МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет **ФИЗИЧЕСКИЙ**

Кафедра **автоматизации физико-технических исследований**

Магистерская программа **Информационные процессы и системы**

Направление подготовки **03.04.02 ФИЗИКА**

Образовательная программа: **МАГИСТРАТУРА**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(проектно-исследовательский формат)

Набиева Мадина Абдувалиевна

Фамилия, Имя, Отчество автора)

Тема работы **Разработка микроконтроллерного модуля управления тиристорами для распределенной системы управления установкой анодного оксидирования**

«К защите допущена»

И. о. зав. кафедрой

к.т.н., доцент

Лысаков К.Ф./……..
(фамилия И., О.) / (подпись, МП)

«……»……20…г.

Научный руководитель

д.т.н., доцент

Зав. лаб. ИАиЭ СО РАН

Зюбин В.Е./……..
(фамилия И., О.) / (подпись, МП)

«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2024

ОГЛАВЛЕНИЕ

Введение

Благодаря развитию современной электронной техники, уменьшению габаритов, повышению функционального насыщения идеология проектирования крупных систем изменилась. Происходит переход от интегрированных систем к распределенным, в которых каждый элемент системы является активным устройством и все они управляются одним мощным процессором. Распределенные системы обладают такими преимуществами, как легкая расширяемость, высокая надежность, малые сроки разработки, легкость тестирования и отладки, возможность распределения системы по объекту, использование компьютеров и контроллеров меньшей мощности.

Такие системы находят успешное применение в промышленной автоматизации технологических процессов. В данной работе предлагается рассмотреть возможность автоматизации процесса анодного оксидирования.

На сегодняшний день анодное оксидирование алюминия — одно из самых перспективных способов обработки алюминия и его сплавов. Использование оксидных пленок наиболее распространено в таких отраслях промышленности как, строительная, военная, автомобильная, самолето-строение и др., обеспечивая надежную защиту металла от коррозии, увеличивая твердость поверхности и создавая электро- и теплоизоляционный слой. С их использованием изготавливаются износостойкие детали машин, трансформаторы сухого типа, эндопротезы, тактические бронежилеты и многие другие изделия.

Работы на УАО производятся в гальванических цехах предприятий и относятся к классу вредных и опасных, требующих строгого соблюдения техники безопасности труда. Поэтому особенно актуальной становится автоматизация технологического процесса анодного оксидирования, в результате которой существенно повышается производительность и эффективность метода, расширяются функциональные возможности и обеспечивается строгий контроль получаемых оксидных пленок.

В классических подходах, применяемых к вопросу автоматизации данного процесса, используются в основном ПЛК, которые имеют ряд существенных недостатков в сравнении с микропроцессорными устройствами: относительно высокая стоимость, отсутствие масштабирования аппаратной части без замены всего контроллера, проблемы с закупкой, сложности с ремонтом.

Разработка киберфизических систем с использованием микроконтроллерных модулей и программирование таких систем является актуальной задачей в промышленной автоматизации на замену классическим решениям.

Объектом исследования в данной работе является экспериментальная установка анодного оксидирования (УАО).

В Институте Автоматики и Электрометрии СО РАН выполняется проект — «Автоматизация технологического процесса термоэлектрохимического оксидирования». Процесс ведется в ручном режиме, при котором в ходе анодирования предполагается получить пленку уникальных параметров до 400 мкм (при типовой 5-25 мкм). Режимы оксидирования не выявлены и, как следствие, большая экспериментальная составляющая процесса. Во время отработки режимов возможны серьезные изменения в составе программно-аппаратного комплекса, появление дополнительных датчиков, исключение существующих, а также возможны коррекции исполнительных органов. Поэтому вместо классической централизованной архитектуры с одним ПЛК планируется использовать распределенную архитектуру на микроконтроллерах ATmega.

Цель данной магистерской работы: разработка микроконтроллерного модуля управления тиристорами распределенной системы управления установкой анодного оксидирования.

Задачи:

- анализ подходов к разработке с учетом специфики установки анодного оксидирования;
- формирование списка требований;

- разработка алгоритма;
- реализация алгоритма на микроконтроллерном модуле;
- создание имитационного стенда для экспериментальной апробации на микроконтроллерном модуле;
- экспериментальная апробация на имитационном стенде.

Результаты работы описаны в трех главах. В первой главе проводится анализ специфики задачи и существующих подходов к решению, представлены требования к разрабатываемой системе. Во второй главе описана архитектура системы и предлагаемое распределенное решение. В третьей главе рассмотрены вопросы реализации и результаты эмпирических исследований на имитационном стенде.

1. Техническое задание

Данная работа выполняется в рамках проекта Института Автоматики и Электрометрии СО РАН «Автоматизация технологического процесса термоэлектрохимического оксидирования». В рамках этой работы требуется разработать микроконтроллерный модуль управления тиристорами для экспериментальной установки анодного оксидирования. Ниже перечислены пункты технического задания.

1. Изучить процесс анодного оксидирования алюминия и его сплавов и описать особенности его протекания.
2. Провести анализ существующих подходов к решению задачи автоматизации процесса анодного оксидирования.
3. Провести критический анализ специфики инструментальных и аппаратных средств по автоматизации технологического процесса анодного оксидирования и разработать базовую схему распределенной системы управления.

4. Для разработки выделить модуль данного автоматизируемого процесса, а именно модуль управления тиристорными блоками как ключевой модуль в системе управления процессом анодного оксидирования.
5. Составить план проекта и определить список решаемых задач.
6. Сформировать список требований к программно-аппаратному решению.
7. Разработать алгоритм управления и реализовать его на микроконтроллерном модуле.
8. Создать имитационный стенд для экспериментальной апробации микро-контроллерного модуля.
9. Провести экспериментальную апробацию разработанного алгоритма на имитационном стенде и проанализировать полученные результаты.

ГЛАВА 1. АНАЛИЗ СПЕЦИФИКИ ЗАДАЧИ

1.1 Описание технологического процесса анодного оксидирования

В настоящее время развивающиеся новые технологии в промышленности дают возможность появлению новых видов высококачественных изделий, предъявляющих высокие технические требования для металлических и неметаллических неорганических покрытий. Особое значение принадлежит группе конверсионных неметаллических неорганических покрытий, формирующихся в результате конверсии при взаимодействии металла с раствором электролита так, что структура полученного покрытия включает в себя ионы металла.

Основу таких покрытий составляют оксидные или солевые пленки, которые образуются на поверхности металла в результате электрохимической обработки. Наибольшее распространение получили оксидные покрытия алюминия и его сплавов, так как благодаря им металл приобретает наилучшие механические, диэлектрические, физико-химические свойства [1].

Известно, что естественная тонкая оксидная пленка Al_2O_3 или

$Al_2 O_{3*} nH_2 O$ всегда присутствует на поверхности алюминия и его сплавов и имеет толщину 0,02-0,04 мкм. Однако, такая пленка не предоставляет защиту металлу от коррозии в загрязненной хлоридами атмосфере. Чтобы защитить изделие из металла более толстым оксидным слоем, его очищают от разного рода загрязнений химическими воздействиями и подвергают реакциям анодного и химического оксидирования.

При анодной поляризации алюминия на его поверхности протекают сложные физико-химические явления, в результате которых образуется тонкий барьерный слой, растущий при взаимодействии ионов алюминия с ионами кислорода, когда они движутся в направлении друг к другу в барьерном слое. Процесс образования оксидной пленки изображен на рис. 1.

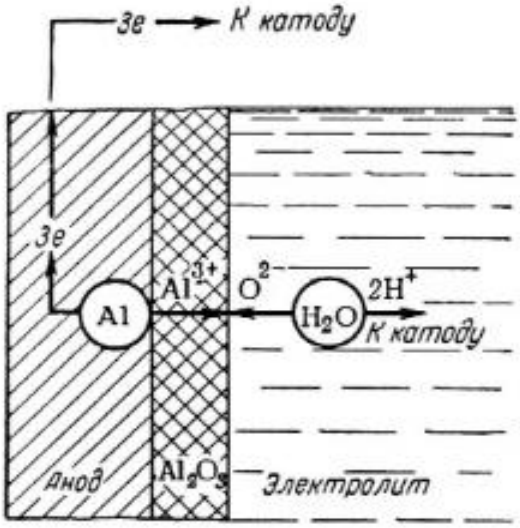

Рисунок 1. Схема анодного процесса ионизации алюминия.

В представленной ниже реакции показан в общем виде процесс анодного окисления:

$$2Al + 3H_2O \rightarrow Al_2O_3 + 6H^+ + 6\bar{e}$$

Далее поры металла (d = 0,05-0,1 мкм) наполняются электролитом, формируя новый барьерный слой. В результате окисная пленка увеличивается в толщине, благодаря образованию пористого слоя и проникая далее вглубь

металла. Схема формирования оксидной пленке на алюминии представлена на рисунке 2.

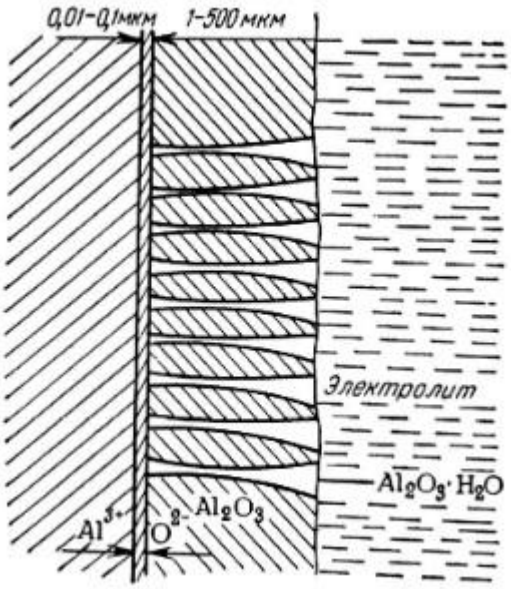

Рисунок 2. Схема формирования оксидной пленки на алюминии.

В процессе электрохимической обработки алюминия и его сплавов одновременно протекают две противоположные реакции: реакция электрохимического окисления металла и реакция химического растворения оксидного слоя.

Параметры оксидного покрытия (свойства, структура, толщина) зависят от отношения скоростей протекания данных химических реакций. В случае отсутствия химического растворения образующегося оксида формируется тонкая, беспористая пленка барьерного типа. При равенстве скоростей реакций на поверхности металла непрерывно возникает и сразу растворяется тонкая пассивирующаяся пленка. Если же скорость электрохимического процесса заметно превосходит скорость химического растворения пленки, образуется значительная толщина оксидных покрытий. Данные покрытия обладают высокими антикоррозионными свойствами и способствуют защите металла от внешнего воздействия [1].

Технологический процесс анодного оксидирования состоит из семи последовательно выполняемых операций:

Подготовительные операции

- *Химическое обезжиривание*, на поверхности металла растворяются все жиры минерального, животного и растительного происхождения;
- *Травление,* с поверхности покрываемых изделий удаляются все окислы;
- *Осветление,* удаляется с поверхности травильный шлам, который содержит алюминаты.
- *Операция анодирования.*

Состав ванны: H2SO4 180-200 г/л; Вода, деминерализованная с электропроводностью 400 мкСм/см2; Т = 19 ± 6 С. Анодирование постоянным током обычно проводят при плотности тока 1,5 А/дм2 и температуре 20 °С. Раствор постоянно перемешивается, чтобы поддерживать необходимую температуру и охлаждается специально установленными холодильными устройствами. Для получения пленки заданной толщины выбирают время анодирования исходя из расчета скорости роста 0,20 мкм на 1 А-мин/дм2.

Заключительные операции

- *Пассивация пленки в растворе хромпика* выполняется для увеличения коррозионной стойкости, уменьшения пористости и маслоемкости;
- *Промывка в холодной (горячей) воде* смывает химические вещества и продукты реакций в целях препятствия попадания их в следующую ванну;
- *Сушка.*

Является заключительной операцией технологии анодирования.

Рассмотрим параметрическую модель технологического процесса анодного оксидирования, представленную на рис.3.

Рисунок 3. Параметрическая модель технологического процесса анодного оксидирования.

Структура оксидной пленки зависит от соотношения следующих входных и выходных параметров:

Входные параметры

- мощность (P),
- концентрация компонентов электролита (C),
- время протекания технологического процесса (τ),
- площадь обрабатываемых изделий (S),
- уровень электролита (L),
- качество предварительной обработки металла (H).

Управляемые выходные параметры — температура (T), уровень электролита (L), кислотность электролита (pH), длительность процесса(τ), плотность тока (j) [2].

Кроме этого на процесс оказывают влияние внешние возмущения, такие как концентрация посторонних ионов в электролите (Z), качество поверхности обрабатываемого металла (K), вынос электролита в процессе промывки (Y), испарение электролита (I).

Рассмотрим следующие измеряемые параметры, оказывающие влияние на механические свойства анодных покрытий:

Температура электролита.

Для формирования качественного покрытия требуется строгое соблюдение предельного температурного режима и предотвращение перегрева электролита. Повышении температуры влечет за собой возрастание выхода по току, что приводит к образованию рыхлой сползающей пленки, повышению стоимости работы и теплопотерям ванны [2].

Длительность технологического процесса.

Длительность процесса зависит от скорости осаждения металла и оказывает значительное влияние на качество оксидных покрытий. Поэтому этот параметр должен строго соответствовать требованиям, описанным в технологии. Для каждой температуры в технологии заложена строго определенная длительность процесса анодного оксидирования. Если в течении процесса это соответствие нарушается, то изменяются структурные характеристики оксидной пленки. С увеличением длительности по времени по сравнению с оптимальной, пленка начинает разрыхляться и её защитные свойства понижаются [2].

Уровень электролита.

Испарение электролита в окружающую среду, а также потери раствора с поверхностей деталей при их извлечении из ванны изменяют уровень электролита. При понижении уровня детали не полностью погружаются в ванну, тем самым ухудшается качество их обработки. Кроме того, изменение уровня электролита оказывает влияние на электрический режим ванны изменяя величину сопротивления [3].

pH раствора.

Для нормального протекания процесса образования оксидной пленки необходимо контролировать состав электролита. Контроль производится регулированием уровня кислотности и концентрации электролита, а также стабилизацей величины pH электролита. Величина pH влияет на механические характеристики покрытия: низкие значения pH уменьшают твердость покрытия, сокращают выход по току, увеличивают пластичность, а высокие значения наоборот [4].

Толщина электрохимических покрытий.

Толщина электрохимических покрытий (δ) согласно закону Фарадея зависит от плотности тока (i_k) и продолжительности электролиза (τ) и может быть вычислена с учетом выхода по току и электрохимического эквивалента (Э) по формуле:

$$\delta = \frac{i_к \cdot \tau \cdot Э}{p}$$

Где p – плотность осаждаемого металла [2].

Микропроцессорные устройства позволяют автоматически контролировать сохранение оптимального соответствия параметров заданной технологии, что обеспечивает бесперебойный режим работы оборудования, экономию энергозатрат. Кроме того, в непредвиденной ситуации оператор получает возможность скорректировать ход процесса.

1.2 Анализ существующих подходов к решению.

В результате анализа задачи можно предложить следующий подход к автоматизации технологического процесса:

1) Внешним элементом системы управления, контролируемой средой, является установка АО, включающая в себя аппаратные средства и оборудование, в котором происходят физические процессы

2) С этой средой посредством датчиков и исполнительных механизмов связывается микроконтроллер

3) Датчики собирают данные из окружающей среды и передают их в систему управления

4) Контроллер реагирует на входные данные и передает их в систему управления для исполнительных механизмов

5) Исполнительные механизмы изменяют ход физического процесса.

В настоящее время существует множество различных подходов к автоматизации технологического процесса практически любого уровня сложности. Для реализации поставленной цели, рассмотрим применимость распределенных CAN систем, как способ автоматизации процесса анодного оксидирования.

Распределенная система состоит из некоторого числа компьютеров, которые не зависят друг от друга, не используют совместную память и обмениваются информацией между собой с помощью коммуникационной сети через передачу сообщений. При этом, каждый компьютер обладает своими собственными оперативной памятью и операционной системой. Однако эти операционные системы совместно решают общую задачу, предоставляя друг другу свои службы [4].

Применение микропроцессорных устройств в системах с распределенной архитектурой предоставляет возможность снизить стоимость как самой системы, так и ее обслуживания в целом.

Метод распределенного управления впервые был применен в середине 80-х годов в автомобильной промышленности, когда был разработан ряд протоколов подключения для обеспечения сетевого взаимодействия внутри автомобиля. Затем применение распределенных систем стало активно исследоваться и применяться во многих других областях науки и промышленности, появилось множество научных статей с разработками новых подходов к этой проблеме [5].

Так был рассмотрен алгоритм «управление по времени», используемый сегодня в промышленности, который может быть сформирован следующим образом:

(1) контроллеры передают друг другу периодические сообщения о состоянии ключевых системных переменных (закодированная информация, полученная от тесно связанных датчиков, и производная информация на основе обработки этих данных);

(2) более важная информация передается с более короткими интервалами;

(3) периоды рассчитываются таким образом, чтобы при одновременной передаче не возникало большого числа коллизий;

(4) сеть передачи данных по существу распределена на независимые подсети, которые не создают помехи друг другу при передаче разнородной информации (как в примере, блоку управления двигателем не обязательно знать о радиостанции, играющей в данный момент) [10].

В статьях [8] и [9] сравниваются приложения подходов с запуском по времени и с запуском событий в автомобильных распределенных системах управления с использованием шины CAN и ее варианта TTCAN (time-triggered CAN). Отмечается, что работа подсистем, рассчитываемых по временным окнам, контролирует соблюдение требований о том, чтобы не нарушалось максимальное время работы. Однако такие системы не могут должным образом реагировать на важные сообщения о критическом состоянии системы, поскольку последние могут быть обработаны только после арбитража шины. С другой стороны, полностью управляемые событиями системы могут обрабатывать сообщения с желаемыми приоритетами; однако из-за недетерминизма при замене компонентов системы на другие (например, быстрее или медленнее) алгоритм управления должен быть полностью пересмотрен. Делается вывод, что необходимо использовать подход, инициируемый событиями, только для нескольких наиболее важных сообщений и включать их обработку в специальное временное окно раньше всех остальных, в то время как обработка обычных сообщений должна выполняться с использованием управления, инициируемого временем. В этом случае можно рассчитать изменение закона управления объектом, и оно становится предсказуемым. Таким образом, при разработке алгоритмов управления важно учитывать и моделировать задержки, связанные с шиной.

Модульно-ориентированный (device-centric) подход [8], в котором каждый контроллер системы управления программируется отдельно на

практике заменил существующие подходы. Основными недостатками этого подхода являются сложность обслуживания программного обеспечения, его читаемость и модификация, гибкость и сложность верификации, что имеет решающее значение для многих распределенных систем. Поэтому появилась потребность разработки топологически независимого (ориентированного на приложения) программирования киберфизических систем управления, неоспоримым преимуществом которого является возможность простого и последовательного описания алгоритма управления, независимого от типа узлов, их количества и топологии распределенной системы управления. В работе [10] была представлена общая схема развертывания процесс-ориентированной спецификации на произвольной распределенной архитектуре и предложен алгоритм распределения компонентов по кластерам, работающим на одном контроллере, чтобы не приходилось заботиться о синхронизации внутри кластера. Процессно-ориентированная парадигма предполагает, что управляющая программа, заданная как набор слабо связанных параллельных процессов, структурно и функционально соответствует технологическому описанию установки. Каждый процесс задается последовательным набором состояний. Состояния задаются набором арифметических конструкций, дополняемых операцией тайм-аута, операцией установки СОСТОЯНИЯ и операциями ЗАПУСКА / ОСТАНОВКИ / проверки состояния для связи с другими процессами [7].

1.3 Требования к техническому обеспечению.

В системе автоматизации процесса анодного оксидирования предусмотрены следующие функции:

1) автоматический сбор и контроль всех необходимых технологических параметров;

2) автоматическая защита оборудования в случае возникновении аварийных ситуаций;

3) удаленное управление электроприводной аппаратурой и автоматическими устройствами;

4) мониторинг и идентификация отказов оборудования при его работе;

5) сохранение на сервере баз данных основных контролируемых технологических параметров;

6) контроль уровня электролита и растворов;

7) сигнализация в случае достижения предельных значений контролируемых параметров (температуры, уровня электролита, длительности процесса);

8) отображение информация о работе установки на экране оператора.

Для реализации вышеперечисленных требований комплекс технических средств должен включать следующие устройства:

- датчики, контроллеры, исполнительные механизмы;
- средства удаленного управления,
- программно-технические средства обработки, хранения и передачи информации,
- средства отображения и регистрации информации (вторичные приборы, видеомониторы).

Датчики, используемые в системе, контактируют с агрессивной средой электролитов, поэтому они должны быть изготовлены из коррозионностойких материалов во взрывобезопасном варианте.

Архитектура микроконтроллеров, применяемых в данной системе должна иметь модульную структуру, благодаря чему появляется возможност легко модифицировать систему, исключая существующие компоненты или добавляя новые при необходимости. Это особенно актуально, если проект реализуется в качестве экспериментального для выявление режимов анодирования [20].

1.4 Требования к программному обеспечению.

Программное приложение (ПО) должно удовлетворять следующим критериям:

- модульность построения всех составляющих;
- иерархичность собственно ПО и данных;
- распределенная архитектура системы управления;
- CAN bus;
- защита от копирования кода;
- возможность передачи массивов данных до 1 Кбайта;
- интеграция с компьютером оператора по USB интерфейсу;
- возможность сохранения параметров в энергонезависимой памяти.

К программному обеспечению запрещается несанкционированный доступ.

Для этого предлагаются следующие средства:

- обязательное использование паролей для выполнения функций обновления ПО;
- разграничения доступа к функциям обновления ПО;
- контроль базового ПО на сохранение целостности данных.

Регулярное сохранение исходных пользовательских программ на электронных носителях и, в случае возникновения необходимости, загрузка через интерфейсные каналы в память контроллеров и в устройства верхнего уровня.

Если в результате эксплуатации задачи возникает потребность в изменении локальных программ, то должна существовать возможность их изменения не затрагивая остальные программы.

ГЛАВА 2. АРХИТЕКТУРА СИСТЕМЫ И ПРЕДЛАГАЕМОЕ РАСПРЕДЕЛЕННОЕ РЕШЕНИЕ.

2.1 Распределенная топология.

Исходя из вышесказанного предполагается, что автоматизация процесса анодного оксидирования с помощью распределенных CAN систем и процесс-ориентированного программирования может быть приемлемым решением для улучшения эффективности процесса и его точности.

В качестве базовой схемы процесса анодного оксидирования рассматривается только операция оксидирования, которая разделяется на шесть блоков в соответствии с их функциональностью по описанию техпроцесса. Схема представлена на рис. 4.

Для реализации предложена распределенная система управления, включающая персональный компьютер оператора и микроконтроллерную систему на основе CAN bus.

Рисунок 4. Базовая схема системы анодного оксидирования.

На основании базовой схемы создаются 12 микроконтроллерных модулей в соответствии с технологическим процессом системы: коммуникационный модуль, модуль управления пусковым режимом, модуль

управления технологическим режимом, модуль управления вытяжной вентиляцией, модуль управления подсистемой гомогенизации электролита, модуль управления подсистемой охлаждения электролита, и шесть измерительных модулей, которые обеспечивают измерение электрических параметров с частотой сэмплирования 10 КГц.

Полученная распределенная система имеет модульную архитектуру, преимуществом которой является возможность компоновки различными блоками расширения. Изменение или модификация определенного модуля не требует перестройки всей платформы, а ограничивается только узлом, связанным со своим модулем. Схема представлена на рис.5.

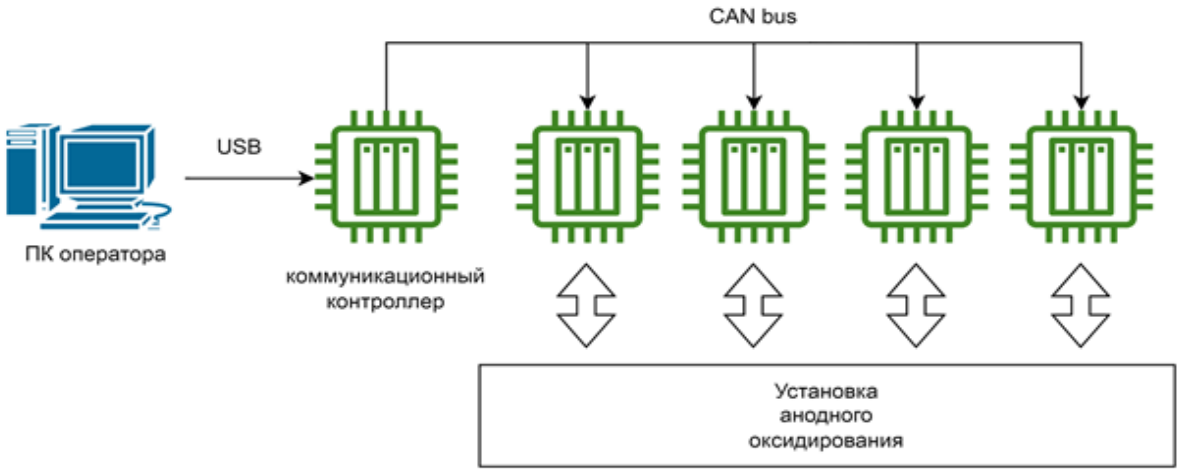

Рисунок 5. Схема распределенной системы управления анодным оксидированием.

Микроконтроллерный модуль управления тиристорами (пусковым и шунтирующими) в базовой схеме распределенной системы управления отвечает за подсистему плавного пуска и подсистему регулирования тока. Он выполняет управление пусковым режимом для обеспечения плавного пуска системы, блокировку запуска при пониженном или повышенном напряжении

сети и защиты от других нештатных ситуаций, а также производит регулирование тока в соответствии с требуемым для технологического процесса значением. Интеграция микроконтроллеров в систему осуществляется посредством шины CAN bus [6].

2.2 Интерфейс CAN.

Интерфейс CAN (Controller Area Network, локальная сеть контроллеров), относится к протоколам высокого уровня и используется для создания последовательных, высококачественных каналов связи в распределённых системах управления. Протокол CAN оптимизирован для систем, в которых должно передаваться относительно небольшое количество информации к любому или всем узлам сети [18].

Протокол CAN (Controller Area Network) выполняет следующие функции:

1. Обеспечение высокоскоростной передачи данных: CAN предоставляет возможность передачи данных на очень высоких скоростях (до 1 Мбит/с). Это позволяет передавать большое количество информации в реальном времени.

2. Управление приоритетами сообщений: CAN использует механизм приоритетов, позволяющий определять важность и срочность сообщений. Это позволяет системе эффективно передавать данные и реагировать на события с разной степенью важности.

3. Детектирование ошибок передачи данных: CAN обеспечивает надежную передачу данных путем обнаружения и исправления ошибок, которые могут возникнуть при передаче информации по среде передачи.

4. Многопользовательский доступ к среде передачи: CAN позволяет нескольким устройствам одновременно передавать и принимать данные по

общей шине, что обеспечивает эффективное использование системных ресурсов.

5. Гибкость в настройке сети: CAN позволяет легко настраивать и программировать сеть, включая добавление или удаление устройств, изменение приоритетов сообщений и другие параметры.

6. Реакция на приоритетные события: CAN позволяет системе быстро реагировать на приоритетные события и передавать соответствующую информацию другим устройствам в сети.

Преимущества CAN-системы выражаются следующими свойствами:

- работа пользователей в режиме реального времени;
- простота реализации и минимальные затраты на использование;
- высокая устойчивость к помехам;
- арбитраж доступа к сети без потери пропускной способности;
- надёжный контроль ошибок передачи и приёма;
- широкий диапазон скоростей работы.

Однако, существуют и недостатки протокола CAN:

- длина сети и скорость передачи зависят друг от друга обратно пропорционально;
- полезные данные занимают в пакете меньший объем по сравнению с служебной информацией;
- отсутствие единого общепринятого стандарта на протокол высокого уровня.

Стандарт CAN компании Bosch не устанавливает уровень физической передачи информации, поэтому соединение между узлами обычно производится путем использования двухпроводной дифференциальной линии (витой пары). Длина линии зависит от скорости передачи и измеряется от 40 м до 5000 м (чем больше скорость, тем меньше длина) [19].

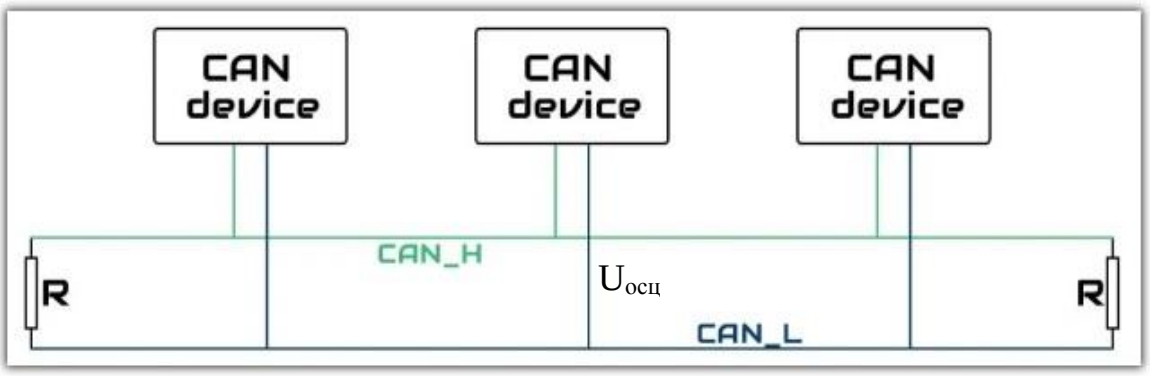

Рисунок 6. Шина распределенной CAN-системы.

Для обеспечения высокой работоспособности шины на концах витой пары должны находиться два согласующих резистора с сопротивлением 120 Ом.

Данные передаются в последовательном режиме, при котором сообщения формируют кадры определенного вида. [12]

Протокол CAN определяет 4 вида кадров:

- Кадр данных (data frame)
- Кадр удаленного запроса (remote frame)
- Кадр перегрузки (overload frame)
- Кадр ошибки (error frame)

Базовый формат кадра данных представлен в табл.1.

Таблица 1. Формат кадра данных

Поле	Длина, бит	Описание
Начало кадра (SOF-Start Off Frame)	1	Сигнализирует начало передачи кадра. Доминантный бит
Идентификатор (ID-Identifier)	11	Уникальный идентификатор
Запрос на передачу (RTR,	1	Должен быть доминантным

Remote Transmission Request)		
Бит расширения идентификатора (IDE-Identifier Extension)	1	Должен быть доминантным
Нулевой резервный бит (RB0-Reserved Bit Zero)	1	Резервный
Длина данных (DLC-Data Length Code)	4	Длина поля данных в байтах (0–8)
Поле данных, байт	0 - 8	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC-Cyclic Redundancy Code)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Бит подтверждения (ACK)	1	Передатчик передаёт рецессивный бит, а приёмник вставляет доминантный бит
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

Расширенный формат кадра данных представлен в табл.2.

Таблица 2. Расширенный формат кадра данных

Поле	Длина, бит	Описание
Начало кадра (SOF-Start Off Frame)	1	Сигнализирует начало передачи кадра. Доминантный бит
Идентификатор A	11	Первая часть идентификатора
Подмена запроса на передачу (SRR)	1	Должен быть рецессивным
Бит расширения идентификатора (IDE-Identifier Extension)	1	Должен быть рецессивным
Идентификатор B	18	Вторая часть идентификатора
Запрос на передачу (RTR)	1	Должен быть доминантным
Резервные биты (RB1 и RB0)	2	Резерв
Длина данных (DLC)	4	Длина поля данных в байтах (0 - 8)
Поле данных, байт	0 - 8	Передаваемые данные (длина в поле DLC)
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра
Разграничитель контрольной суммы	1	Должен быть рецессивным
Промежуток подтверждения (ACK)	1	Передатчик передаёт рецессивный бит, а приёмник

		вставляет доминантный бит
Разграничитель подтверждения	1	Должен быть рецессивным
Конец кадра (EOF)	7	Должен быть рецессивным

В процессе передачи кадр данных и кадр удаленного запроса отделены других кадров специальным межкадровым промежутком (паузой). Фреймы ошибки и перегрузки передаются без пауз для сокращения времени передачи узлами сети [16].

Важную роль в работе протокола CAN играет протокол ошибок, для исполнения которого предусмотрено несколько механизмов таких как контроль передачи битов, использование дополнительных битов (stuffing bit) для кодирования всех полей фреймов данных и запроса, стандартная процедура проверки контрольной суммы, а также выполняется контроль битов фрейма, которые должны иметь заранее определенное значение [12].

Таким образом минимизируется вероятность появления ошибки. Кроме того, узел, который обнаружил ошибку в сообщении, отправляет соответствующее сообщение об этом в сеть CAN при помощи фрейма ошибки. Всем участникам коммуникации одновременно приходит уведомление об ошибке и затем происходит повторная передача сообщения, в котором была обнаружена ошибка.

Существуют определенные правила обслуживания счетчиков ошибок приема, которые обеспечивают надежность, безопасность и удобство в использовании шины CAN для обмена сообщениями в распределенной системе.

2.3 Процесс-ориентированное программирование. Язык Рефлекс.

Методы процесс-ориентированного программирования доказали, что они являются эффективным инструментом для описания распределённых алгоритмов систем управления в области промышленной автоматизации на базе ПЛК. Распределенные системы имеют следующие отличительные особенности: присутствие объекта управления (открытость), непрерывное взаимодействие объекта управления с управляющим алгоритмом, зависящее от событий, происходящих на объекте (цикличность), синхронизация исполнения управляющего алгоритма с физическими процессами на объекте управления (синхронизм), а также логический параллелизм – протекание на объекте управления множества независимых процессов [13].

Процессно-ориентированный подход позволяет описывать алгоритм как будто он исполняется на одном микроконтроллере, а потом автоматически разбивает и разворачивает этот алгоритм на распределённой микроконтроллерной системе. Этот новый концептуальный подход имеет отличие от существующих в том, что описывает управляющий алгоритм на базе модели гипер-процесса и обеспечивает, тем самым, иерархическую структуризацию программы в виде параллельно исполняемых слабосвязанных процессов [14].

Чтобы описать алгоритмы с подобными особенностями, в ИАиЭ СО РАН разработали новый формальный язык программирования четвертого поколения - Си-подобный процесс-ориентированный язык Рефлекс. Данный язык предполагается использовать для спецификации алгоритмов функционирования сложных объектов автоматизации. Он рассчитан на программирование сложных процессов в промышленной автоматизации и робототехнике: для систем, предполагающих активное взаимодействие с внешней средой, технологическим оборудованием, физическими процессами через датчики и органы управления.

В основе концепции языка лежит модель гипер-автомата, описанная в статье [15].

Гипер-автомат задается совокупностью процессов, которые активизируются с заданной периодичностью.

Процесс состоит из набора своих состояний-функций и текущего состояния. События и реакция на эти события задают состояние. Активизация процесса заключается в выполнении его текущего состояния. Смена текущего состояния – одна из возможных реакций на события. Процесс строится как расширение классического конечного автомата (С-автомата), дополненное объективной необходимостью ввести в модель автомата свойства цикличности, синхронности, событийности; предусмотреть совместное функционирование процессов, их взаимодействие и адаптировать терминологию к современному уровню программирования [15].

Программа на языке Reflex представляет собой упорядоченное множество процессов, являющихся конечными автоматами с некоторым набором состояний и множеством правил перехода между этими состояниями. В программе предполагается циклический запуск множества процессов с заданным периодом активизации

.

«Во время работы процессы меняют своё текущее состояние в соответствии со внешними событиями на объекте управления, либо в соответствии со временными событиями (тайм-аутами). Язык предоставляет средства для организации взаимодействия процессов: операторы запуска и остановки процессов, переменные, разделяемые между процессами. В языке Reflex существует спецификация для удобного описания входных и управляющих сигналов, которые привязаны к физическим портам управляющего устройства (ПЛК), а также предоставляются средства для чтения и изменения значений управляющих сигналов (переменные, отображаемые на входные и выходные порты)» [17].

Язык обладает такими свойствами, что алгоритм допускается проектировать в терминологии технологического процесса по методике «сверху-вниз» так, как разрабатывается конструкторская документация на технологический процесс. Создание программы на языке «Рефлекс», по сути, просто отражает алгоритм, с необходимостью уже разработанный при создании объекта управления” [11].

2.4 Описание установки АО

Экспериментальная установка анодного оксидирования представлена на рис. 7.

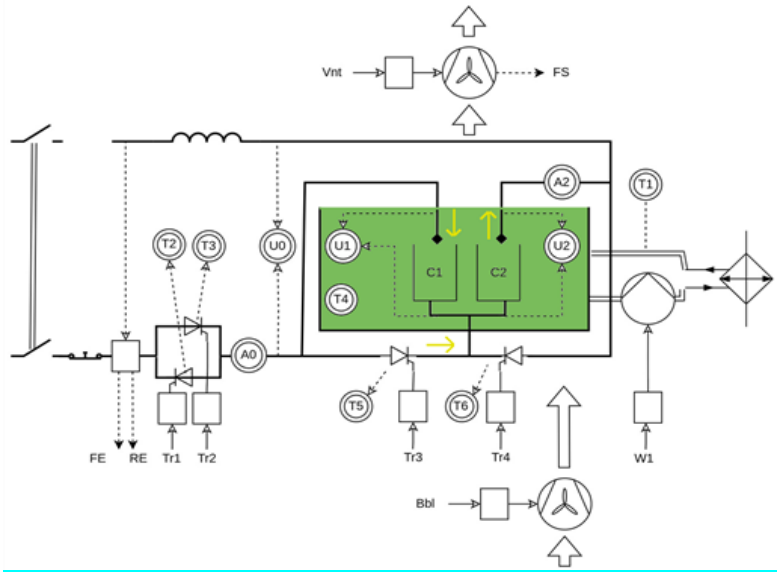

Рисунок 7. Экспериментальная установка анодного оксидирования.

Параметры установки представлены на рис. 8.

C1, C2 -- ванна 1/2
T4 -- температура электролита в ванне
T1 -- температура охлажденного электролита
T2, T3 -- температура пусковых тиристоров
T5, T6 -- температура рабочих тиристоров
A0 -- ток на входе установки
A1 -- ток через ванну C1
A2 -- ток через ванну C2
U0 -- напряжение на входе установки
U1 -- напряжение на ванне C1
U2 -- напряжение на ванне C2
RE -- 0 напряжения питания, положительный фронт
FE -- 0 напряжения питания, отрицательный фронт

i/o Table	
Тип	Имя/параметры
DI	RE, FE, FS
DO	Tr1, Tr2, Tr3, Tr4, Vnt, Bbl
AI	A1, A2, A0 (0-1000A, 10bit, f = 10KSPS) U1, U2, U0 (0-1000D, 10bit, f = 10KSPS) T1, T2, T3, T4, T5, T6 (0-100C, б=1%, 10bit, f = 1SPS)
AO	W1 (0-5V, ШИМ, Tq = 1Hz)

$U_{осц}$

Рисунок 8. Параметры экспериментальной установки АО.

Данная установка работает от сети переменного тока с максимальной амплитудой силы тока 500 А и максимальной амплитудой напряжения 640 В. Установка АО включает сглаживающий дроссель, который защищает остальные компоненты от непредвиденных скачков напряжения в сети.

В настоящее время наиболее распространено анодирование при постоянном токе. В разрабатываемой экспериментальной установке анодного оксидирования процесс происходит под действием переменного тока. При этом все подготовительные и заключительные операции остаются неизменными. Преимущество переменного тока заключается в том, что обрабатываются одновременно две детали. В случае, если необходимо обработать одну деталь, то вместо второго электрода используется лист алюминия или просто болванка. Следует отметить, что одинаковой плотности тока можно добиться как при постоянном токе, так и при переменном токе при напряжении 10-12 В. При этом длительность процесса анодирования занимает 25-30 мин.

Для реализации процесса анодирования в установке содержится емкость для электролита. Внутри емкости расположены ванны С1 и С2, в которых находится рабочая область процесса АО. Их поверхности выступают в качестве электрических контактов. Каждая из них соединена со свинцовой проводящей пластиной.

В процессе работы установки переменный ток создает разность потенциалов между парами ванна-алюминиевое изделие, вследствие чего возможно протекание реакции. Благодаря схеме из тиристоров, регулирующих ток, направление реакции анодирования меняется на противоположное с каждым полупериодом источника тока. Кроме того, направление реакции АО противоположно в каждой из ванн. Таким образом, в положительный полупериод входного сигнала на одном изделии происходит

анодный процесс, а на втором - катодный, а в отрицательный полупериод - наоборот.

Для гомогенизации среды и создания пузырьков в электролите используется барботер.

Для контроля температуры раствора предусмотрен насос системы охлаждения установки. Процесс анодного оксидирования алюминия должен происходить с соблюдением значений входных параметров, представленных в табл. 3.

Таблица 3. Значения входных параметров технологического процесса АО.

Плотность тока	0.9 - 1.5 А/дм^2
Напряжение	14 - 24 В
Температура	15 - 30^0 С
Перемешивание	Постоянное
Время	15 - 60 мин
Время стекания	10 сек

Детали загружают в ванну при небольшом напряжении 12-15 В. После погружения деталей в ванну, через 30-40 секунд напряжение повышают до 18-22 В, при рабочей плотности тока 0.9-1.4 А/дм2. Процесс анодного оксидирования проводят в течении 20-30 минут, поддерживая температуру электролита в пределах 15-30° С. Контроль температуры электролита должен проводиться вблизи от поверхности анодируемых деталей. Если в ходе процесса анодного оксидировании возникает необходимость последующего окрашивания, то время процесса увеличивают до 40-60 мин.

ГЛАВА 3. РЕАЛИЗАЦИЯ И РЕЗУЛЬТАТЫ ЭМПИРИЧЕСКИХ ИССЛЕДОВАНИЙ

3.1. Описание используемых технических средств.

Для разработки микроконтроллерного модуля управления тиристорами в установке анодного оксидирования предполагается использовать микроконтроллеры Arduino Nano семейства ATmega и контроллер шины CAN MCP2515, размещаемых в пылевлагозащищенном корпусе IP 67.

Микроконтроллер характеризуется:

- прогрессивной архитектурой разработки и высокой производительностью;
- оптимизацией архитектуры для разработки на языке Си и его диалектах;
- широкими интерфейсными возможностями (Flash, SRAM, EEPROM; поддержка протоколов SPI, USART, I2C; АЦП , таймеры)
- гибкостью при реконфигурации входов/выходов микроконтроллеров;
- открытой архитектурой представленных на рынке платформ для прототипирования (Arduino и клоны Arduino), в том числе, для Arduino доступны модули расширения для промышленной сети CAN;
- наличием на рынке широкого набора модулей расширения для плат прототипирования;
- наличием большого числа открытых инструментальных средств;
- высоким качеством документации и широким спектром литературы, ориентированной на разработчиков;
- низкой стоимостью.

Микроконтроллер Arduino имеет непосредственное подключение к ПК через USB, вследствие чего облегчается процесс разработки и отладки программ. Кроме того, благодаря наличию кнопки reset, представляется возможность не прибегать в случае зависания к отключению питания или разборке корпуса.

Рисунок 9. Микроконтроллер Arduino Nano-V3-ch340g

Микроконтроллер **MCP2515** — специализированный, выделенный контроллер сети Controller Area Network (**CAN**), предназначен для подключения приложений к CAN-шине. Данный микроконтроллер позволяет осуществлять прием и передачу как стандартных, так и расширенных фреймов данных, а также и фреймы remote. Для понижения нагрузки на управляющий микроконтроллер в MCP2515 предусмотрены 2 маски разрешения (acceptance mask) и 6 фильтров (acceptance filter), применяемых для отбрасывания нежелательных сообщений. Подключение модуля MCP2515 к микроконт-роллеру производится с помощью интерфейса SPI, поэтому его легко подключить ко всем микроконтроллерам с данным интерфейсом.

Рисунок 10. Контроллер MCP2515 шины CAN.

В результате проведённой работы был спроектирован универсальный микроконтроллерный модуль на базе серийно изготавливаемых компонентов Arduino Nano и контроллера шины CAN MCP2515, размещаемых в пылевлагозащищенном корпусе IP 67. Модуль запитывается от напряжения 24В или USB и предусматривает возможность работы с входными-выходными дискретными сигналами (до 13 штук) и аналоговыми сигналами (до 6 каналов). Также для модуля разработано системное программное обеспечение удаленной загрузки пользовательских программ по шине CAN. Разработанный модуль позволяет изменять конфигурацию устройств и датчиков и обеспечивает возможность программирования в процесс-ориентированной парадигме на языке Reflex.

Следующим компонентом блока управления являются тиристоры.

Тиристоры — это полупроводниковые приборы, которые имеют два устойчивых состояния: закрытое состояние (низкой проводимости) и открытое состояние (высокой проводимости).

Рисунок 11. Тиристор КУ202Н

Тиристоры выполняют следующие функции:

1) Регуляция электрического тока, что существенно важно для обеспечения стабильного функционирования электрических сетей и оборудования.

2) Управление напряжением, что позволяет улучшить эффективность работы и снизить энергопотери.

3) Контроль перепада напряжения в системе, чтобы предотвратить его скачки и обеспечить стабильное питание оборудования.

4) Коррекция фазы электрического тока, что позволяет более эффективно использовать энергию.

Благодаря тиристорам появляется возможность осуществлять плавный пуск.

Тиристоры управляются импульсами, которые подаются на управляющий электрод с определенной частотой. Чем больше задержка включения тиристора, тем меньшая мощность поступает в нагрузку.

Регулирование мощности, которая подводится к нагрузке, производится по схеме, показанной на рис. 12.

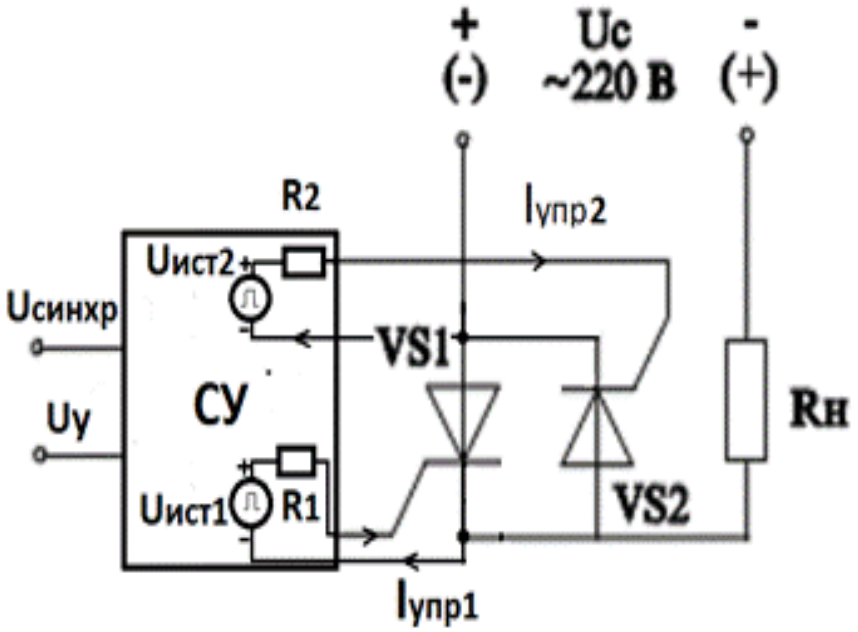

Рисунок 12. Тиристорный регулятор напряжения.

За счет задержки включения тиристора на полупериоде переменного тока регулируется мгновенное и действующее напряжение. В момент, когда ток нагрузки достигает нулевого значения происходит отключение соответствующего тиристора, который работал. Диаграмма работы такого регулятора представлена на рис. 13

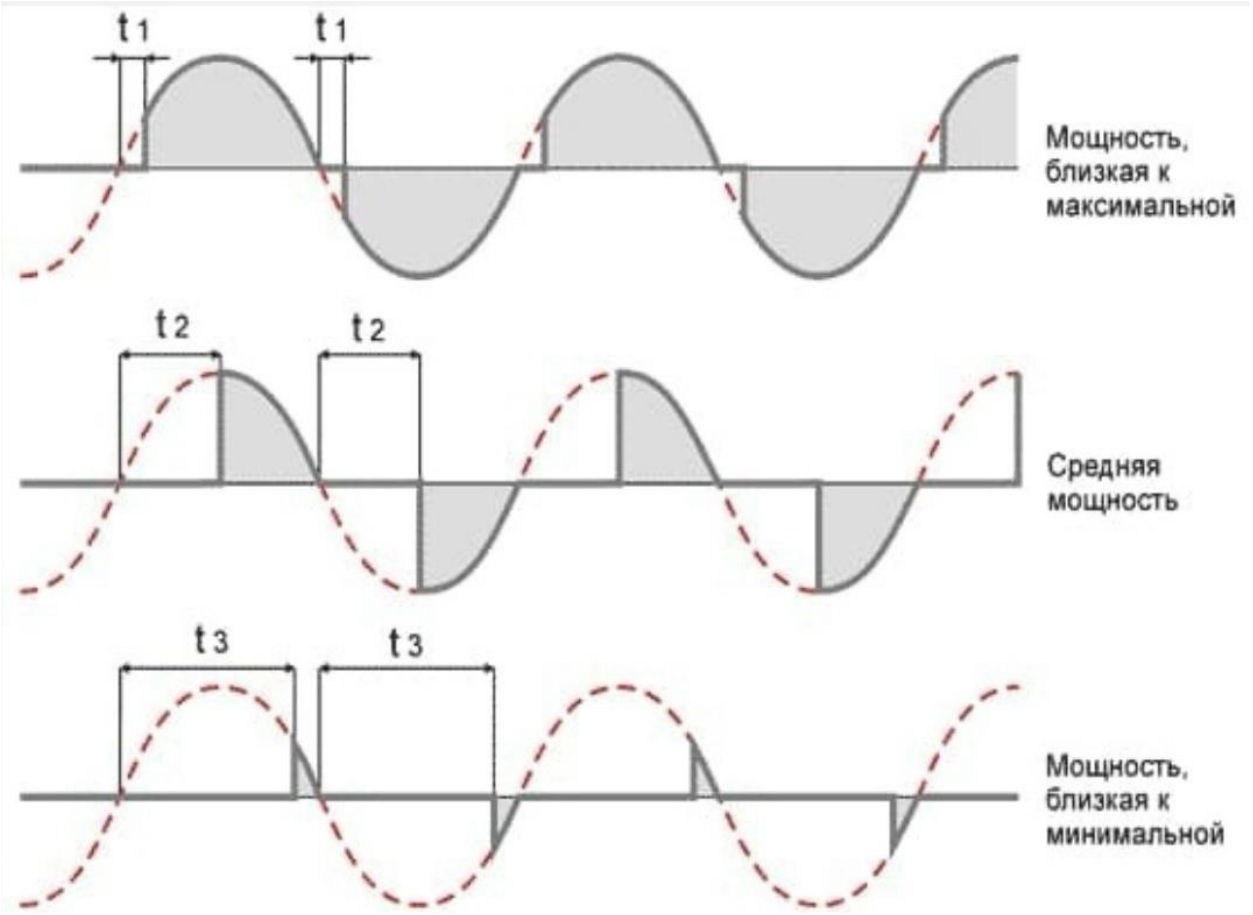

Рисунок 13. Диаграмма работы тиристорного регулятора напряжения.

Специфика управления работой тиристоров под действием переменного тока заключается в том, что управляющие сигналы должны быть синхронизированы с частотой, соответствующей частоте сети переменного тока. То есть отпирающие сигналы следует подавать на управляющий электрод только тогда, когда напряжение на аноде положительное относительно катода.

3.2. Эмпирическая модель системы управления.

Эмпирическая модель системы управления тиристорами для распределенной системы управления установкой анодного оксидирования представлен на рис. 14.

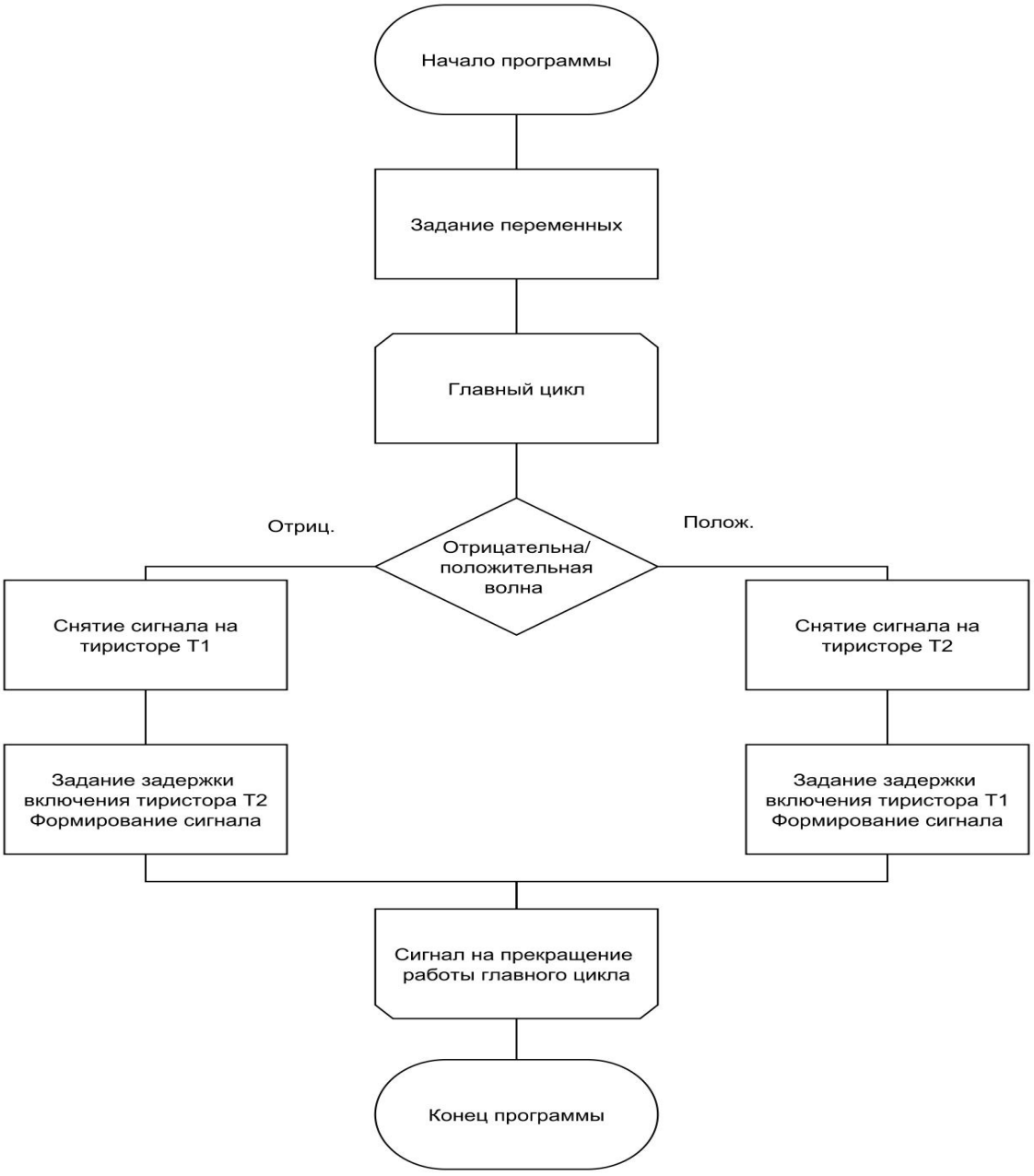

Рисунок 14. Эмпирическая модель системы управления.

Алгоритм функционирования модуля управления построен на обработке сигналов перехода фазы через ноль через механизм прерывания.

Сигнал перехода питающего напряжения через ноль вызывает процедуру обработки прерывания, в которой инициализируется прерывание от таймера.

В момент перехода через ноль из положительной полуволны в отрицательную или наоборот, срабатывает программное прерывание которое запускает отдельный процесс расчета задержки. Спустя определенное количество времени рассчитанное при определении задержки. Будет подан сигнал на открытие первого или второго тиристора в зависимости от характера перехода через ноль (в положительную или отрицательную стороны). Работа по переключению тиристоров будет длиться пока не будет прекращена работа основного цикла, что может произойти либо штатно по сигналу управляющего блока или не штатно при прекращении питания.

3.3. Реализация алгоритма.

Две платы Arduino Nano для симуляции двух отдельных процессов. Одна - «controller» - управляющая программа для системы, подает напряжение на выходы контроллера симулируя включение и отключение тиристоров.

Вторая - «plant» - программа отвечает за подачу сигнала с определённым периодом для считывания его первой платой и отвечает за связь с компьютером, через COM порт.Платы симулируют работу блока управления тиристорами. Одна плата, условно называемая «plant» генерирует сигнал в виде меандра. Он выполняет функцию синусоидального сигнала питающего напряжения. Условно принято, что переход меандра из состояния «ноль» в «единица» (или же повышение напряжение на выходе контакта Arduino с нуля до ~5V) будет являться переходом эмитируемой синусоиды от отрицательной

полуволны к положительной через «ноль». Соответственно, обратное значение (переход синусоиды от положительной полуволны в отрицательную) будет сигнализироваться падением напряжения с ~5V до нуля. Так же, помимо этого, данная плата отвечает за связь опытного макета с ПК через COM port посредством USB кабеля. Благодаря этому у нас есть возможность пронаблюдать отработку алгоритма через графики. Данный функционал доступен в стандартной Arduino IDE.

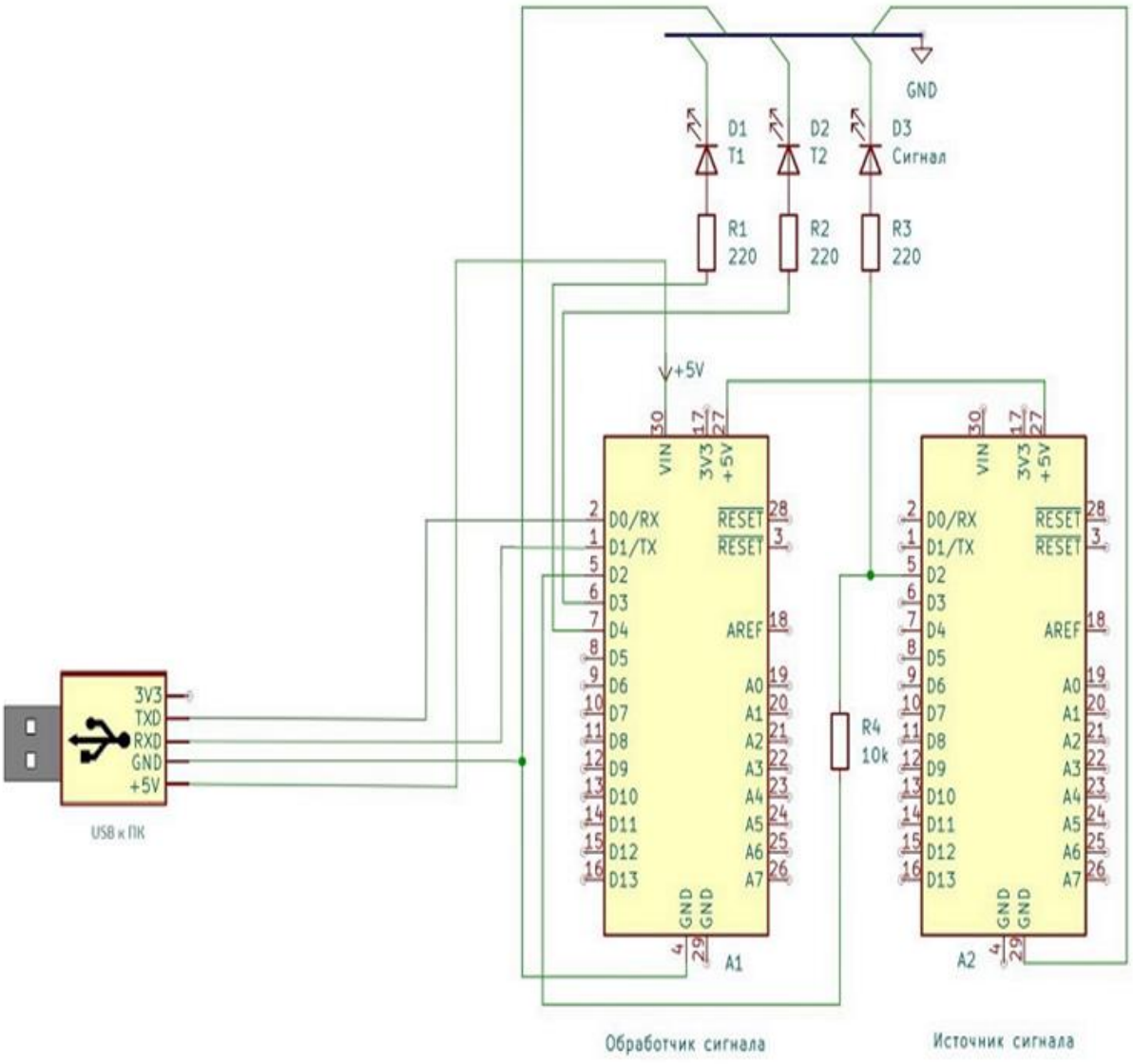

Рисунок 15. Принципиальная схема работы блока управления.

Вторая плата условно называемая «controller» отвечает за обработку получаемого сигнала (меандр), его обработку, а именно:

1) Определение переход от «нуля» к «единице» или от «единицы» к «нулю» произошел.

2) В зависимости от этого запуск расчета задержек для первого или второго тиристора.

3) Включение или отключение сигнальных светодиодов в зависимости от происходящего процесса.

3.4. Имитационный стенд.

Для экспериментальной апробации разработанного алгоритма решения поставленной задачи разработки микроконтроллерного блока управления тиристорами установки анодного оксидирования создан имитационный стенд, представленный на рис.16.

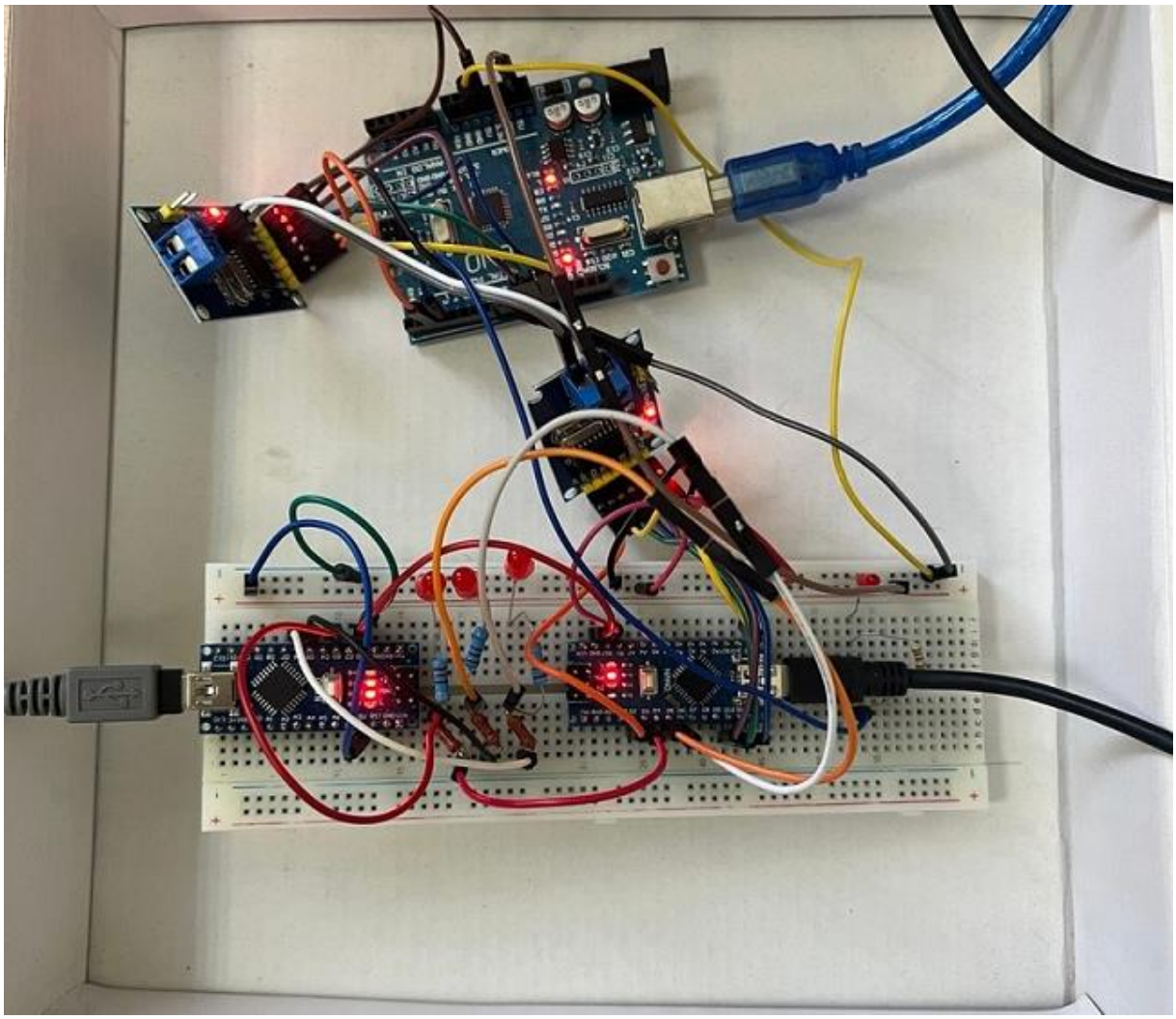

Рисунок 16. Имитационный стенд блока управления тиристорами в установке анодного оксидирования.

Имитационный стенд включает в себя:

2 микроконтроллера Arduino Nano

2 микроконтроллера MCP2515

Коммуникационный контролллер Arduino Uno

Шину CAN

Светодиоды, которые имитируют работу тиристоров

Принципиальная схема работы имитационного стенда представлена на рис. 17.

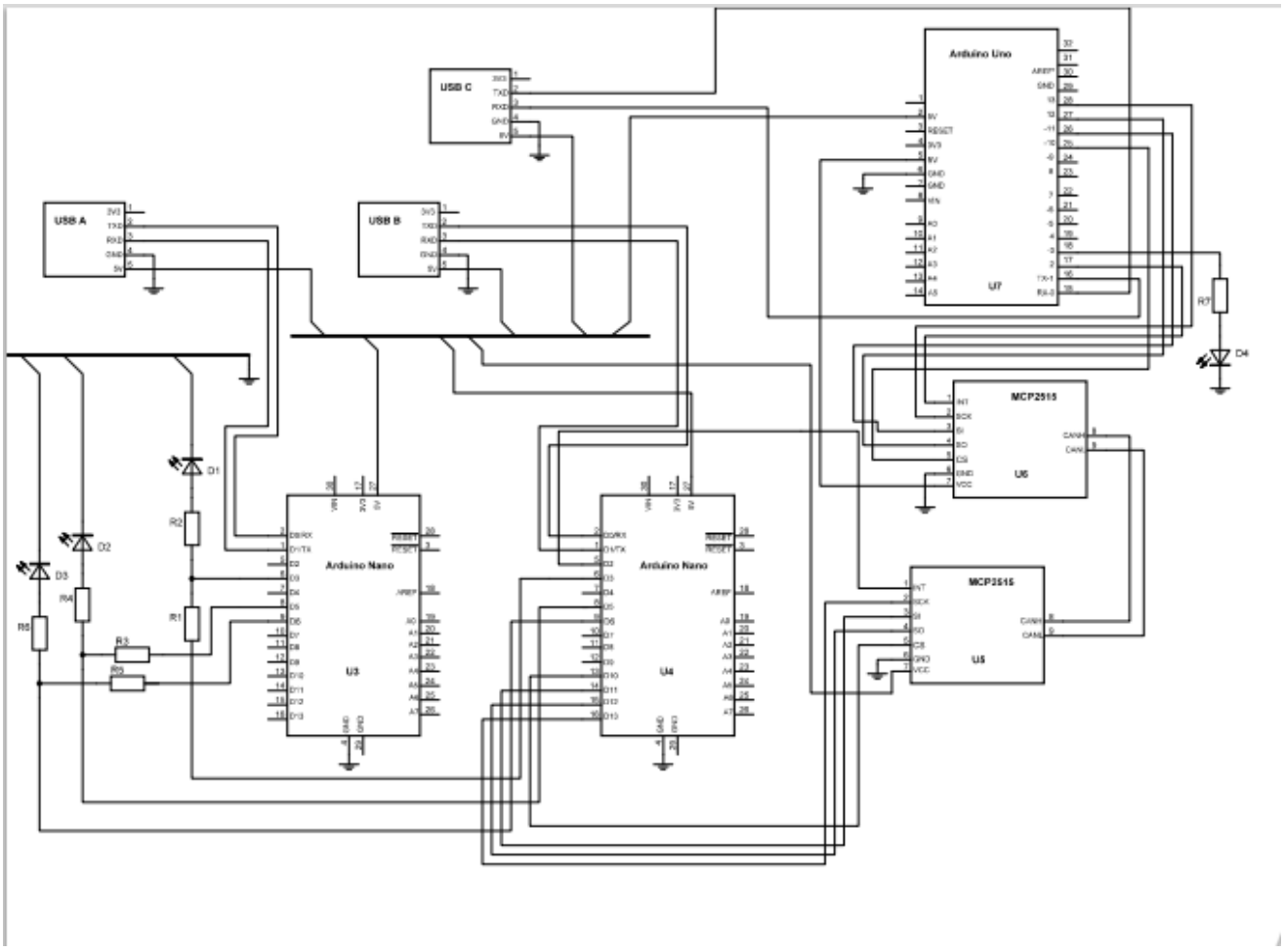

Рисунок 17. Принципиальная схема работы имитационного стенда.

3.5. Полученные результаты.

Текущий мониторинг поведения сигналов управления представлен в виде графиков на рис.18 и рис.19.

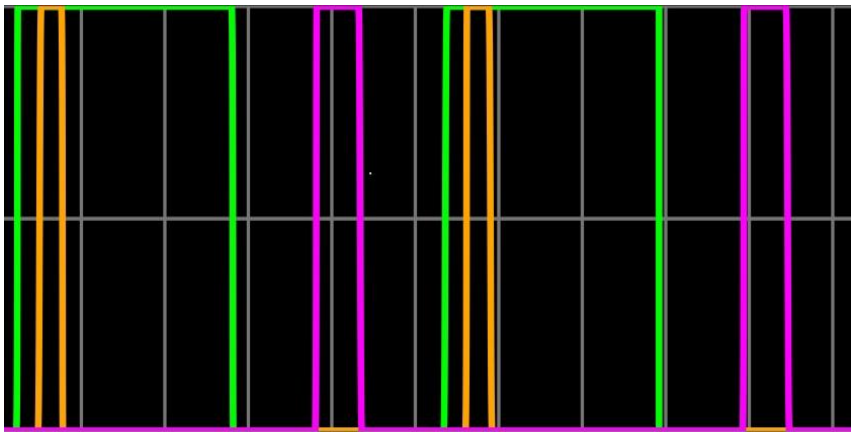

Рисунок 18 – Графики меандра, первого и второго тиристора.

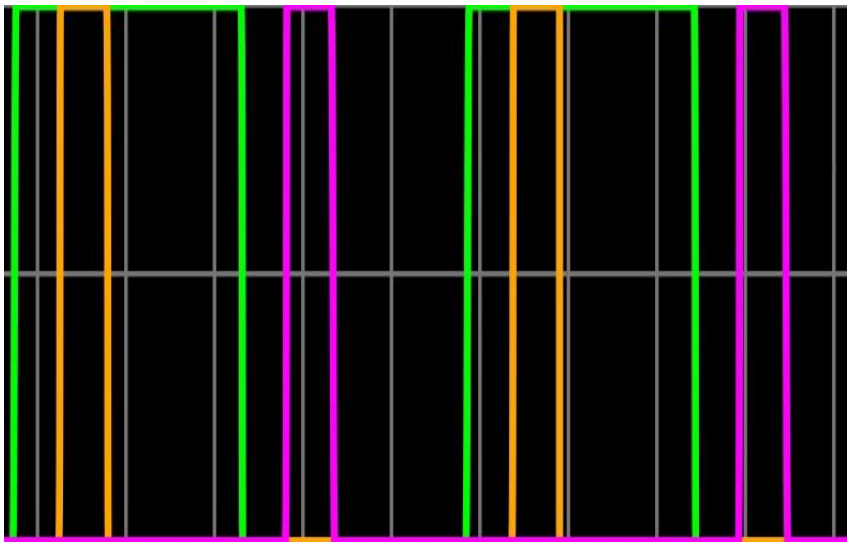

Рисунок 19 – Графики меандра, первого и второго тиристора с измененными временными характеристиками.

Зеленый график – меандр. Это основной сигнал, относительно которого рассчитываются задержки и подаются сигналы на тиристоры.Оранжевый график – сигнал на включение тиристора 1. Сигнал поступает с определенной задержкой когда питающее напряжение переходит через ноль в положительную полуволну. На рисунке 15 этот момент отображен как переход меандра от нуля к единице. Можно видеть, как после перехода меандра к

единице, спустя некоторый промежуток времени возникает краткосрочный сигнал на включение тиристора 1.

Фиолетовый график - сигнал на включение тиристора 2. Сигнал на включение тиристора 2 возникает аналогично сигналу для тиристора 1. С тем отличием, что сигнал будет подан при переходе питающего напряжения через ноль в отрицательную полуволну.

На рисунках 15 и 16 можно заметить отличия во времени перед подачей сигнала на включение тиристора 1 и 2, а также период действия этих сигналов. Данные параметры настраиваются при прошивке контроллера.

3.6. Описание программы.

Для макетирования были использованы 2 платы arduino nano для симуляции двух отдельных процессов. Один, названный «controller» представляет собой управляющую программу для системы. Она подает напряжение на выходы контроллера симулируя включение и отключение тиристоров. Это зависит от того, какого рода сигнал поступил на вход контроллера. Сигналом служит переход через ноль питающего напряжения в отрицательную или положительную полуволну.

Алгоритм состоит из следующих процессов и состояний в них:

- InitDelay – устанавливает время задержки сигнала.

o setDelay.

- CheckSensorAC – проверяет состояние сигнала на входе контроллера и состоит из следующих стостояний (state).

o waitSynchro – дожидается, когда на входе не будет сигнала и переходит в следующее состояние.

o waitPosSensorAC – отключает тиристоры, если на входе есть сигнал.

o waitPosDelay – задержка на определенное время, а затем включение тиристора 1.

o waitNegSensorAC - отключает тиристоры, если на входе отсутствует сигнал.

o waitNegDelay - задержка на определенное время, а затем включение тиристора 2 и переключение в состояние waitPosSensorAC.

В макете для симуляции такого сигнала служит вторая плата, выполняющая программу «plant». Эта программа отвечает за подачу сигнала с определённым периодом для считывания его первой платой, а также отвечает за связь с компьютером, через COM порт. Благодаря этому, можно определить время «открытия» и «закрытия» тиристоров, которые эмулирует контроллер наглядным образом через графики. Графики были построены штатными средствами Arduino IDE, а именно через Serial Monitor.

Алгоритм состоит из следующих процессов и состояний в них:

- InitPeriod – задает числовое значение задержки для подачи сигнала нв вывод контроллера.

o setPeriod – состояние внутри процесса.

- CheckSensorAC – включаем и отключаем подачу сигнала на вывод контроллера в течении задержки установленной в предыдущем процессе.

o set_signal – выдаем сигнал и ждем определённое время равное значению задержки.

o wait – прекращаем выдачу сигнала и ждем определённое время равное значению задержки.

- Monotor1 – отвечает за связь с компьютером через COM порт.

o SerialPrint.

Код программы представлен в Приложении 1.

ЗАКЛЮЧЕНИЕ

В работе на задаче разработки периферийного модуля управления тиристорным блоком отработана технология построения распределенной микроконтроллерной системы на базе процесс-ориентированной парадигмы программирования и апробирован проприетарный протокол прикладного уровня popCAN.

Отработанная информационная технология предполагает централизованное описание управляющего алгоритма в виде, независимом от топологии целевой распределенной системы управления. Технология сокращает трудоемкость создания распределенных микроконтроллерных систем управления, открывает возможность разработки бесшовных методов верификации.

По теме работы была подготовлена статья на международную конференцию SibirCon и сделан доклад на МНСК-24.

В дальнейшем планируется использовать отработанную технологию для создания системы управления установкой анодного оксидирования в целом (на настоящий момент 11 периферийных модулей).

СПИСОК ЛИТЕРАТУРЫ

1. Васильева, Н. Г. К вопросу автоматизации технологического процесса нанесения гальванических покрытий на примере анодного оксидирования / Н. Г. Васильева, Л. Н. Грачева. — Текст: непосредственный // Технические науки: традиции и инновации: материалы I Междунар. науч. конф. (г. Челябинск, январь 2012 г.). — Челябинск: Два комсомольца, 2012. — С. 58-62. — URL: https://moluch.ru/conf/tech/archive/6/1579/ (дата обращения: 23.07.2023).

2. Гамбург, Ю. Д. Гальванические покрытия / Ю. Д. Гамбург // Справочник по применению. — Москва:Техносфера, 2006. — С. 216с.

3. Евгений Марков Архитектура распределённых приложений / Евгений Марков [Электронный ресурс] // itweek : [сайт]. — URL: https://www.itweek.ru/infrastructure/article/detail] (дата обращения: 2024).

4. Зюбин В.Е.; Гаранина Н.О.; Ануреев И.С.; Старолетов С.М. На пути к программированию без топологии для киберфизических систем с процессно-ориентированной парадигмой. *Датчики* 2023, *23*, 6216.

5. Зюбин, В. Е. Процесс-ориентированное программирование: Учеб. пособие/ В. Е.Зюбин – Новосиб. гос. ун-т. – 2011. – 194 с.

6. Зюбин, В. Е. Язык «Рефлекс» – диалект Си для программируемых логических контроллеров // Шестая международная научно-практическая конференция «Средства и системы автоматизации» CSAF. – Т. 6.

7. Зюбин В.Е. Язык Рефлекс - диалект Си для программируемых логических контроллеров // Шестая международная научно-практическая конференция "Средства и системы автоматизации " CSAF'06 / Томск, 1-3 ноября 2005 г. Томск: ТУСУР, 2005;

8. Зюбин В. Е. «Си с процессами»: язык программирования логических контроллеров // Мехатроника, автоматизация, управление. 2006 № 12 С. 31–35.

9. Косяков, М. С. Введение в распределенные системы : учебное пособие / М. С. Косяков— С.-Петербург, 2014. — 155 с.

10. Обермайссер Р. *Парадигмы управления, инициируемые событиями и временем*; Springer Science & Business Media: Берлин / Гейдельберг, Германия, 2004; Том 22. [**Google Scholar**]

11. Распределенные вычисления и приложения: учебное пособие / составитель А.А. Романов. – Ульяновск : УлГТУ, 2018. – 151 с. ISBN 978-5-9795-1802-2

12. Стекольников Ю.А., Стекольникова Н.М. Физико-химические процессы в технологии машиностроения: Учеб. пособие. – Елец: Издательство Елецкого государственного университета имени И.А. Бунина, 2008.

13. Rozov A. S. Process-oriented programming language for MCU-based automation/ RozovA.S., Zyubin V. E. // 2013 International Siberian Conference on Control and Communications(SIBCON). – IEEE, 2013. – С. 1-4

14. Таненбаум, Э. Распределенные системы. Принципы и парадигмы // Andrew S. Tanenbaum, Maarten van Steen. «Destributed systems. Principles and paradigms». — СанктПетербург : Питер, 2003. — 877 с.

15.Управление тиристором переменным напряжением Источник: https://uralchip.ru/elektrika/upravlenie-tiristorom-peremennym-napryazeniem / [Электронный ресурс] // uralchip : [сайт]. — URL: https://uralchip.ru/elektrika/upravlenie-tiristorom-peremennym-napryaz (дата обращения: 2024).

16. Albert, A. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embed. World* **2004**, *2004*, 235–252. [**Google Scholar**]

17. Aveal Протокол CAN. Описание, формат кадра, контроль ошибок. / Aveal [Электронный ресурс] // microtechnics : [сайт]. — URL: https://microtechnics.ru/protokol-can/ (дата обращения: 2024).

18. CAN CONTROLLER NETWORK DEVELOPMENT FOR MTI-350G MASS SPECTROMETER OPERATING A.B.Maleev, D.V.Novikov, A.V.Saprygin 624130, 2003 г, czl@ueip.ru

19. Leen, G.; Heffernan, D. TTCAN: A new time-triggered controller area network. *Microprocess. Microsyst.* **2002**, *26*, 77–94. [**Google Scholar**] [**CrossRef**]

20. Vladimir E. Zyubin, Towards Topology-Free Programming for Cyber-Physical Systems with Process-Oriented Paradigm / Vladimir E. Zyubin, Vladimir E. Zyubin, Vladimir E. Zyubin и Vladimir E. Zyubin [Электронный ресурс] // MDPI : [сайт]. — URL: https://doi.org/10.3390/s23136216 (дата обращения: 2024).

ПРИЛОЖЕНИЯ

Приложение 1.

1.1 - Код программы «controller» на языке Reflex.

```
program controller {
	clock 0t10ms;
	const uint64 AC_PEDIOD = 0t1000ms;
	// 220 VCC 0 - negative half-wave
	bool iSensorAC as input (read = PIND, config = DDRD, bit = PD3);
	// Thyristor 1
	bool oTr1 as output (read = PINB, write = PORTB, config = DDRB, bit = PB0);
	// Thyristor 2
	bool oTr2 as output (read = PINB, write = PORTB, config = DDRB, bit = PB1);
	time Delay;

	active process InitDelay {
		state setDelay {
			Delay = 0t500ms;
			stop;
		}
	}
	active process CheckSensorAC {
		state waitSynchro {
			if (iSensorAC == false) {
			set next state;
		}
	}
	state waitPosSensorAC {
		if(iSensorAC == true) {
			oTr1 = false;
			oTr2 = false;
		}
		set next state;
	}
	state waitPosDelay {
		timeout (Delay) {
			oTr1 = true;
			set next state;
		}
	}
	state waitNegSensorAC {
		if(iSensorAC == false) {
			oTr1 = false;
			oTr2 = false;
		}
		set next state;
	}
	state waitNegDelay {
		timeout (Delay) {
			oTr2 = true;
```

1.2 – Код программы «controller» скомпилированный для прошивки arduino nano.

```
#include "lib/r_cnst.h" /* system constants */
#include "platform.h"   /* platform dependencies */
// Atmega 368p frequency
#ifndef F_CPU
#define F_CPU 16000000UL
#endif
INT32_U _r_cur_time;
INT32_U _r_next_act_time;
/* register images and masks for output variables */
INT8 _img_PORTB;
INT8 _mask_PORTB = 0 | (1 << PB0) | (1 << PB1);
/* register images for initializable output variables*/
INT8 _img_PINB;
BOOL iSensorAC;
BOOL oTr1;
BOOL oTr2;
TIME Delay;
#define AC_PEDIOD (INT32_U) 1000UL
#define _r_CLOCK (INT32_U) 10UL
/* ===== process state constants ====== */
#define _START 0
#define _CONFIRMED 253
#define _ERROR 254
#define _STOP 255
/* ===== process state vars ====== */
uint8_t _p_InitDelay_state;
uint8_t _p_CheckSensorAC_state;
/* ===== process timer vars ====== */
uint32_t _p_InitDelay_time;
uint32_t _p_CheckSensorAC_time;
/* ===== process states enumerators ====== */
enum _p_InitDelay_states {
  _p_InitDelay_s_setDelay = _START
```

```
};
enum _p_CheckSensorAC_states {
  _p_CheckSensorAC_s_waitSynchro = _START,
  _p_CheckSensorAC_s_waitPosSensorAC,
  _p_CheckSensorAC_s_waitPosDelay,
  _p_CheckSensorAC_s_waitNegSensorAC,
  _p_CheckSensorAC_s_waitNegDelay
};
void setup() {   /* Init */
  /* Ports configuration */
  SET_BIT_8_INPUT_MODE(DDRD, PD3);
  SET_BIT_8_OUTPUT_MODE(DDRB, PB0);
  SET_BIT_8_OUTPUT_MODE(DDRB, PB1);
  /* output variables init */
  _img_PINB = READ_PORT_8(PINB);
  oTr1 = _img_PINB &= (1 << PB0) ? true : false;
  oTr2 = _img_PINB &= (1 << PB1) ? true : false;
  _r_cur_time = millis();
  _r_next_act_time = _r_cur_time;
  // #include "port_init.h"
  /*========= PROCESS INIT: ========*/
  _p_InitDelay_state = _START;
  _p_CheckSensorAC_state = _START;
  /*========= END OF PROCESSES INIT========*/
}
void loop() {   /* Control algorithm */
  _r_cur_time = get_time();
  if (_r_cur_time - _r_next_act_time >= 0) {
    // Find next activation time
    _r_next_act_time += _r_CLOCK;
    if (_r_next_act_time - _r_cur_time > _r_CLOCK) {
      _r_next_act_time = _r_cur_time + _r_CLOCK;
    }
    /*==vvvv== Input ==vvvv==*/
    iSensorAC = READ_PORT_8_BIT(PIND, PD3);
    /*==^^^^== End of Input ==^^^^==*/
```

```
/*==================================*/
 /*========= PROCESS IMAGES: ========*/

 //== Process "InitDelay":
  switch (_p_InitDelay_state) {
   case _p_InitDelay_s_setDelay: { /* State: setDelay */
    Delay = (INT32_U) 100UL;
    _p_InitDelay_state = _STOP;
    break;
   }
  }
   //== Process "CheckSensorAC":
  switch (_p_CheckSensorAC_state) {
   case _p_CheckSensorAC_s_waitSynchro: { /* State: waitSynchro */
    if (iSensorAC == FALSE){
     _p_CheckSensorAC_state = _p_CheckSensorAC_s_waitPosSensorAC;}
    break;
   }
   case _p_CheckSensorAC_s_waitPosSensorAC: { /* State: waitPosSensorAC */
    if (iSensorAC == TRUE{
     oTr1 = FALSE;
     oTr2 = FALSE;}
    _p_CheckSensorAC_state = _p_CheckSensorAC_s_waitPosDelay;
    _p_CheckSensorAC_time = _r_cur_time;
    break;}
case _p_CheckSensorAC_s_waitPosDelay: { /* State: waitPosDelay */
    if ((_r_cur_time - _p_CheckSensorAC_time) > Delay) // timeout
     {
     oTr1 = TRUE;
     _p_CheckSensorAC_state = _p_CheckSensorAC_s_waitNegSensorAC;
     }
    break;
   }
   case _p_CheckSensorAC_s_waitNegSensorAC: { /* State: waitNegSensorAC */
    if (iSensorAC == FALSE)
     {
```

```
          oTr1 = FALSE;
          oTr2 = FALSE;
          }
         _p_CheckSensorAC_state = _p_CheckSensorAC_s_waitNegDelay;
         _p_CheckSensorAC_time = _r_cur_time;
         break;
        }
        case _p_CheckSensorAC_s_waitNegDelay: { /* State: waitNegDelay */
         if ((_r_cur_time - _p_CheckSensorAC_time) > Delay){ // timeout
          oTr2 = TRUE;
          _p_CheckSensorAC_state = _p_CheckSensorAC_s_waitPosSensorAC;}
         break;
        }
       }
 /*========= END OF PROCESSES ========*/
    /*==================================*/

    /*==vvvv== Output ==vvvv==*/
    _img_PORTB = ((oTr1) ? (_img_PORTB | (1 << PB0)) : (_img_PORTB & ~(1 << PB0)));
    _img_PORTB = ((oTr2) ? (_img_PORTB | (1 << PB1)) : (_img_PORTB & ~(1 << PB1)));
    WRITE_PORT_8_MASK(PORTB,_img_PORTB,_mask_PORTB);
    /*==^^^^== End of Output ==^^^^==*/
  }
 }
```

1.3 - Код программы «Plant» на языке Reflex.

```
program Plant {
        clock 0t10ms;
        const uint64 AC_PEDIOD = 0t1000ms;
        // 220 VCC 0 - negative half-wave
        bool oSensorAC as output (read = PINB, write = PORTB, config = DDRB, bit = PB0);
        // Thyristor 1
        bool iTr1 as input (read = PIND, config = DDRD, bit = PB3);
        // Thyristor 2
        bool iTr2 as input (read = PIND, config = DDRD, bit = PB4);
        time Period;
        active process InitPeriod {
                state setPeriod {
                        Period = 0t1000ms;
                        stop;
                }
        }
        active process CheckSensorAC {
                // Подаем 5v на выход ждем 1000 мс.
                state set_signal {
                        oSensorAC = true;
                        timeout (Period) {
                        set next state;
                }
```

```
	}
	state wait {
		oSensorAC = false;
			timeout (Period) {
				restart;
			}
		}
	}

	active process Monotor1 {
		state SerialPrint looped {
					$Serial.print (oSensorAC);
					$Serial.print (" ");
					$Serial.print (iTr1);
					$Serial.print (" ");
					$Serial.println (iTr2);
					restart;
			}
		}
	}




program Plant {
	clock 0t10ms;
	const uint64 AC_PEDIOD = 0t1000ms;
	// 220 VCC 0 - negative half-wave
	bool oSensorAC as output (read = PINB, write = PORTB, config = DDRB, bit = PB0);
	// Thyristor 1
	bool iTr1 as input (read = PIND, config = DDRD, bit = PB3);
	// Thyristor 2
	bool iTr2 as input (read = PIND, config = DDRD, bit = PB4);
	time Period;
	active process InitPeriod {
		state setPeriod {
			Period = 0t1000ms;
			stop;
		}
	}
	active process CheckSensorAC {
		// Подаем 5v на выход ждем 1000 мс.
		state set_signal {
			oSensorAC = true;
			timeout (Period) {
			set next state;
		}
	}
	state wait {
		oSensorAC = false;
			timeout (Period) {
				restart;
			}
		}
	}

	active process Monotor1 {
		state SerialPrint looped {
					$Serial.print (oSensorAC);
					$Serial.print (" ");
					$Serial.print (iTr1);
					$Serial.print (" ");
					$Serial.println (iTr2);
					restart;
			}
		}
	}
```

1.4 - Код программы «plant» скомпилированный для прошивки arduino nano.

```
#include "lib/r_cnst.h" /* system constants */
```

```
#include "platform.h"   /* platform dependencies */
// Atmega 368p frequency
#ifndef F_CPU
#define F_CPU 16000000UL
#endif
INT32_U _r_cur_time;
INT32_U _r_next_act_time;
/* register images for input variables */
INT8 _img_PIND;
BOOL oSensorAC, iTr1, iTr2;
TIME Period;
#define AC_PEDIOD (INT32_U) 1000UL
#define _r_CLOCK (INT32_U) 10UL
/* ===== process state constants ====== */
#define _START 0
#define _CONFIRMED 253
#define _ERROR 254
#define _STOP 255
/* ===== process state vars ====== */
uint8_t _p_InitPeriod_state;
uint8_t _p_CheckSensorAC_state;
uint8_t _p_Monotor1_state;
/* ===== process timer vars ====== */
uint32_t _p_InitPeriod_time;
uint32_t _p_CheckSensorAC_time;
uint32_t _p_Monotor1_time;
/* ===== process states enumerators ====== */
enum _p_InitPeriod_states {
  _p_InitPeriod_s_setPeriod = _START
};
enum _p_CheckSensorAC_states {
  _p_CheckSensorAC_s_set_signal = _START,
  _p_CheckSensorAC_s_wait
};
enum _p_Monotor1_states {
  _p_Monotor1_s_SerialPrint = _START
void setup() {   /* Init */
  Serial.begin(9600);
```

```
  /* Ports configuration */
  SET_BIT_8_OUTPUT_MODE(DDRB, PB0);
  SET_BIT_8_INPUT_MODE(DDRD, PB3);
  SET_BIT_8_INPUT_MODE(DDRD, PB4);
  /* output variables init */
  oSensorAC = READ_PORT_8_BIT(PINB, PB0);



  _r_cur_time = millis();
  _r_next_act_time = _r_cur_time;
  // #include "port_init.h"
  /*========= PROCESS INIT: ========*/
  _p_InitPeriod_state = _START;
  _p_CheckSensorAC_state = _START;
  _p_Monotor1_state = _START;
  /*========= END OF PROCESSES INIT========*/
}
void loop() {   /* Control algorithm */
  _r_cur_time = get_time();
  if (_r_cur_time - _r_next_act_time >= 0) {
    // Find next activation time
    _r_next_act_time += _r_CLOCK;
    if (_r_next_act_time - _r_cur_time > _r_CLOCK) {
      _r_next_act_time = _r_cur_time + _r_CLOCK;
    }
    /*==vvvv== Input ==vvvv==*/
    _img_PIND = READ_PORT_8(PIND);
    iTr1 = _img_PIND &= (1 << PB3) ? true : false;
    iTr2 = _img_PIND &= (1 << PB4) ? true : false;
    /*==^^^^== End of Input ==^^^^==*/

    /*==================================*/



/*========= PROCESS IMAGES: ========*/
```

```
  //== Process "InitPeriod":
    switch (_p_InitPeriod_state) {
      case _p_InitPeriod_s_setPeriod: { /* State: setPeriod */
        Period = (INT32_U) 1000UL;
        _p_InitPeriod_state = _STOP;
        break;
      }
    }

  //== Process "CheckSensorAC":
    switch (_p_CheckSensorAC_state) {
      case _p_CheckSensorAC_s_set_signal: { /* State: set_signal */
        oSensorAC = TRUE;
        if ((_r_cur_time - _p_CheckSensorAC_time) > Period) // timeout
          {
          _p_CheckSensorAC_state = _p_CheckSensorAC_s_wait;
          _p_CheckSensorAC_time = _r_cur_time;
          }
        break;
      }
      case _p_CheckSensorAC_s_wait: { /* State: wait */
        oSensorAC = FALSE;
        if ((_r_cur_time - _p_CheckSensorAC_time) > Period) // timeout
          {
          _p_CheckSensorAC_state = _START;
          }
        break;
      }
    }


//== Process "Monotor1":
    switch (_p_Monotor1_state) {
      case _p_Monotor1_s_SerialPrint: { /* State: SerialPrint */
        // #C code insertion:
        Serial.print (oSensorAC);
```

```
        // #C code insertion:
        Serial.print (" ");
        // #C code insertion:
        Serial.print (iTr1);
        // #C code insertion:
        Serial.print (" ");
        // #C code insertion:
        Serial.println (iTr2);
        break;
       }
      }
   /*========= END OF PROCESSES ========*/
   /*==================================*/

   /*==vvvv== Output ==vvvv==*/
   WRITE_PORT_8_BIT(PORTB, oSensorAC, PB0);
   /*==^^^^== End of Output ==^^^^==*/
  }
}
```