Настройка пина для датчика C (PORT1_10)
    GPIO_InitStruct.Pin = HALL_C;  // Пин для датчика C
    GPIO_InitStruct.Mode = HAL_GPIO_MODE_GPIO_INPUT;  // Режим входаs
    GPIO_InitStruct.Pull = HAL_GPIO_PULL_DOWN;  // Подтяжка к питанию
    HAL_GPIO_Init(GPIO_1, &GPIO_InitStruct);// Инициализация пина для датчика C
    HAL_GPIO_InitInterruptLine(GPIO_MUX_PORT1_8_LINE_0,GPIO_INT_MODE_CHANGE);
    HAL_GPIO_InitInterruptLine(GPIO_MUX_PORT1_9_LINE_1,GPIO_INT_MODE_CHANGE);
    HAL_GPIO_InitInterruptLine(GPIO_MUX_PORT1_10_LINE_2,GPIO_INT_MODE_CHANGE);
}
```

Окончание листинга А.3

```
uint8_t get_hall_sensor_state(void)
{
    uint8_t hall_state = 0;
    if (HAL_GPIO_ReadPin(GPIO_1, HALL_A) == GPIO_PIN_HIGH) hall_state |= 0x01;
    if (HAL_GPIO_ReadPin(GPIO_1, HALL_B) == GPIO_PIN_HIGH) hall_state |= 0x02;
    if (HAL_GPIO_ReadPin(GPIO_1, HALL_C) == GPIO_PIN_HIGH) hall_state |= 0x04;
    return hall_state & 0x07;
}

void update_commutation()
{
    uint8_t hall_state = get_hall_sensor_state();  // Читаем состояние холлов
    commutation(hall_state); // Обновляем коммутацию по таблице
}
```

Листинг А.4 – Исходный файл pwm.h

```
#ifndef PWM_H
#define PWM_H

#include "mik32_hal_timer32.h"

#define DEAD_TIME_US 1

#define A_HIGH htimer32_channel0
#define A_LOW  htimer32_channel1
#define B_HIGH htimer32_channel2
#define B_LOW  htimer32_channel3
#define C_HIGH htimer32_channel4
#define C_LOW  htimer32_channel5

extern TIMER32_HandleTypeDef htimer32_1;
extern TIMER32_HandleTypeDef htimer32_2;

extern TIMER32_CHANNEL_HandleTypeDef A_HIGH;
extern TIMER32_CHANNEL_HandleTypeDef A_LOW;
extern TIMER32_CHANNEL_HandleTypeDef B_HIGH;
extern TIMER32_CHANNEL_HandleTypeDef B_LOW;
extern TIMER32_CHANNEL_HandleTypeDef C_HIGH;
extern TIMER32_CHANNEL_HandleTypeDef C_LOW;

typedef struct {
    TIMER32_CHANNEL_HandleTypeDef* high_phase_high;
    TIMER32_CHANNEL_HandleTypeDef* high_phase_low;
    TIMER32_CHANNEL_HandleTypeDef* low_phase_high;
    TIMER32_CHANNEL_HandleTypeDef* low_phase_low;
    TIMER32_CHANNEL_HandleTypeDef* off_phase_high;
    TIMER32_CHANNEL_HandleTypeDef* off_phase_low;
} CommutationStep;
```

Окончание листинга А.4

```
extern const CommutationStep comm_table[6];

extern uint32_t pwm_top;
extern uint32_t pwm_duty_percent;
extern uint32_t pwm_duty_cycle;
extern volatile uint8_t motor_direction;

void pwm_init(void);
void           safe_switch_phase(TIMER32_CHANNEL_HandleTypeDef*          high,
TIMER32_CHANNEL_HandleTypeDef* low, uint16_t high_duty, uint16_t low_duty);
void commutation(uint8_t comm_state);

#endif // PWM_H
```

Продолжение приложения А

Листинг А.5 - Исходный файл pwm.c

```
#include "pwm.h"

TIMER32_HandleTypeDef htimer32_1;
TIMER32_HandleTypeDef htimer32_2;

TIMER32_CHANNEL_HandleTypeDef A_HIGH;
TIMER32_CHANNEL_HandleTypeDef A_LOW;
TIMER32_CHANNEL_HandleTypeDef B_HIGH;
TIMER32_CHANNEL_HandleTypeDef B_LOW;
TIMER32_CHANNEL_HandleTypeDef C_HIGH;
TIMER32_CHANNEL_HandleTypeDef C_LOW;
const CommutationStep comm_table[6] = {
    { &A_HIGH, &A_LOW, &C_HIGH, &C_LOW, &B_HIGH, &B_LOW }, // Step 0
    { &B_HIGH, &B_LOW, &C_HIGH, &C_LOW, &A_HIGH, &A_LOW }, // Step 1
    { &B_HIGH, &B_LOW, &A_HIGH, &A_LOW, &C_HIGH, &C_LOW }, // Step 2
    { &C_HIGH, &C_LOW, &A_HIGH, &A_LOW, &B_HIGH, &B_LOW }, // Step 3
    { &C_HIGH, &C_LOW, &B_HIGH, &B_LOW, &A_HIGH, &A_LOW }, // Step 4
    { &A_HIGH, &A_LOW, &B_HIGH, &B_LOW, &C_HIGH, &C_LOW }, // Step 5
};

uint32_t pwm_top = 7543;
uint32_t pwm_duty_percent;
uint32_t pwm_duty_cycle;

volatile uint8_t motor_direction;

static void Timer32_PWM_Init(void) {
    // === Таймер 32.1 ===
    htimer32_1.Instance = TIMER32_1;
    htimer32_1.Top = pwm_top;
    htimer32_1.State = TIMER32_STATE_DISABLE;
    htimer32_1.Clock.Source = TIMER32_SOURCE_PRESCALER;
    htimer32_1.Clock.Prescaler = 0;
    htimer32_1.InterruptMask = 0;
    htimer32_1.CountMode = TIMER32_COUNTMODE_FORWARD;
    HAL_Timer32_Init(&htimer32_1);

    // === Таймер 32.2 ===
```

Продолжение листинга А.5

```
    htimer32_2.Instance = TIMER32_2;
    htimer32_2.Top = pwm_top;
    htimer32_2.State = TIMER32_STATE_DISABLE;
    htimer32_2.Clock.Source = TIMER32_SOURCE_PRESCALER;
    htimer32_2.Clock.Prescaler = 0;
    htimer32_2.InterruptMask = 0;
    htimer32_2.CountMode = TIMER32_COUNTMODE_FORWARD;
    HAL_Timer32_Init(&htimer32_2);

    // === Каналы таймера 32.1 ===
    TIMER32_CHANNEL_HandleTypeDef* channels_32_1[] = {
        &htimer32_channel0, &htimer32_channel1,
        &htimer32_channel2, &htimer32_channel3
    };
    for (int i = 0; i < 4; i++) {
        channels_32_1[i]->TimerInstance = TIMER32_1;
        channels_32_1[i]->ChannelIndex = i;
        channels_32_1[i]->PWM_Invert = TIMER32_CHANNEL_NON_INVERTED_PWM;
        channels_32_1[i]->Mode = TIMER32_CHANNEL_MODE_PWM;
        channels_32_1[i]->CaptureEdge = TIMER32_CHANNEL_CAPTUREEDGE_RISING;
        channels_32_1[i]->OCR = 0;
        channels_32_1[i]->Noise = TIMER32_CHANNEL_FILTER_ON;
        HAL_Timer32_Channel_Init(channels_32_1[i]);
    }

    // === Каналы таймера 32.2 ===
    htimer32_channel4.TimerInstance = TIMER32_2;
    htimer32_channel4.ChannelIndex = TIMER32_CHANNEL_0;
    htimer32_channel4.PWM_Invert = TIMER32_CHANNEL_NON_INVERTED_PWM;
    htimer32_channel4.Mode = TIMER32_CHANNEL_MODE_PWM;
    htimer32_channel4.CaptureEdge = TIMER32_CHANNEL_CAPTUREEDGE_RISING;
    htimer32_channel4.OCR = 0;
    htimer32_channel4.Noise = TIMER32_CHANNEL_FILTER_ON;
    HAL_Timer32_Channel_Init(&htimer32_channel4);

    htimer32_channel5.TimerInstance = TIMER32_2;
    htimer32_channel5.ChannelIndex = TIMER32_CHANNEL_1;
    htimer32_channel5.PWM_Invert = TIMER32_CHANNEL_NON_INVERTED_PWM;
    htimer32_channel5.Mode = TIMER32_CHANNEL_MODE_PWM;
    htimer32_channel5.CaptureEdge = TIMER32_CHANNEL_CAPTUREEDGE_RISING;
    htimer32_channel5.OCR = 0;
    htimer32_channel5.Noise = TIMER32_CHANNEL_FILTER_ON;
    HAL_Timer32_Channel_Init(&htimer32_channel5);

    // === Включение всех каналов ===
    for (int i = 0; i < 4; i++) {
        HAL_Timer32_Channel_Enable(channels_32_1[i]);
    }
    HAL_Timer32_Channel_Enable(&htimer32_channel4);
    HAL_Timer32_Channel_Enable(&htimer32_channel5);
}

void pwm_init(void) {
    Timer32_PWM_Init();
}
```

Окончание листинга А.5

```
      void           safe_switch_phase(TIMER32_CHANNEL_HandleTypeDef*           high,
TIMER32_CHANNEL_HandleTypeDef* low, uint16_t high_duty, uint16_t low_duty) {
          // Отключаем оба ключа
          HAL_Timer32_Channel_OCR_Set(high, 0);
          HAL_Timer32_Channel_OCR_Set(low, 0);

          // Dead-time пауза
          HAL_DelayUs(DEAD_TIME_US);

          // Включаем нужные ключи
          HAL_Timer32_Channel_OCR_Set(high, high_duty);
          HAL_Timer32_Channel_OCR_Set(low, low_duty);
      }

      void commutation(uint8_t comm_state) {
          if (comm_state < 1 || comm_state > 6)
          {
              // Всё выключить, если неверное состояние
              safe_switch_phase(&A_HIGH, &A_LOW, 0, 0);
              safe_switch_phase(&B_HIGH, &B_LOW, 0, 0);
              safe_switch_phase(&C_HIGH, &C_LOW, 0, 0);
              return;
          }
          // Если реверс, то просто зеркалим индекс
          uint8_t table_index = motor_direction ? (6 - comm_state) : (comm_state - 1);
          const CommutationStep* step = &comm_table[table_index];

          // Верхняя фаза (HIGH ключ работает)
          safe_switch_phase(step->high_phase_high,             step->high_phase_low,
pwm_duty_cycle, 0);

          // Нижняя фаза (LOW ключ работает)
          safe_switch_phase(step->low_phase_high,       step->low_phase_low,       0,
pwm_duty_cycle);
          // Свободная фаза (всё выключено)
          safe_switch_phase(step->off_phase_high, step->off_phase_low, 0, 0);
      }
```

Листинг А.6 - Исходный файл adc.h

```
      #ifndef ADC_H
      #define ADC_H

      #include <stdint.h>
      #include "mik32_hal_adc.h"
      #include "pwm.h"

      #define ADC_OVERCURRENT_THRESHOLD 3368

      extern volatile bool conversion_ready_flag;
      extern ADC_HandleTypeDef hadc;
      extern volatile uint16_t adc_value;

      void ADC_Init(void);
      bool ADC_CheckOvercurrent(void);
      #endif // ADC_H
```

Листинг А.7 - Исходный файл adc.c

```
#include "adc.h"
#include "mik32_hal.h"
#include "mik32_hal_irq.h"

ADC_HandleTypeDef hadc;
volatile uint16_t adc_value = 0;
volatile bool conversion_ready_flag = 1;

void ADC_Init(void)
{
    hadc.Instance = ANALOG_REG;

    hadc.Init.Sel = ADC_CHANNEL5;
    hadc.Init.EXTRef = ADC_EXTREF_OFF;       // Встроенное опорное напряжение
    hadc.Init.EXTClb = ADC_EXTCLB_CLBREF;     // Калибровка внутренним опорным
напряжением
    HAL_ADC_Init(&hadc);
}

bool ADC_CheckOvercurrent(void)
{
    return (adc_value > ADC_OVERCURRENT_THRESHOLD);
}
```

Листинг А.8 - Исходный файл uart.c

```
#ifndef USART_H
#define USART_H

#include "mik32_hal_usart.h"

#define BUF_MAX 10

extern USART_HandleTypeDef husart0;

/* Буферы и флаги */
extern char rx_buf[];
extern volatile uint8_t rx_index;
extern volatile bool rx_complete;
extern volatile bool start_char_received;

void USART_Init(void);

#endif // USART_H
```

Листинг А.9 - Исходный файл uart.c

```
#include "usart.h"
#include "mik32_hal_usart.h"

USART_HandleTypeDef husart0;

char rx_buf[BUF_MAX];
volatile uint8_t rx_index = 0;
volatile bool rx_complete = false;
volatile bool start_char_received = false;
```

Окончание листинга А.9

```
void USART_Init()
{
    husart0.Instance = UART_0;
    husart0.transmitting = Enable;
    husart0.receiving = Enable;
    husart0.frame = Frame_8bit;
    husart0.parity_bit = Disable;
    husart0.parity_bit_inversion = Disable;
    husart0.bit_direction = LSB_First;
    husart0.data_inversion = Disable;
    husart0.tx_inversion = Disable;
    husart0.rx_inversion = Disable;
    husart0.swap = Disable;
    husart0.lbm = Disable;
    husart0.stop_bit = StopBit_1;
    husart0.mode = Asynchronous_Mode;
    husart0.xck_mode = XCK_Mode3;
    husart0.last_byte_clock = Disable;
    husart0.overwrite = Disable;
    husart0.rts_mode = AlwaysEnable_mode;
    husart0.dma_tx_request = Disable;
    husart0.dma_rx_request = Disable;
    husart0.channel_mode = Duplex_Mode;
    husart0.tx_break_mode = Disable;
    husart0.Interrupt.ctsie = Disable;
    husart0.Interrupt.eie = Disable;
    husart0.Interrupt.idleie = Disable;
    husart0.Interrupt.lbdie = Disable;
    husart0.Interrupt.peie = Disable;
    husart0.Interrupt.rxneie = Disable;
    husart0.Interrupt.tcie = Disable;
    husart0.Interrupt.txeie = Disable;
    husart0.Modem.rts = Disable; //out
    husart0.Modem.cts = Disable; //in
    husart0.Modem.dtr = Disable; //out
    husart0.Modem.dcd = Disable; //in
    husart0.Modem.dsr = Disable; //in
    husart0.Modem.ri = Disable;  //in
    husart0.Modem.ddis = Disable;//out
    husart0.baudrate = 9600;
    HAL_USART_Init(&husart0);
}
```