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
«___»_____ 2023 г.

Разработка программно-аппаратной части стенда для исследования цифрового двойника вентильного двигателя

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.144 ПЗ ВКП

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ П. О. Шабуров
«___»_____2023 г.

Автор работы,
студент группы КЭ-405
_____Д. Т. Тазеева
«___»_____2023 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2023 г.

Челябинск-2023

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
«___»_____2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Тазеевой Динаре Тимуровне
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка программно-аппаратной части стенда для исследования цифрового двойника вентильного двигателя» утверждена приказом по университету от «25» апреля 2023 г. № 753-13/12.

2. **Срок сдачи студентом законченной работы:** 01 июня 2023 г.

3. **Исходные данные к работе:**

   3.1 микроконтроллер;

   3.2 3 аналоговых датчика тока (на основе датчика Холла);

   3.3 3 датчика температуры;

   3.4 3 датчика тока;

   3.5 преобразователь интерфейсов.

4. **Перечень подлежащих разработке вопросов:**
    1. Анализ предметной области. Цифровой двойник электродвигателя;
    2. Определение требований;
    3. Проектирование;
    4. Реализация;
    5. Тестирование.
5. **Дата выдачи задания:** 1 декабря 2022 г.

Руководитель работы _____/ *П. О. Шабуров/*

Студент _____/ *Д. Т. Тазеева /*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Анализ предметной области. Цифровой двойник электродвигателя	01.03.2023	
Определение требований	15.03.2023	
Проектирование	01.04.2023	
Реализация	01.05.2023	
Тестирование	15.05.2023	
Компоновка текста работы и сдача на нормоконтроль	22.05.2023	
Подготовка презентации и доклада	30.05.2023	

Руководитель работы _____/ *П. О. Шабуров* /

Студент _____/ *Д. Т. Тазеева* /

АННОТАЦИЯ

Д. Т. Тазеева. Разработка программно-аппаратной части стенда для исследования цифрового двойника вентильного двигателя. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2023, 60 с., 27 ил., библиогр. список – 10 наим., 1 прил.

В рамках данной выпускной квалификационной работы рассматривается процесс разработки программно-аппаратной части стенда для исследования цифрового двойника вентильного двигателя. Осуществляется анализ предметной области и основных технологических решений. Производится проектирование и реализация макета комплекса. Разрабатывается плата приема и передачи данных для управления работой микроконтроллера. За основу взаимодействия взят протокол обмена данными TCP/IP. Производится тестовая передача данных в программную среду Twin Builder. Работа выполнена на основании материалов открытой печати и интернет – ресурсов.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Цифровые двойники и их распространение обусловлено фактом развития такого понятия, как цифровизация процессов производства, что представляет собой замену аналоговых или физических ресурсов на информационные или цифровые. По этой причине многие предприятия начали изучать различные варианты того, как цифровизация может помочь им достичь цель извлечь выгоду как с операционной, так и со стратегической стороны, стремясь не отставать от последних технологий.

Термин «цифровой двойник» в различных научных источниках трактуется по-разному. Он представляет собой модель, внедренную в существующий продукт как для хранения информации обо всех неисправностях и недоработках изделия, так и для постоянного обновления в процессе физического использования.

Иначе цифровой двойник называют цифровой моделью, которая функционирует на основании информации с датчиков, установленных на физическом объекте, что позволяет симулировать поведение объекта в реальном мире [1].

Математическая модель – основа цифрового двойника, которая отображает физические процессы, происходящие конкретно в объекте.

Приемопередатчик – это устройство, с помощью которого можно передавать сигналы между двумя разными платформами связи. Приемопередатчик соединяет стык вычислительной машины с локальной сетью – Ethernet.

Ethernet содержит электронные устройства, сигнал с которых передается от витой пары к оптовлооконным кабелям.

Для передачи данных с сети Ethernet в работе используется протокол tcp/ip. Данный протокол применяется с целью передачи цифровых сигналов и описывает способы и правила передачи данных, стандарты связи между компьютерами, а также содержит соглашения о маршрутизации и межсетевом взаимодействии [2].

Целью представленной выпускной квалификационной работы является разработка программно-аппаратной части стенда с целью исследования цифрового двойника вентильного двигателя.

Для достижения поставленной цели, необходимо решить следующие задачи:

1. Проанализировать существующие аналоги и выявить их преимущества и недостатки.
2. Выявить функциональные и нефункциональные требования к комплексу.
3. Спроектировать архитектуру программно-аппаратного комплекса.
4. Реализовать программно-аппаратный комплекс.
5. Протестировать и оценить работоспособность разработанного программного комплекса.

Для решения научно-исследовательских задач и получения результата исследования в данной работе используются в первую очередь теоретические методы (анализ, сравнение, моделирование и т.д.), затем эмпирические методы, обеспечивающие сбор данных (наблюдение, изучение продуктов деятельности, документации, эксперимент и т.д.) и наконец математические методы – обработка количественных данных.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ЦИФРОВОЙ ДВОЙНИК ЭЛЕКТРОДВИГАТЕЛЯ

На сегодняшний день электрические машины повышенной мощности являются довольно сложными объектами. Любая неисправность или поломка влечет за собой дорогой ремонт, в следствие чего машина выходит из строя на долгий срок. Цифровой двойник способен решить данные проблемы, оповещая о развитии поломки раньше, чем штатная система [3].

В такой области, как строительство, данная технология работает посредством сбора и обработки данных с помощью различных датчиков. Цифровой двойник в этом случае способен воспроизвести каждую деталь объекта, учитывается поведение объекта в целом, а также процессы, связанные со строительством, принимая во внимание отдельные компоненты [4].

Электродвигателем называется машина, которая при помощи магнитного поля преобразует электрическую энергию в механическую.

Электродвигатель состоит из 3 основных частей:

- механическая;
- электромагнитная;
- преобразователь.

Преобразователь управляет электромагнитной частью электродвигателя, в результате чего приводится в движение механическая часть.

Преобразователи частоты (ПЧ) – электронные устройства, необходимые для регуляции сетевых параметров под потребности электродвигателей. Позволяют плавно менять частоту вращения и крутящего момента, защищая двигатели электрических машин от перегрузок, энергопотерь и аварийных ситуаций. Также, ПЧ предназначен для исследования и приведения двигателя в движение.

Цели частотного преобразователя для электродвигателя:

- предотвращение перегрева оборудования;
- снижение вероятности перегорания мотора при скачках напряжения;
- исключение износа оборудования;
- увеличение срока службы механизмов;

– автоматическое регулирование;

– контроль двигателя;

– предотвращение «Сухого хода» [5].

Асинхронный электродвигатель является основой большинства электроприводов. Чаще всего для управления таким двигателем используется как раз таки частотный преобразователь – инвертор с ШИМ-регулированием.

Синхронный двигатель – это электродвигатель, индуктор которого состоит из постоянных магнитов. В состав такого двигателя, как, собственно, и любого другого, входят статор и ротор. Статор является неподвижной частью двигателя, а ротор – вращающейся. Ток, температура и магнитное поле в нем устанавливаются при помощи датчиков.

Вентильный электродвигатель – один из видов электртческих двигателей постоянного тока, представляет собой такую систему, которая включает в себя преобразователь – коммутатор. Электронный коммутатор представляет собой автономный инвертор, посредством которого осуществляется питание обмоток двигателя. Для работы также необходимы датчики тока, напряжения и другие измерительные устройства [6].

В качестве стенда для данной работы был выбран двигатель ДБУ-210 (рисунок 1.1). По характеристикам этот двигатель моментный, служит для создания большого момента и рассчитан на 100 об./мин. и момент 120 Н/м, что составляет 10 рад./с. Мощность данного двигателя будет составлять 3 кВт. ДБУ-210 – синхронный двигатель с возбуждением от постоянных магнитов имеет три фазы соединения «звезда». Сопротивление 1 фазы равно 11,75 Ом.

Рисунок 1.1 – Двигатель ДБУ-210

Также в ходе работы используются следующие компоненты:

- преобразователь частоты ALTIVAR ATV630 VFD;
- исследуемый вентильный двигатель;
- нагружаемый вентильный двигатель;
- аналоговый датчик тока (на основе датчика Холла) SS49E;
- платиновый датчик температуры L-серии LN222;
- датчик тока ACS712 30A;
- микроконтроллер STM32F303VCT6.

1.1 Пример реализации цифрового двойника двигателя

В производстве сегодня очень широко используют технологию цифрового двойника. Она помогает разработчикам создавать цифровые копии различных типов двигателей, что значительно экономит время и средства на разработку и тестирование новых изделий. Многие крупные компании активно внедряют эту технологию для различных типов двигателей. Рассмотрим несколько таких типов.

«УМНЫЙ» ДВОЙНИК МОРСКОГО ДВИГАТЕЛЯ

Двигателестроительная корпорация Ростеха совместно с «Центром компьютерного инжиниринга» НТИ СПбПУ работает над созданием цифрового двойника морского газотурбинного двигателя (рисунок 1.2). Это позволит контролировать все этапы жизненного цикла двигателя и увеличить надежность российских морских двигателей.

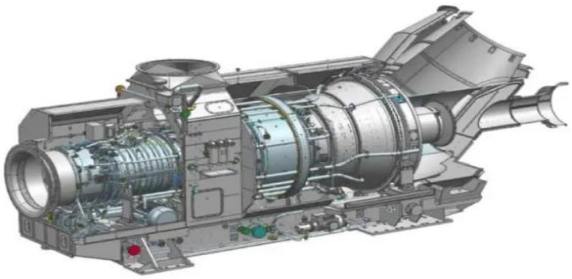


Рисунок 1.2 – Морской газотурбинный двигатель М90ФР

Создание цифровой модели также позволит сократить время и стоимость проектирования нового поколения двигателей. "Умный" двойник позволит снизить затраты на создание опытных образцов и управлять жизненным циклом продукта, что способствует повышению надежности и коммерческой привлекательности российских морских газотурбинных двигателей [7].

1.2 Обзор аналогов

Особенности ЦД вентильного электродвигателя:

- Вентильный электродвигатель приводится в движения с помощью электричества, это позволяет ему пракически сразу достичь максимального крутящего момента. От чего двигатель более ровно и динамично разгоняется.
- Принцип работы Вентильного двигателя: Вентильный двигатель работает по четкому позиционированию магнитов на роторе относительно электромагнитного импульса на фазных электрических обмотках. Датчики отслеживают положение магнитов в пространстве и изменяют передачу электрического сигнала, что позволяет валу вращаться. При этом не требуется использование скользящего контакта, что относит вентильные двигатели к категории бесколлекторных электродвигателей.

Погружной электродвигатель

В настоящее время в отрасли нефтедобычи проявляются такие тенденции, как рост экологичности добычи и переработки нефти и газа, рост требований заказчика к нефтесервисным технологиям и операционной эффективности, санкции на покупку зарубежного оборудования. Как следствие, активизируются работы, связанные с энергетической эффективностью нефтедобычи и увеличением межремонтных периодов основного оборудования, в том числе и погружной электродвигатель (ПЭД) (рисунок 1.3).

Рисунок 1.3 – Погружной электродвигатель

Для эффективного управления ПЭД появилась необходимость в построении математической модели.

Тем самым предлагается способ создания цифровой копии ПЭД с наименьшей возможной вычислительной сложностью. На примере ПЭД ЭД(Т) 45-117-1000 рассматривается создание цифровой модели с малым количеством вычислительных операций, а также минимальными затратами времени выполнения и использованием меньшего количеством ресурсов, необходимых для аппаратной реализации.

Погружной электродвигатель является асинхронным с короткозамкнутым ротором. Математическое моделирование таких машин обычно основывается на теории электромеханического преобразования энергии и может приводить к математическим моделям в виде системы дифференциально-алгебраических уравнений, которые решаются с помощью численных методов, что требует значительных вычислительных ресурсов [8].

ЦД насоса с электродвигателем

Компании PTC, NI и HPE разработали цифровой двойник насоса и его клапана. Такой ЦД демонстрирует комплексную систему, включающую в себя динамику жидкости, электромеханику, электромагнетизм и теплопередачу. Эта модель позволит решить задачу управления и контроля температуры электродвигателя и его компонентов в номинальном режиме работы.

На большинстве электродвигателей датчики температуры не установлены. Оценку температуры можно провести по входной мощности, току и напряжению, но этот способ не является точным. Даже если датчики на электродвигателе имеются, то использовать их выходит довольно дорого, а также полученные данные часто бывают неточны или же обрабатываются с задержкой.

Наличие цифрового двойника продукта способно решить проблему определения температуры и повысить долговечность двигателя, тем самым добиться оптимального температурного режима работы двигателя и работы насоса с лучшей производительностью. При наличии цифрового двойника для моделирования состояний системы требуется лишь информация с двух датчиков положения клапанов насоса, способных определить расход. Использование виртуальных датчиков, встроенных в расчётные модели, снижает необходимость в

использовании физических датчиков. Благодаря цифровому двойнику и информации с датчиков на насосе, можно определять температуру двигателя, электрический ток, скорость и давление потока жидкости в разных местоположениях и в любой момент времени.

Также цифровой двойник прогнозирует температуру компонентов двигателя в будущем. Такое преимущество позволяет заблаговременно предпринимать действия для сохранения ресурса двигателя и его компонентов [9].

Для более наглядного сравнения приведенных ЦД двигателей были выделены следующие критерии, результаты сведены в таблицу 1.1.

Таблица 1.1 – Результаты обзора аналогов

	ЦД погружного электродвигателя	ЦД насоса с двигателем	ЦД вентильного электродвигателя
Собственное ПО	+	- (в ПК Ansys)	+
Отечественная разработка	+	-	+
Простота использования	-	+	+
Модель в составе сложной технической системы	-	+	-
Визуализация процесса	-	+	-

Выводы:

1. Первый аналог имеет несомненное преимущество, поскольку имеет собственное программное обеспечение, реализующее решение соответствующей системы дифференциальных уравнений. Второй аналог разработан в программном комплексе численного моделирования Ansys.

2. Второй аналог гораздо удобнее в использовании, поскольку разрабатывался как коммерческий продукт в удобном для пользователя программном комплексе.

3. В первом аналоге учитываются параметры и свойства только самого электродвигателя, тогда, как второй разрабатывался как структурный компонент в составе сложной технической системы (насоса).

4. Преимущество второго аналога в наглядной визуализации процесса (изополя температур, давления, скоростей и т.д.).

2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

В соответствии с техническим заданием были выявлены требования к комплексу. Разработаны функциональные требования – задачи, которые поставлены перед системой для ее полного функционирования, а также нефункциональные – они определяют, как должна работать система изнутри (например, производительность, слаженность, безопасность и т. д.)

2.1 Функциональные требования

Представим функциональные требования к разрабатываемой системе:

1. разрешающая способность блока АЦП микроконтроллера должна составлять 12-bit для удовлетворительного шага дискретизации сигнала;
2. наличие блока DMA (блока прямого доступа к памяти) для распределения задач МК;
3. поддержка протокола SPI МК для передачи данных в вышестоящий уровень системы;
4. обеспечение передачи данных между МК и сервером в формате big-endian (software) по протоколу TCP, передача внутренними средствами Twin Builder в модель системного уровня.

2.2 Нефункциональные требования

Разрабатываемая система должна соответствовать следующим нефункциональным требованиям:

1. время установления связи между сервером и МК не должно превышать 30 секунд;
2. комплект датчиков, состоящий из датчика тока, датчика магнитного поля, датчика температуры, должен быть установлен в каждую фазу электродвигателя;
3. процесс передачи данных в программную среду должен длиться непрерывно после установления связи между МК и сервером.

3 ПРОЕКТИРОВАНИЕ

В представленной главе описаны методы реализации ПАК. Здесь будут рассмотрены схемы и алгоритмы решения поставленных задач.

Для начала была спроектирована структурная схема стенда для сбора и обработки данных (рисунок 3.1), верифицирования цифрового двойника.

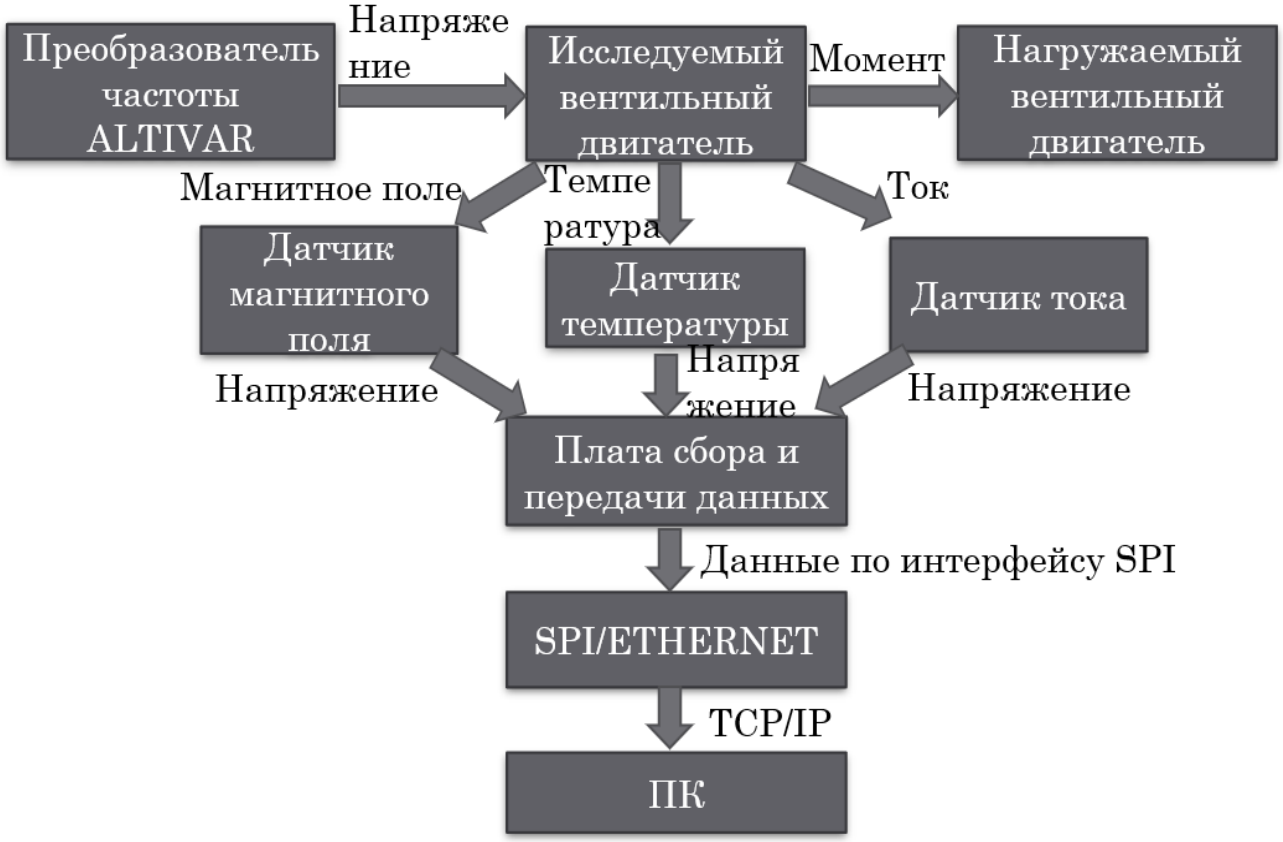


Рисунок 3.1 – Структурная схема стенда для сбора информации цифрового двойника

Преобразователь частоты передает напряжение на исследуемый нами вентильный двигатель, который в свою очередь передает момент на нагружаемый нами вентильный двигатель, выступающий в роли генератора, а также данные на датчики. На датчик магнитного поля передаются данные магнитного поля, на датчик температуры – данные о температуре, и на датчик тока – соответственно данные о токе. Далее с датчиков на плату сбора и передачи данных передается напряжение. Плата в свою очередь передает данные по интерфейсу SPI на SPI/ETHERNET. И следующим шагом идет передача на данных на ПК с помощью модели TCP/IP.

3.1 Архитектура аппаратной части системы

Аппаратная часть комплекса включает в себя следующие составляющие (рисунок 3.2):

– три типа датчика (датчик тока, датчик температуры, датчик магнитного поля);

– плата сбора и передачи данных (микроконтроллер);

– контроллер ENC28J60.

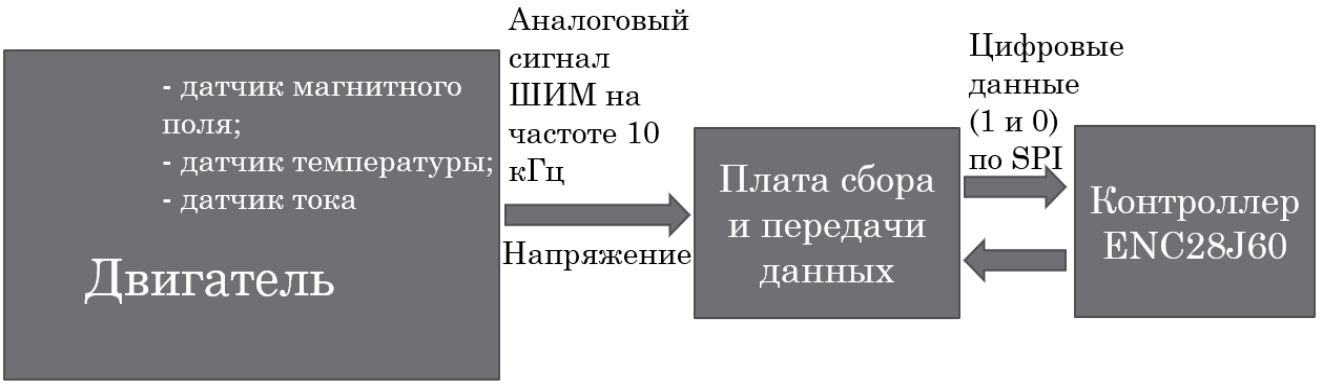


Рисунок 3.2 – Структурная схема аппаратной части

На рисунке 3.2 показана структурная схема аппаратной части. Двигатель (ДБУ-210) передает аналоговый сигнал на плату сбора и передачи данных.

ШИМ работает на частоте 10 кГц, требуется получать данные в промежутках работы ШИМ. Для этого берем частоты опроса датчиков на частоте 30 кГц от 9 значений в размере 18 байт. Так как мы хотим передать все данные сразу, мы за 30 кГц должны передать 18 байт. 1 байт должен идти на частоте 30 кГц * 18 байт = 540 кГц.

540 кГц * 10 бит (один стартовый бит и один стоповый бит данных) = 5400 бит/с. Исходя из этого мы берем частоту передачи по SPI 8 Мб/с.

Далее на контроллер, выступающий как клиент, передается цифровой сигнал (1 и 0) по SPI.

В каждой из трёх фаз электродвигателя величина тока фиксируется с помощью датчиков, которые в свою очередь преобразовывают полученные значения в аналоговые сигналы. Каждый сигнал считывается соответствующим входом порта ввода-вывода МК, который настроен на работу с помощью блока АЦП.

Контроллер ENC28J60 распознаётся сервером как виртуальный COM-порт.

Далее клиентское ПО производит манипуляцию с данными подключенного COM-порта.

3.2 Архитектура программной части системы

Как было сказано ранее аналоговый сигнал поступает на соответствующий вход порта ввода-вывода. Этот сигнал конвертируются с помощью блока АПЦ в 8-байтное число. Затем по причине условий работы протокола SPI каждое сформированное двух байтное число разбивается на 2 отдельных байта, которые по очереди выставляются в памяти регистра на вывод SPI.

Клиентская часть отвечает за управление работой МК, а именно разрешение и запрещение пересылки данных по SPI, а также за настройку сокета для передачи пришедших данных с МК по протоколу TCP.

Структурная схема программной части системы изображена на рисунке 3.3.

Рисунок 3.3 – Структурная схема программной части

3.3 Алгоритмы работы системы

Рассмотрим алгоритм работы системы (рисунок 3.4).

После успешной инициализации компонентов через плату происходит передача команды на запуск, далее происходит проверка получения команды на передачу данных. В случае, если команда получена, в буфере обмена происходит считывание значения, в противном случае проверка повторяется.

Далее, если считывание прошло успешно, происходит передача данных по протоколу SPI. Следующим шагом считывается пришедший байт в буфер, после чего байты преобразовываются в биты, а биты в реальные показания формата double. Для передачи по TCP данные необходимо преобразовать в формат big-endian.

При попытке соединения с сервером следует дождаться установки разрешающего флага о передаче данных. Если же при соединении произошла ошибка, анализ данных закрывается. В случае успешной передачи, на протяжении 100 секунд происходит считывание полученного значения, в результате чего данные с датчиков будут получены.

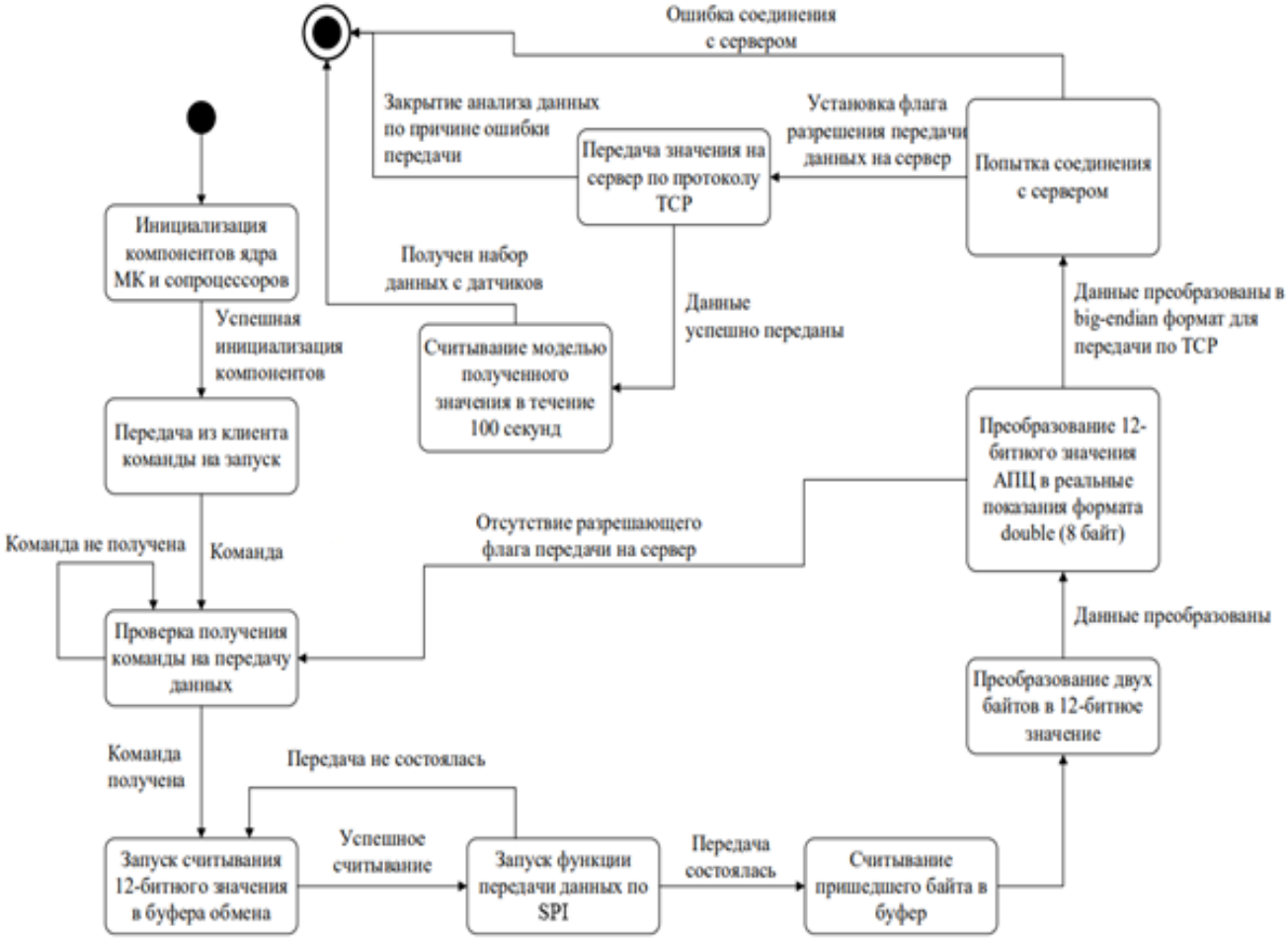


Рисунок 3.4 – Диаграмма состояний и переходов

3.4 Подбор компонентной базы

На основе полного перечня требований к разрабатываемой аппаратной части комплекса, был произведен подбор компонентной базы.

3.4.1 Подбор микроконтроллера

В качестве элемента управления аппаратной частью был определён техническим заданием микроконтроллер STM32F303VCT6 (рисунок 3.5).

– способен обрабатывать 2 канала АЦП;

– имеет блок DMA, цель которого – перенос данных из памяти периферии в оперативную без использования ресурсов ядра МК.

Рисунок 3.5 – Микроконтроллер

3.4.2 Подбор датчиков

В проекте используются датчики для измерения температуры (рисунок 3.6). Платиновый датчик температуры L-серии отличается долговременной стабильностью, компактностью и высокой точностью в широком температурном диапазоне.

Рисунок 3.6 – Внешний вид датчика температуры

Рассматриваем электрическое включение датчика температуры (рисунок 3.7). Моделирование схемы проведено в программном обеспечении LTspice.

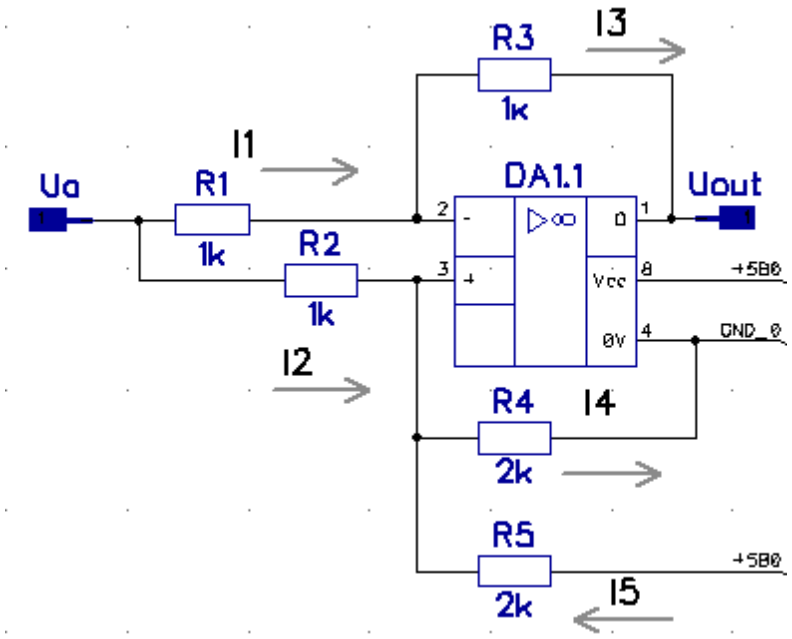


Рисунок 3.7 – Электрическое включение датчика температуры

Рассмотрим отдельно неинвертирующий вход (рисунок 3.8).

Ток, втекающий в операционный усилитель (ОУ) на + и – очень мал.

Нулевые входные токи по обоим входам (у реальных ОУ они лежат в пределах от сотых долей пА до единиц мкА).

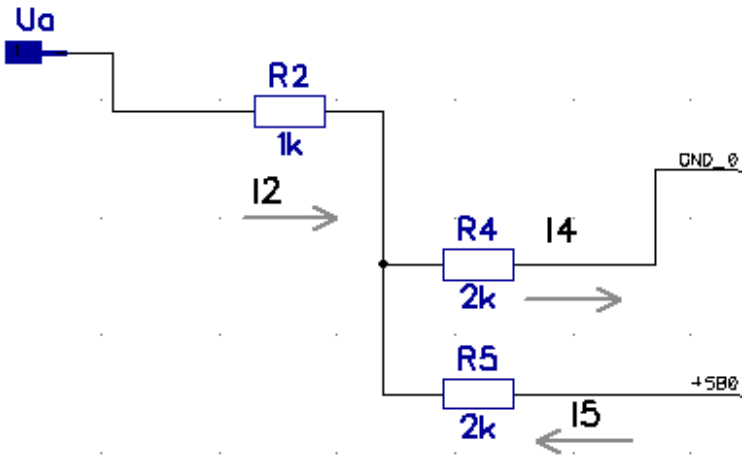


Рисунок 3.8 – Неинвертирующая часть ОУ для датчика температуры

И поэтому по первому закону Кирхгофа $I_4 = I_2 + I_5$.

По второму закону Кирхгофа рассматриваем два контура: первый содержит элементы R2, R4; второй – R5, R4.

$\begin{cases} R2 * I_2 + R4 * I_4 = Ua \\ R5 * I_5 + R4 * I_4 = 5 \end{cases}$, откуда токи $\begin{cases} I_2 = \frac{Ua - R4 * I_4}{R2} \\ I_5 = \frac{5 - R4 * I_4}{R5} \end{cases}$

$I_4 = \frac{Ua - R4 * I_4}{R2} + \frac{5 - R4 * I_4}{R5}$;

$I_4(1 + \frac{R4}{R2} + \frac{R4}{R5}) = \frac{Ua}{R2} + \frac{5}{R5}$, умножим все на $R5 * R2$

$I_4(R2 * R5 + R4 * R5 + R2 * R4) = Ua * R5 + 5 * R2$;

$I_4 = \frac{Ua * R5 + 5 * R2}{R2 * R5 + R4 * R5 + R2 * R4}$.

Напряжение в точке неинвертирующего входа 3 ОУ будет $R4 * I_4$.

$U_+ = R4 * I_4 = R4 * \frac{Ua * R5 + 5 * R2}{R2 * R5 + R4 * R5 + R2 * R4}$;

Примем $R2 = R, R5 = R4 = 2R$, тогда

$U_+ = 2R * \frac{Ua * 2R + 5 * R}{R * 2R + 2R * 2R + R * 2R} = \frac{4 * Ua + 10}{2 + 4 + 2} = \frac{Ua + 2{,}5}{2}$.

Напряжения между инвертирующим и неинвертирующим входом должны быть равны.

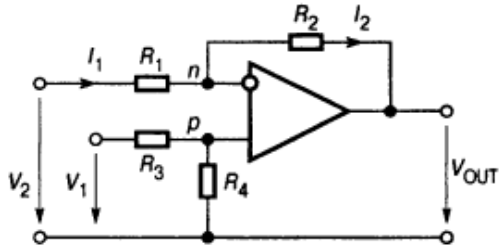


Рисунок 3.9 – Дифференциальное включение ОУ

Поскольку напряжение между инвертирующим входом и общей шиной $V_n – V_p$, ток I_1 определяется соотношением [10]:

$$I_1 = (V_2 – V_p) / R_1.$$

Тогда $I_1 = I_3 = \frac{Ua - U_-}{R1}$, и $U_+ = U_-$.

Выходное напряжение определим так:

$U_{out} = U_- — R3 * I3 = U_- - \frac{R3}{R1}\ (Ua - U_-)$;

$U_{out} = 0{,}5 * Ua + 1{,}25 - \frac{R3}{R1}\ (Ua - 0{,}5 * Ua - 1{,}25)$;

Т.к. $R1 = R$, $R3$ переменный терморезистор, то примем его значение так:

$R3 = R + \Delta R$;

$U_{out} = 0{,}5 * Ua + 1{,}25 - \frac{R+\Delta R}{R}\ (0{,}5 * Ua - 1{,}25)$;

$U_{out} = 1{,}25 + 1{,}25 - \frac{\Delta R}{R}\ (0{,}5 * Ua - 1{,}25) = 2{,}5 - \frac{\Delta R}{R}\ (0{,}5 * Ua - 1{,}25)$.

Выходное напряжение может быть в пределах от 0 до 5В. Необходимо оцифровать разброс ΔR полностью, для этого было подобрано $Ua = 5$В.

Находим реальные значения в формате с плавающей точкой для температуры:

$U_{АЦП}^{t} = 2{,}5 - \frac{\Delta R}{R}(0{,}5 * 5 - 1{,}25)$;

$R = R_0 * (1 + \alpha * t)$;

$\Delta R = R - R_0 = R_0 + R_0 * \alpha * t - R_0 = R_0 * \alpha * t$;

Температурный коэффициент (ТК) = 3850 ppm/K;

Напряжение на входе АЦП:

$U_{АЦП} = 2{,}5 - 1{,}25 * \frac{R0*\alpha*t}{R0} = 2{,}5 - 1{,}25 * \alpha * t$;

$U_{АЦП} = \frac{УЦ*3{,}3}{4095}$, где $U_Ц$ – преобразованное напряжение после оцифровки;

$\frac{УЦ*3{,}3}{4095} = 2{,}5 - 1{,}25 * \alpha * t$;

$\alpha = 0{,}00385$;

$t = \frac{-\frac{3{,}3*УЦ}{4095} + 2{,}5}{1{,}25*\alpha}$;

$t = \frac{-3{,}3*Uц + 10237{,}5}{4095*1{,}25*\alpha}$;

$t = \frac{10237{,}5 - 3{,}3*Uц}{19{,}707}$;

$t = 519 - 0{,}167 * U_ц$.

Датчик температуры в схеме электрической цепи представлен в следующем виде (рисунок 3.10).

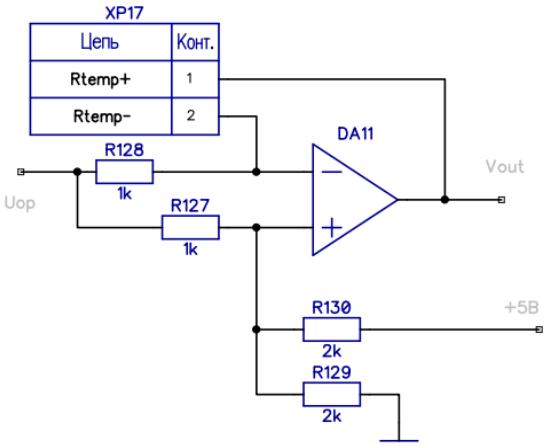


Рисунок 3.10 – Элемент датчика температуры в схеме электрической цепи

Также в проекте следует использовать аналоговые датчики тока, которые будут установлены в каждой фазе симметричной системы питания электродвигателя.

Для измерения силы как переменного, так и постоянный тока в цепи, были взяты аналоговые датчики ACS712-30A, с допустимыми пределами 0 – 30 А. Внешний вид модуля изображён на рисунке 3.11.

Рисунок 3.11 – Внешний вид датчика ACS712

Находим реальные значения в формате с плавающей точкой для тока:

$U_{Ц}{}^{I} = U_{АЦП}{}^{I} * \frac{4095}{3,3}$; $U_{АЦП}{}^{I} = \frac{Uц * 3,3}{4095}$;

$I = (U_{АЦП}{}^{I} * \frac{3}{2} - 2,5) * \frac{30}{2}$;

$I = (\frac{UЦ * 3,3}{4095} * \frac{3}{2} - 2,5) * 15$;

$I = \frac{UЦ * 3,3}{2730} * 15 - 37,5$;

$I = U_{ц} * 0,0181 - 6825$.

Датчик тока в схеме электрической цепи представлен в следующем виде (рисунок 3.12).

Цепь	Конт.
+5В	1
0V	2
U_dt_3	3

Рисунок 2.12 – Элемент датчика тока в схеме электрической цепи

Далее был взят датчик – магнитоэлектрическое устройство серии SS49E, использующее эффект Холла (рисунок 3.13). Суть его работы в следующем: если проводник с током помещён в магнитное поле, на его краях возникает ЭДС, которая направлена перпендикулярно к направлению тока и направлению магнитного поля.

Рисунок 3.13 – Внешний вид датчика S49402

Находим реальные значения в формате с плавающей точкой для магнитного поля:

0 Гс – 2,5 В; 420 Гс – 4 В

U (Гс) = 2,5 + β*Гс; 4 = 2,5 + β*420;

$\beta = \frac{1,5}{420} = 0,00357$ В/Гс;

$U_{АЦП} = 2,5 + 0,00357$ В (индукция);

$U_Ц = \frac{2}{3} * U_{АЦП} * \frac{4095}{3,3}$; $U_{АЦП} = \frac{UЦ * 3,3 * 3}{4095*2}$;

$B = \frac{\frac{9,9 * UЦ}{8190} - 2,5}{0,00357} = \frac{0,0012087 * Uц - 2,5}{0,00357}$;

B = 0,339 * Uц – 700,2.

Датчик магнитного поля в схеме электрической цепи представлен в следующем виде (рисунок 3.14).

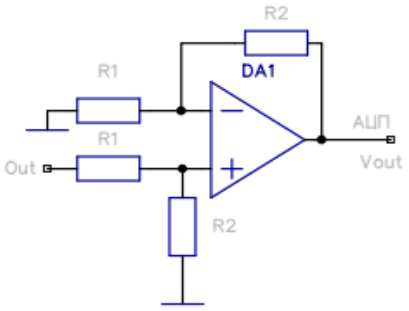


Рисунок 3.14 – Элемент датчика магнитного поля в схеме электрической цепи

3.4.3 Подбор контроллера ENC28J60

Контроллер ENC28J60 – это высокопроизводительный контроллер Ethernet (рисунок 3.15). Он обеспечивает полный доступ к сети Ethernet посредством SPI, а также управление физической средой передачи данных. Также он имеет небольшой размер и потребляет мало энергии.

С помощью данного контроллера можно реализовать передачу данных в режиме реального времени, или автоматизацию производственного процесса.

Рисунок 3.15 – Внешний вид контроллера ENC28J60

Контроллер в схеме электрической цепи представлен в следующем виде (рисунок 3.16).

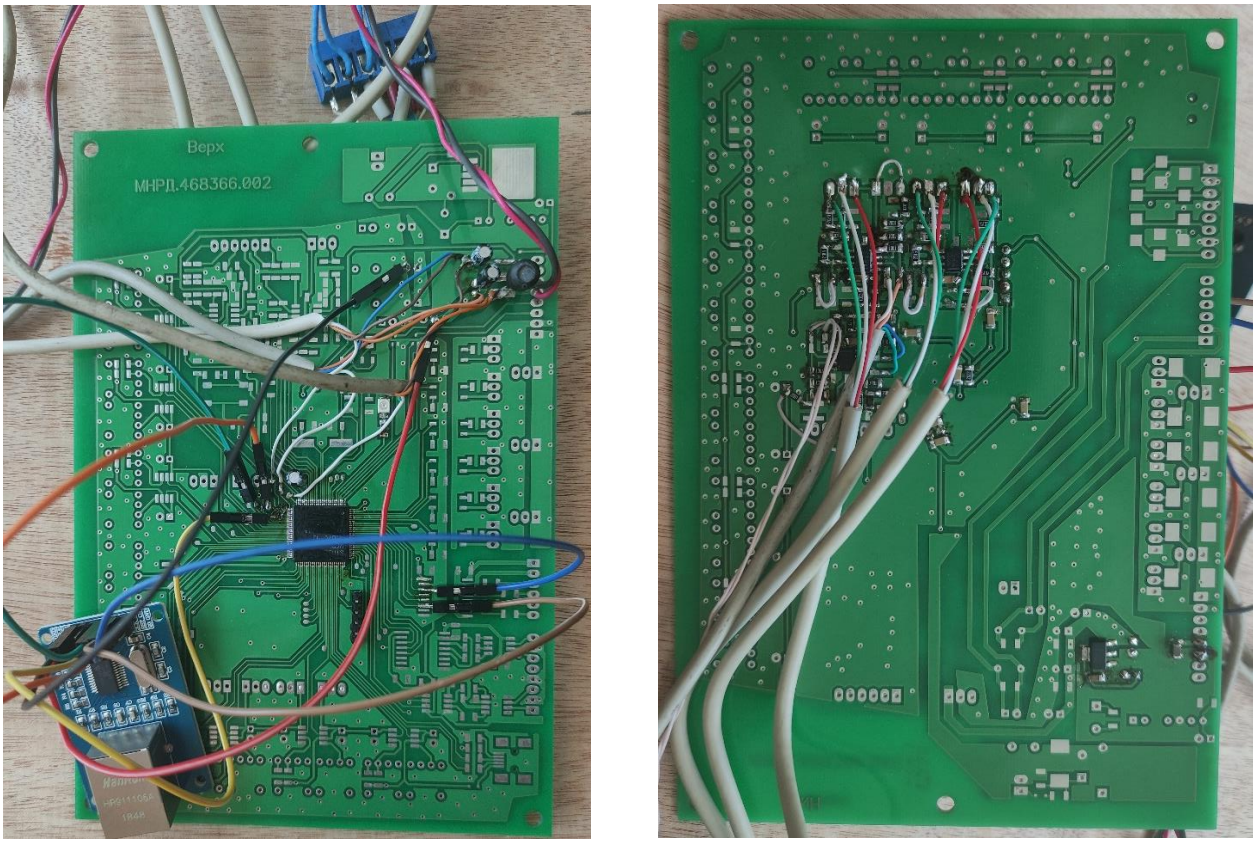
Цепь	Конт.
+5V	1
GND	2
SO	3
SCK	4
SI	5
RST	6
CS	7
ST	8

Рисунок 3.16 – Элемент контроллера в схеме электрической цепи

4 РЕАЛИЗАЦИЯ

4.1 Сборка макета

Для проверки работоспособности проектируемого комплекса был собран макет из вышеописанных компонентов (рисунок 4.1).



а) б)

Рисунок 4.1 – Макет: а – лицевая сторона; б – тыльная сторона

На рисунке 4.2 представлена конечная сборка всех компонентов в составе с двигателем ДБУ-210.

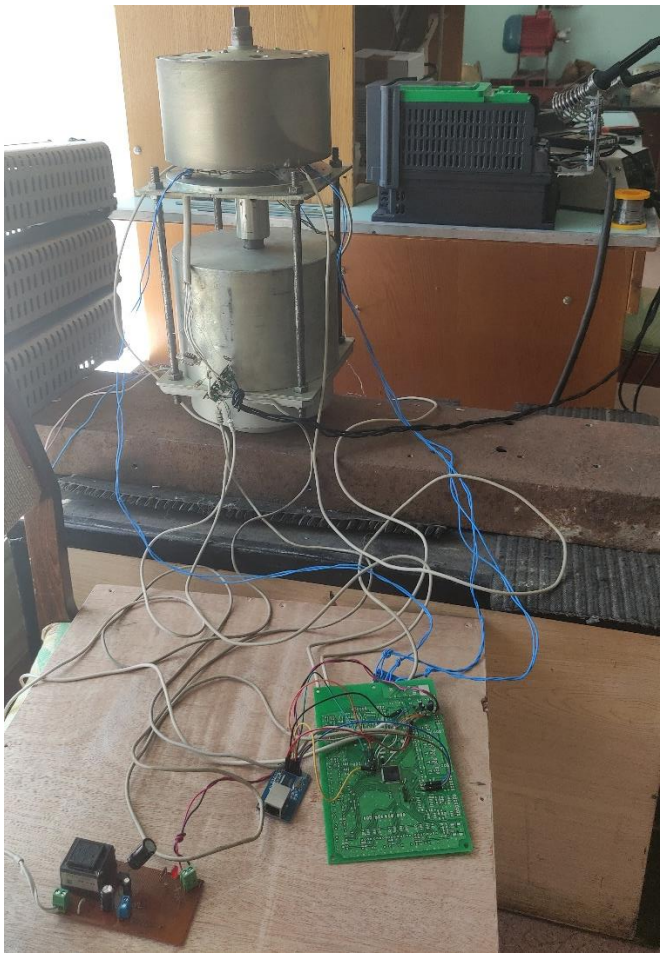


Рисунок 4.2 – Конечное представление комплекса

4.2 Реализация передачи данных из микроконтроллера по SPI

Рассмотрим подключение контроллера ENC28J60 и платы микроконтроллера STM32F303VCT6 по интерфейсу SPI. А также настройки и подключение данного контроллера, прием и передача данных.

4.2.1 Настройка и подключение ENC28J60

Как было сказано ранее, в качестве платформы был взят микроконтроллер STM32F303VCT6, а в качестве микросхемы физического уровня Ethernet – контроллер ENC28J60.

Контроллер ENC28J60 использует типовой механизм взаимодействия с внешним устройством – при помощи чтения и записи значений в регистры, отвечающие за определенные функции.

Работа с ENC28J60 реализовывается следующим образом – контроллер отправляет один байт, за которым следуют данные. От команды зависит, что за ней следует: передача данных в ENC28J60, либо чтение данных из нее же.

Прежде всего рассмотрим физическое подключение, инициализацию интерфейсов и настройку ENC28J60.

Подключение контроллера производится по интерфейсу SPI. Для подключения микроконтроллера был выбран графический генератор кода STM32CubeMx и HAL. Выводы для работы с ENC28J60 – сигналы Chip Select, Reset и SPI1 представлены на рисунке 4.3.

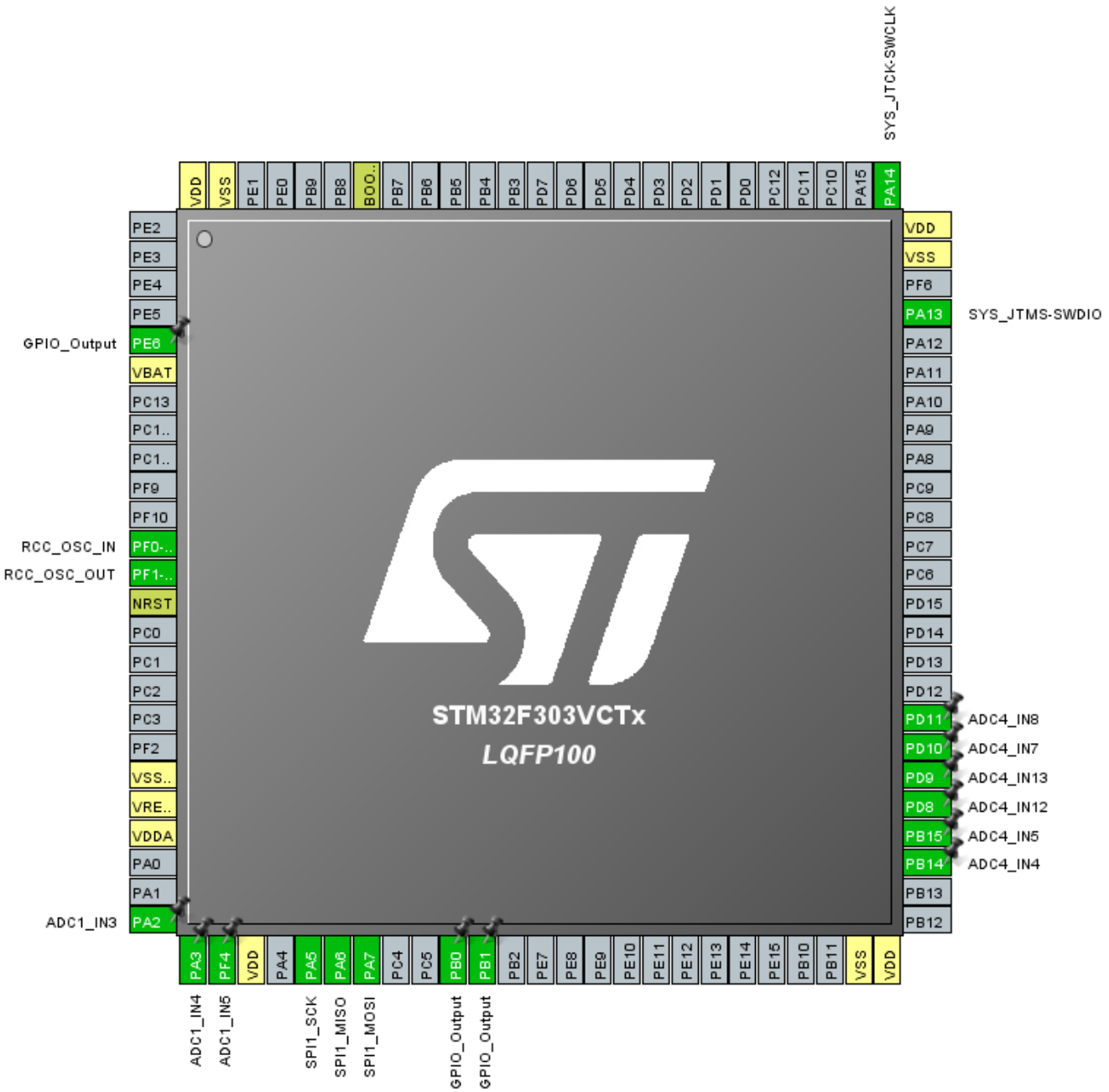


Рисунок 4.3 – Выводы

В работе используется внешний кварцевый резонатор на 8 МГц. На рисунке 4.4 представлено окно тактирования.

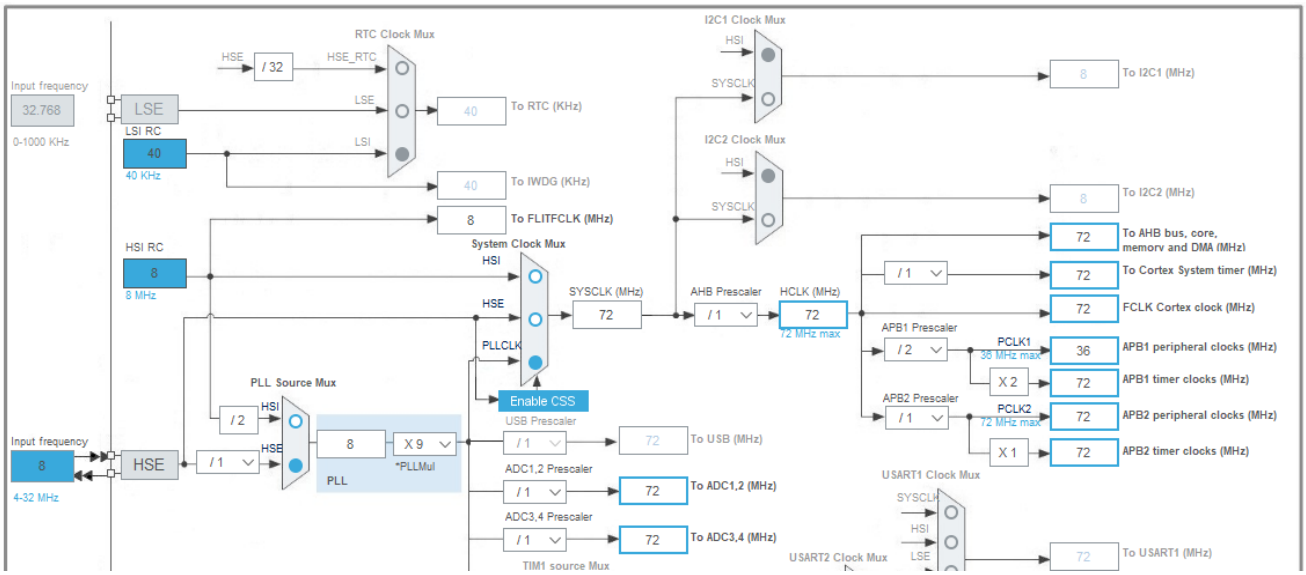


Рисунок 4.4 – Окно тактирования

Следующий этап – настройка SPI. Как изображено на рисунке 18, используем ножки PA5 – SPI SCK, PA6 – SPI MISO и PA7 – SPI MOSI. ENC28J60 настраиваем следующим образом, как показано на рисунке 4.5.

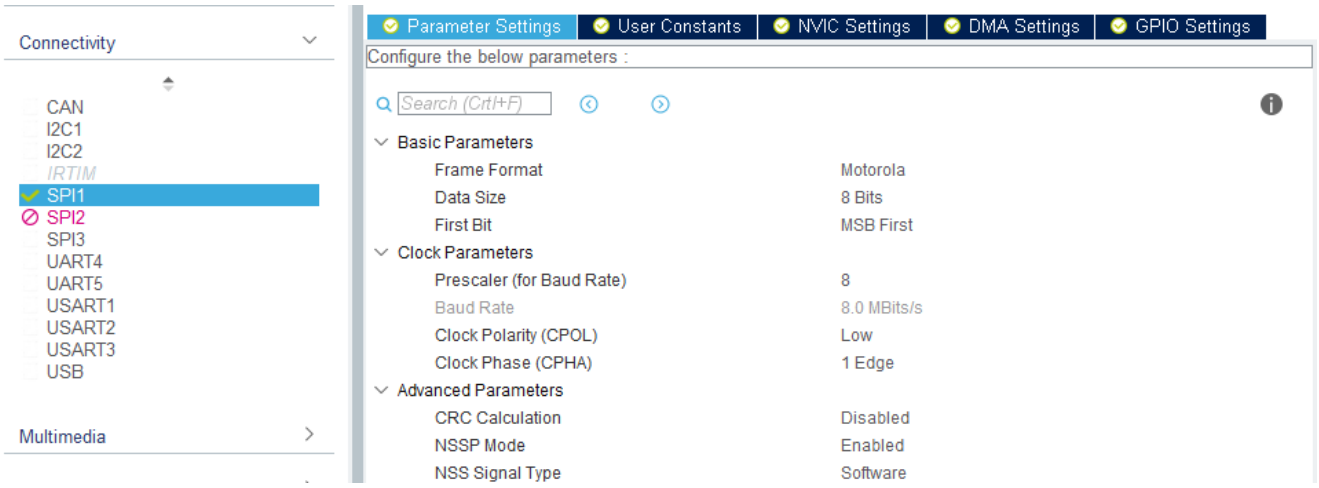


Рисунок 4.5 – Настройка SPI

Инициализация и работа с ENC28J60 представлена в разделе «ПРИЛОЖЕНИЯ. Листинг».

4.2.2 Прием и передача

Прием и передача данных в ENC28J60 реализовывается с помощью кольцевых буферов, они же в свою очередь расположены в RAM-памяти.

Для осуществления таких целей имеется порядка 8 КБ памяти (рисунок 4.6).

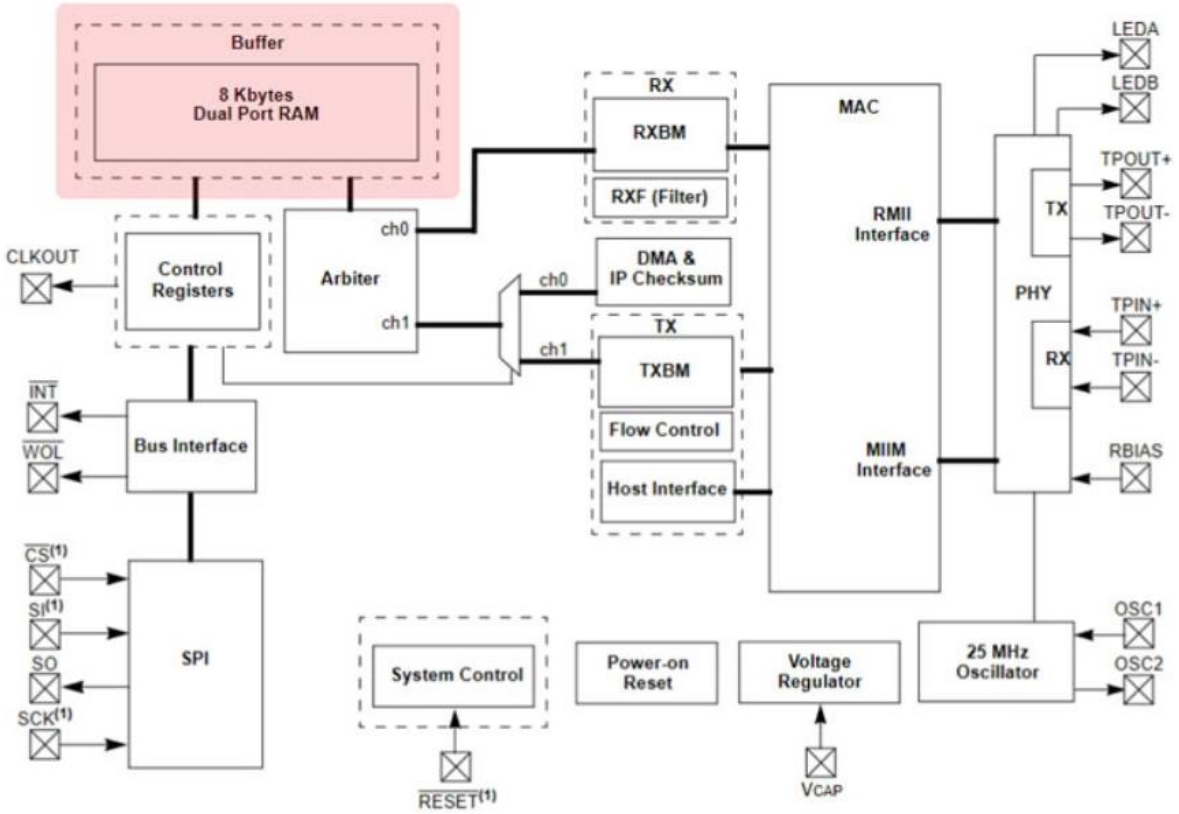


Рисунок 4.6 – RAM-память

При передаче данных адреса начального и конечного байтов конкретного кадра Ethernet, который собираемся передавать, записываются в регистры.

Но ENC28J60 в свою очередь не проверяет адреса, которые задаются в регистры с целью перекрывания адресов буфера, – это задача управляющего контроллера.

Для отправки кадра выполняется следующий набор действий:

1. Происходит запись в регистр свободного адреса памяти, куда следующим шагом поместятся данные передаваемого кадра.
2. Данные в буфер помещаются с помощью команды Write Buffer Memory.
3. В регистр записывается конечный адрес, который соответствует размеру передаваемых данных.
4. Далее запускается процесс установкой бита.

Далее в память помещаются отправляемые данные кадра Ethernet, в которые входят заголовок кадра (14 байт) и 72 байта данных. В конце указывается набор байтов статуса (4 байта), их модуль добавляет самостоятельно.

И в заключении необходимо предоставить информацию модулю ENC28J60 о получении пакета. Для этого записываем "1" в бит *PKTDEC* регистра ECON2 (рисунок 4.7).

```
1.   BitFieldSet(ECON2, ECON2_PKTDEC_BIT);
```

Рисунок 4.7 – Получение пакета

5 ТЕСТИРОВАНИЕ

В рамках реализации проекта производится рассмотрение пользовательских ошибок при работе с ПАК (сквозное тестирование), тестирование реализованной системы на соответствие функциональным требованиям. Затем организовывается тестовая передача данных в программную среду Twin Builder. После установления соединения с web-сервером начинается процесс передачи (рисунок 5.1). На графике мы видим, что переданные значения с клиента на сервер совпадают, это означает, что реализация проекта достигла поставленной цели.

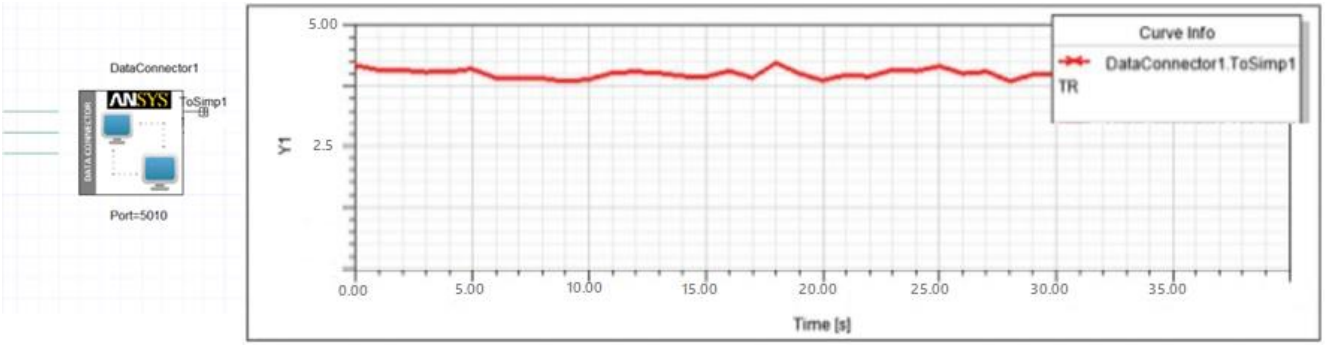


Рисунок 5.1 – Результат тестирования

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы был спроектирован и реализован комплекс приёмопередатчика, готовый для передачи данных в ПК Ansys Twin Builder.

По ходу работы были решены следующие задачи:

- проанализирована предметная область на наличие существующих аналогов;
- выявлены функциональные и нефункциональные требования к разрабатываемому комплексу;
- спроектирована архитектура программного комплекса;
- реализован программно-аппаратный комплекс;
- произведена тестовая передача данных в ПК Ansys Twin Builder.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Цифровые двойники. Дизайн через отражение. / [Электронные ресурс] // habr.com: интернет-портал. – Режим доступа: https://habr.com/ru/articles/331562/ (дата обращение: 20.02.2023).

2 TCP/IP. / [Электронный ресурс] // nic.ru: интернет-портал. – Режим доступа: https://www.nic.ru/help/chto-takoe-tcpip_11168.html/ (дата обращения: 20.02.2023).

3 Цифровые двойники в промышленности. Возможности и перспективы. / [Электронный ресурс] // ritm-magazine.com: интернет-портал. – Режим доступа: https://ritm-magazine.com/en/node/8663/ (дата обращения: 20.02.2023).

4 Цифровые двойники в строительстве: практика использования. / [Электронный ресурс] // sodislab.com: интернет-портал. – Режим доступа: https://www.sodislab.com/ru/blog/digitaltwins/ (дата обращения: 20.02.2023).

5 Частотные преобразователи. / [Электронный ресурс] // invt.su: интернет-портал. – Режим доступа: https://www.invt.su/o-kompanii/stati/primenenie-preobrazovatelej-chastoty.html/ (дата обращения: 20.02.2023).

6 Панкратов, В.В. ВЕНТИЛЬНЫЙ ЭЛЕКТРОПРИВОД: ОТ СТИРАЛЬНОЙ МАШИНЫ ДО МЕТАЛЛОРЕЖУЩЕГО СТАНКА И ЭЛЕКТРОВОЗА // Электронные компоненты. – 2007. – № 2. – С. 1-13.

7 Цифровой двойник морского двигателя. / [Электронный ресурс] // rostec.ru: интернет-портал. – Режим доступа: https://rostec.ru/news/odk-zavershila-pervyy-etap-sozdaniya-tsifrovogo-dvoynika-morskogo-dvigatelya/ (дата обращения: 20.02.2023).

8 Электротехнические и информационные комплексы и системы. / [Электронный ресурс] // ngdelo.ru: интернет-портал. – Режим доступа: http://ngdelo.ru/index.php/ele/article/view/11458/ (дата обращения: 20.02.2023).

9 Вторая версия цифрового двойника насоса с двигателем. / [Электронный ресурс] // ansys.soften: интернет-портал. – Режим доступа: https://www.ansys.soften.com.ua/about-ansys/blog/303-vtoraya-versiya-tsifrovogo-dvojnika-nasosa-s-dvigatelem.html / (дата обращения: 20.02.2023).

10 Волович, Г. И. Схемотехника аналоговых и аналого-цифровых электронных устройств: учебное пособие / Г. И. Волович. – Москва: Изд. Дом «Додэка-XXI», 2005. – 530 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Листинг

MAIN.C

```
#include "main.h"

DataTX_TypeDef   dataTX;

/* Private variables ---------------------------------------------------------*/
SPI_HandleTypeDef hspi1;

unsigned short adc1_DTok[4], adc4_DTemp[7];

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_ADC4_Init(void);
static void MX_SPI1_Init(void);

int main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  MX_ADC4_Init();
  MX_SPI1_Init();

// Выполним процедуру калибровки АЦП1
// Включим регулятор напряжения
LL_ADC_EnableInternalRegulator(ADC1);
// Подождем 10 uS
for (uint32_t i = 720; i != 0; i--); // Delay(10);
// Выберем режим калибровки для АЦП1 и после запустим АПЦ1
LL_ADC_StartCalibration(ADC1, LL_ADC_SINGLE_ENDED); //
// Ожидаем завершения калибровки АЦП1
while (LL_ADC_IsCalibrationOnGoing(ADC1) != RESET);
adc1_DTok[3] = LL_ADC_GetCalibrationFactor(ADC1, LL_ADC_SINGLE_ENDED);
```

```
// Выполним процедуру калибровки АЦП4
// Включим регулятор напряжения
LL_ADC_EnableInternalRegulator(ADC4);
// Подождем 10 uS
for (uint32_t i = 720; i != 0; i--); // Delay(10);
// Выберем режим калибровки для АЦП4 и после запустим АПЦ4
LL_ADC_StartCalibration(ADC4, LL_ADC_SINGLE_ENDED); //
// Ожидаем завершения калибровки АЦП4
while (LL_ADC_IsCalibrationOnGoing(ADC4) != RESET);
adc4_DTemp[6] = LL_ADC_GetCalibrationFactor(ADC4, LL_ADC_SINGLE_ENDED);

// Настройка 1 канала DMA1  - для обслуживания ADC1
// Укажем размер массива для результатов преобразования
LL_DMA_SetDataLength(DMA1, LL_DMA_CHANNEL_1, 0x03);
// Укажем адрес массива в ОЗУ (для результатов преобразования)
LL_DMA_SetMemoryAddress(DMA1, LL_DMA_CHANNEL_1, (uint32_t) &adc1_DTok);
// Укажем адрес периферийного устройства (УВВ) т.е. АЦП
LL_DMA_SetPeriphAddress(DMA1, LL_DMA_CHANNEL_1, (uint32_t) &ADC1->DR);

// Настройка 2 канала DMA2  - для обслуживания ADC4
// Укажем размер массива для результатов преобразования
LL_DMA_SetDataLength(DMA2, LL_DMA_CHANNEL_2, 0x06);
// Укажем адрес массива в ОЗУ (для результатов преобразования)
LL_DMA_SetMemoryAddress(DMA2, LL_DMA_CHANNEL_2, (uint32_t) &adc4_DTemp);
// Укажем адрес периферийного устройства (УВВ) т.е. АЦП
LL_DMA_SetPeriphAddress(DMA2, LL_DMA_CHANNEL_2, (uint32_t) &ADC4->DR);

// Включаем канал контроллера ПДП для опроса АЦП1
LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_1);
// Разрешаем прерывание по завершению преобразования
LL_DMA_EnableIT_TC(DMA1, LL_DMA_CHANNEL_1);

// Включаем канал контроллера ПДП для опроса АЦП4
LL_DMA_EnableChannel(DMA2, LL_DMA_CHANNEL_2);
// Разрешаем прерывание по завершению преобразования
LL_DMA_EnableIT_TC(DMA2, LL_DMA_CHANNEL_2);

// Включаем счетчик таймера 3
LL_TIM_EnableCounter(TIM3);

// Включаем АЦП1
LL_ADC_Enable(ADC1); // включение АЦП1
// Ожидаем флаг готовности АЦП1
while (!LL_ADC_IsActiveFlag_ADRDY(ADC1));

// Включаем АЦП4
LL_ADC_Enable(ADC4); // включение АЦП4
// Ожидаем флаг готовности АЦП4
while (!LL_ADC_IsActiveFlag_ADRDY(ADC4));
// Включение светодиода для отображения рабочего состояния
LL_GPIO_SetOutputPin(GPIOE, LL_GPIO_PIN_6);
```

```
LL_ADC_REG_StartConversion(ADC1); // запуск преобразования АЦП1
LL_ADC_REG_StartConversion(ADC4); // запуск преобразования АЦП4

ENC28J60_Init();

dataTX.MAC11=0x0017;
dataTX.MAC12=0x22ED;
dataTX.MAC13=0xA501;

void SystemClock_Config(void)
{
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_2);

   if(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_2)
  {
  Error_Handler();
  }
  LL_RCC_HSE_Enable();

   /* Wait till HSE is ready */
  while(LL_RCC_HSE_IsReady() != 1)
  {

  }
  LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE_DIV_1,
LL_RCC_PLL_MUL_9);
  LL_RCC_PLL_Enable();

   /* Wait till PLL is ready */
  while(LL_RCC_PLL_IsReady() != 1)
  {

  }
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_2);
  LL_RCC_SetAPB2Prescaler(LL_RCC_APB1_DIV_1);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);

   /* Wait till System clock is ready */
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
  {

  }
  LL_Init1msTick(72000000);
  LL_SYSTICK_SetClkSource(LL_SYSTICK_CLKSOURCE_HCLK);
  LL_SetSystemCoreClock(72000000);
  LL_RCC_SetADCClockSource(LL_RCC_ADC12_CLKSRC_PLL_DIV_1);
  LL_RCC_SetADCClockSource(LL_RCC_ADC34_CLKSRC_PLL_DIV_1);
}
```

```
static void MX_ADC1_Init(void)
{
  LL_ADC_InitTypeDef ADC_InitStruct = {0};
  LL_ADC_REG_InitTypeDef ADC_REG_InitStruct = {0};
  LL_ADC_CommonInitTypeDef ADC_CommonInitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_ADC12);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);

  GPIO_InitStruct.Pin = LL_GPIO_PIN_2|LL_GPIO_PIN_3;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = LL_GPIO_PIN_4;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOF, &GPIO_InitStruct);

    LL_DMA_SetDataTransferDirection(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_DIRECTION_PERIPH_TO_MEMORY);

  LL_DMA_SetChannelPriorityLevel(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_PRIORITY_LOW);

  LL_DMA_SetMode(DMA1, LL_DMA_CHANNEL_1, LL_DMA_MODE_CIRCULAR);

  LL_DMA_SetPeriphIncMode(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_PERIPH_NOINCREMENT);

  LL_DMA_SetMemoryIncMode(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_MEMORY_INCREMENT);

  LL_DMA_SetPeriphSize(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_PDATAALIGN_HALFWORD);

  LL_DMA_SetMemorySize(DMA1, LL_DMA_CHANNEL_1,
LL_DMA_MDATAALIGN_HALFWORD);

  ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B;
  ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
  ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
  LL_ADC_Init(ADC1, &ADC_InitStruct);
  ADC_REG_InitStruct.TriggerSource = LL_ADC_REG_TRIG_EXT_TIM3_TRGO_ADC12;
  ADC_REG_InitStruct.SequencerLength = LL_ADC_REG_SEQ_SCAN_ENABLE_3RANKS;
```

```
  ADC_REG_InitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_3RANKS;
  ADC_REG_InitStruct.ContinuousMode = LL_ADC_REG_CONV_SINGLE;
  ADC_REG_InitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_UNLIMITED;
  ADC_REG_InitStruct.Overrun = LL_ADC_REG_OVR_DATA_OVERWRITTEN;
  LL_ADC_REG_Init(ADC1, &ADC_REG_InitStruct);
  LL_ADC_DisableIT_EOC(ADC1);
  LL_ADC_DisableIT_EOS(ADC1);
  ADC_CommonInitStruct.CommonClock = LL_ADC_CLOCK_ASYNC_DIV1;
  ADC_CommonInitStruct.Multimode = LL_ADC_MULTI_INDEPENDENT;
  LL_ADC_CommonInit(__LL_ADC_COMMON_INSTANCE(ADC1), &ADC_CommonInitStruct);
  LL_ADC_REG_SetTriggerEdge(ADC1, LL_ADC_REG_TRIG_EXT_RISING);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_3);
  LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_3,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC1, LL_ADC_CHANNEL_3, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_2, LL_ADC_CHANNEL_4);
  LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_4,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC1, LL_ADC_CHANNEL_4, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC1, LL_ADC_REG_RANK_3, LL_ADC_CHANNEL_5);
  LL_ADC_SetChannelSamplingTime(ADC1, LL_ADC_CHANNEL_5,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC1, LL_ADC_CHANNEL_5, LL_ADC_SINGLE_ENDED);

}
static void MX_ADC4_Init(void)
{

  LL_ADC_InitTypeDef ADC_InitStruct = {0};
  LL_ADC_REG_InitTypeDef ADC_REG_InitStruct = {0};

  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* Peripheral clock enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_ADC34);

  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);
  /**ADC4 GPIO Configuration

  GPIO_InitStruct.Pin = LL_GPIO_PIN_14|LL_GPIO_PIN_15;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
  GPIO_InitStruct.Pin = LL_GPIO_PIN_8|LL_GPIO_PIN_9|LL_GPIO_PIN_10|LL_GPIO_PIN_11;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  LL_DMA_SetDataTransferDirection(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_DIRECTION_PERIPH_TO_MEMORY);

  LL_DMA_SetChannelPriorityLevel(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_PRIORITY_LOW);

  LL_DMA_SetMode(DMA2, LL_DMA_CHANNEL_2, LL_DMA_MODE_CIRCULAR);

  LL_DMA_SetPeriphIncMode(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_PERIPH_NOINCREMENT);

  LL_DMA_SetMemoryIncMode(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_MEMORY_INCREMENT);

  LL_DMA_SetPeriphSize(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_PDATAALIGN_HALFWORD);

  LL_DMA_SetMemorySize(DMA2, LL_DMA_CHANNEL_2,
LL_DMA_MDATAALIGN_HALFWORD);

  ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B;
  ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
  ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
  LL_ADC_Init(ADC4, &ADC_InitStruct);
  ADC_REG_InitStruct.TriggerSource = LL_ADC_REG_TRIG_EXT_TIM3_TRGO__ADC34;
  ADC_REG_InitStruct.SequencerLength = LL_ADC_REG_SEQ_SCAN_ENABLE_6RANKS;
  ADC_REG_InitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_6RANKS;
  ADC_REG_InitStruct.ContinuousMode = LL_ADC_REG_CONV_SINGLE;
  ADC_REG_InitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_UNLIMITED;
  ADC_REG_InitStruct.Overrun = LL_ADC_REG_OVR_DATA_OVERWRITTEN;
  LL_ADC_REG_Init(ADC4, &ADC_REG_InitStruct);
  LL_ADC_DisableIT_EOC(ADC4);
  LL_ADC_DisableIT_EOS(ADC4);
  LL_ADC_SetCommonClock(__LL_ADC_COMMON_INSTANCE(ADC4),
LL_ADC_CLOCK_ASYNC_DIV1);
  LL_ADC_REG_SetTriggerEdge(ADC4, LL_ADC_REG_TRIG_EXT_RISING);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_4);
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_4,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_4, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_2, LL_ADC_CHANNEL_5);
```

```
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_5,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_5, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_3, LL_ADC_CHANNEL_13);
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_13,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_13, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_4, LL_ADC_CHANNEL_12);
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_12,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_12, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_5, LL_ADC_CHANNEL_7);
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_7,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_7, LL_ADC_SINGLE_ENDED);
  /**Configure Regular Channel
  */
  LL_ADC_REG_SetSequencerRanks(ADC4, LL_ADC_REG_RANK_6, LL_ADC_CHANNEL_8);
  LL_ADC_SetChannelSamplingTime(ADC4, LL_ADC_CHANNEL_8,
LL_ADC_SAMPLINGTIME_19CYCLES_5);
  LL_ADC_SetChannelSingleDiff(ADC4, LL_ADC_CHANNEL_8, LL_ADC_SINGLE_ENDED);
}

static void MX_SPI1_Init(void)
{

  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
```

```
  }

}
static void MX_TIM3_Init(void)
{

  LL_TIM_InitTypeDef TIM_InitStruct = {0};

  /* Peripheral clock enable */
  LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_TIM3);

  TIM_InitStruct.Prescaler = 63;
  TIM_InitStruct.CounterMode = LL_TIM_COUNTERMODE_UP;
  TIM_InitStruct.Autoreload = 10000;
  TIM_InitStruct.ClockDivision = LL_TIM_CLOCKDIVISION_DIV1;
  LL_TIM_Init(TIM3, &TIM_InitStruct);
  LL_TIM_DisableARRPreload(TIM3);
  LL_TIM_SetClockSource(TIM3, LL_TIM_CLOCKSOURCE_INTERNAL);
  LL_TIM_SetTriggerOutput(TIM3, LL_TIM_TRGO_UPDATE);
  LL_TIM_DisableMasterSlaveMode(TIM3);
}

static void MX_DMA_Init(void)
{
  /* Init with LL driver */
  /* DMA controller clock enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA2);

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  NVIC_SetPriority(DMA1_Channel1_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,
0));
  NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  /* DMA2_Channel2_IRQn interrupt configuration */
  NVIC_SetPriority(DMA2_Channel2_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,
0));
  NVIC_EnableIRQ(DMA2_Channel2_IRQn);

}
static void MX_GPIO_Init(void)
{
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOE);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOC);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
```

```
 LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOD);

 /**/
 LL_GPIO_ResetOutputPin(GPIOE, LL_GPIO_PIN_6);

 /**/
 LL_GPIO_ResetOutputPin(RST_GPIO_Port, RST_Pin);

 /**/
 LL_GPIO_ResetOutputPin(CS_GPIO_Port, CS_Pin);

 /**/
 LL_GPIO_ResetOutputPin(GPIOB, LL_GPIO_PIN_0|LL_GPIO_PIN_1);

 /**/
 GPIO_InitStruct.Pin = LL_GPIO_PIN_6;
 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
 LL_GPIO_Init(GPIOE, &GPIO_InitStruct);

 /**/
 GPIO_InitStruct.Pin = RST_Pin;
 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
 LL_GPIO_Init(RST_GPIO_Port, &GPIO_InitStruct);

 /**/
 GPIO_InitStruct.Pin = CS_Pin;
 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
 LL_GPIO_Init(CS_GPIO_Port, &GPIO_InitStruct);

 /**/
 GPIO_InitStruct.Pin = LL_GPIO_PIN_0|LL_GPIO_PIN_1;
 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
 LL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
#ifdef  USE_FULL_ASSERT
void assert_failed(char *file, uint32_t line)
}
```

MAIN.H

```
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32f3xx_hal.h"
#include "stm32f3xx_ll_adc.h"
#include "stm32f3xx_ll_dma.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_exti.h"
#include "stm32f3xx_ll_cortex.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_pwr.h"
#include "stm32f3xx_ll_tim.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"

typedef union{
unsigned short single [45];
unsigned char Half [90];
struct {
short MAC11;
short MAC12;
short MAC13;
short MAC21;
short MAC22;
short MAC23;
short serv_1;
double Temp1;
double Temp2;
double Temp3;
double Tok1;
double Tok2;
double Tok3;
double MF1;
double MF2;
double MF3;
int CRC32_1;
};
} DataTX_TypeDef;

void Error_Handler(void);

/* Private defines -----------------------------------------------------------*/
#define RST_Pin LL_GPIO_PIN_4
```

```
#define RST_GPIO_Port GPIOA
#define CS_Pin LL_GPIO_PIN_4
#define CS_GPIO_Port GPIOC

#ifdef __cplusplus
}
#endif
```

STM32F3XX_IT.C

```
#include "main.h"
#include "stm32f3xx_it.h"
extern unsigned short adc1_DTok[3], adc4_DTemp[3], adc_DMF[3];
extern DataTX_TypeDef   dataTX;
//extern unsigned char dataTX[90];
void SysTick_Handler(void)
{
  HAL_IncTick();
}
void DMA1_Channel1_IRQHandler(void)
{
if (LL_DMA_IsActiveFlag_TC1(DMA1) != RESET)
{ LL_DMA_ClearFlag_TC1(DMA1);


dataTX.Tok1=adc1_DTok[0]*0,0181-6825;

dataTX.Tok2=adc1_DTok[1]*0,0181-6825;

dataTX.Tok3=adc1_DTok[2]*0,0181-6825;

dataTX.Temp1=519-0,167*adc4_DTemp[0];

dataTX.Temp1=519-0,167*adc4_DTemp[1];

dataTX.Temp1=519-0,167*adc4_DTemp[2];

dataTX.MF1=0,339*adc_DMF[0]-700,2;

dataTX.MF1=0,339*adc_DMF[1]-700,2;

dataTX.MF1=0,339*adc_DMF[2]-700,2;

ENC28J60_TransmitFrame(&dataTX, 90);
}
}
void DMA2_Channel2_IRQHandler(void)
{
if (LL_DMA_IsActiveFlag_TC2(DMA2) != RESET)
{ LL_DMA_ClearFlag_TC2(DMA2);
}
}
```

ENC28J60.C

```
#include "enc28j60.h"
uint8_t macAddr[MAC_ADDRESS_BYTES_NUM] = {0x00, 0x17, 0x22, 0xED, 0xA5, 0x01};
static uint8_t commandOpCodes[COMMANDS_NUM] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05,
0x07};
static ENC28J60_RegBank curBank = BANK_0;
static uint16_t curPtr = ENC28J60_RX_BUF_START;
extern SPI_HandleTypeDef hspi1;
static void SetCS(ENC28J60_CS_State state)
{
HAL_GPIO_WritePin(ENC28J60_CS_PORT, ENC28J60_CS_PIN, (GPIO_PinState)state);
}
static void WriteBytes(uint8_t* data, uint16_t size)
{
HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, data, size, ENC28J60_SPI_TIMEOUT);
}
static void WriteByte(uint8_t data)
{
HAL_StatusTypeDef res = HAL_SPI_Transmit(&hspi1, &data, 1, ENC28J60_SPI_TIMEOUT);
}
static uint8_t ReadByte()
{
uint8_t txData = 0x00;
uint8_t rxData = 0x00;
HAL_StatusTypeDef res = HAL_SPI_TransmitReceive(&hspi1, &txData, &rxData, 1,
ENC28J60_SPI_TIMEOUT);
return rxData;
}
static ENC28J60_RegType getRegType(uint8_t reg)
{
ENC28J60_RegType type = (ENC28J60_RegType)((reg & ENC28J60_REG_TYPE_MASK) >>
ENC28J60_REG_TYPE_OFFSET);
return type;
}
static ENC28J60_RegBank getRegBank(uint8_t reg)
{
ENC28J60_RegBank bank = (ENC28J60_RegBank)((reg & ENC28J60_REG_BANK_MASK) >>
ENC28J60_REG_BANK_OFFSET);
return bank;
}
static uint8_t getRegAddr(uint8_t reg)
{
uint8_t addr = (reg & ENC28J60_REG_ADDR_MASK);
return addr;
}
static void WriteCommand(ENC28J60_Command command, uint8_t argData)
{
uint8_t data = 0;
data = (commandOpCodes[command] << ENC28J60_OP_CODE_OFFSET) | argData;
WriteByte(data);
```

```
}
static void CheckBank(uint8_t reg)
{
uint8_t regAddr = getRegAddr(reg);
if (regAddr < ENC28J60_COMMON_REGS_ADDR)
{
ENC28J60_RegBank regBank = getRegBank(reg);
if (curBank != regBank)
{
uint8_t econ1Addr = getRegAddr(ECON1);
// Clear bank bits
SetCS(CS_LOW);
WriteCommand(BIT_FIELD_CLEAR, econ1Addr);
WriteByte(ECON1_BSEL1_BIT | ECON1_BSEL0_BIT);
SetCS(CS_HIGH);
// Set bank bits
SetCS(CS_LOW);
WriteCommand(BIT_FIELD_SET, econ1Addr);
WriteByte(regBank);
SetCS(CS_HIGH);
curBank = regBank;
}
}
}
static void BitFieldSet(uint8_t reg, uint8_t regData)
{
uint8_t regAddr = getRegAddr(reg);
CheckBank(reg);
SetCS(CS_LOW);
WriteCommand(BIT_FIELD_SET, regAddr);
WriteByte(regData);
SetCS(CS_HIGH);
}
static void BitFieldClear(uint8_t reg, uint8_t regData)
{
uint8_t regAddr = getRegAddr(reg);
CheckBank(reg);
SetCS(CS_LOW);
WriteCommand(BIT_FIELD_CLEAR, regAddr);
WriteByte(regData);
SetCS(CS_HIGH);
}
static uint8_t ReadControlReg(uint8_t reg)
{
uint8_t data = 0;
ENC28J60_RegType regType = getRegType(reg);
uint8_t regAddr = getRegAddr(reg);
CheckBank(reg);
SetCS(CS_LOW);
WriteCommand(READ_CONTROL_REG, regAddr);
```

```
if (regType == MAC_MII_REG)
{
ReadByte();
}
data = ReadByte();
SetCS(CS_HIGH);
return data;
}
static void WriteControlReg(uint8_t reg, uint8_t regData)
{
uint8_t regAddr = getRegAddr(reg);
CheckBank(reg);
SetCS(CS_LOW);
WriteCommand(WRITE_CONTROL_REG, regAddr);
WriteByte(regData);
SetCS(CS_HIGH);
}
static void WriteControlRegPair(uint8_t reg, uint16_t regData)
{
WriteControlReg(reg, (uint8_t)regData);
WriteControlReg(reg + 1, (uint8_t)(regData >> 8));
}
static uint16_t ReadControlRegPair(uint8_t reg)
{
uint16_t data = 0;
data = (uint16_t)ReadControlReg(reg) | ((uint16_t)ReadControlReg(reg + 1) << 8);
return data;
}
static void WriteBufferMem(uint8_t *data, uint16_t size)
{
SetCS(CS_LOW);
WriteCommand(WRITE_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);
WriteBytes(data, size);
SetCS(CS_HIGH);
}
static void ReadBufferMem(uint8_t *data, uint16_t size)
{
SetCS(CS_LOW);
WriteCommand(READ_BUFFER_MEM, ENC28J60_BUF_COMMAND_ARG);
for (uint16_t i = 0; i < size; i++)
{
*data = ReadByte();
data++;
}
SetCS(CS_HIGH);
}
static void SystemReset()
{
SetCS(CS_LOW);
WriteCommand(SYSTEM_RESET, ENC28J60_RESET_COMMAND_ARG);
```

```
SetCS(CS_HIGH);
curBank = BANK_0;
HAL_Delay(100);
}
static uint16_t ReadPhyReg(uint8_t reg)
{
uint16_t data = 0;
uint8_t regAddr = getRegAddr(reg);
WriteControlReg(MIREGADR, regAddr);
BitFieldSet(MICMD, MICMD_MIIRD_BIT);
while((ReadControlReg(MISTAT) & MISTAT_BUSY_BIT) != 0);
BitFieldClear(MICMD, MICMD_MIIRD_BIT);
data = ReadControlRegPair(MIRDL);
return data;
}
void ENC28J60_StartReceiving()
{
BitFieldSet(ECON1, ECON1_RXEN_BIT);
}
static void WritePhyReg(uint8_t reg, uint16_t regData)
{
uint8_t regAddr = getRegAddr(reg);
WriteControlReg(MIREGADR, regAddr);
WriteControlRegPair(MIWRL, regData);
while((ReadControlReg(MISTAT) & MISTAT_BUSY_BIT) != 0);
}
void ENC28J60_Init()
{
HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_RESET);
HAL_Delay(50);
HAL_GPIO_WritePin(ENC28J60_RESET_PORT, ENC28J60_RESET_PIN, GPIO_PIN_SET);
HAL_Delay(50);
SystemReset();
// Rx/Tx buffers
WriteControlRegPair(ERXSTL, ENC28J60_RX_BUF_START);
WriteControlRegPair(ERXNDL, ENC28J60_RX_BUF_END);
WriteControlRegPair(ERDPTL, ENC28J60_RX_BUF_START);
// MAC address
WriteControlReg(MAADR1, macAddr[0]);
WriteControlReg(MAADR2, macAddr[1]);
WriteControlReg(MAADR3, macAddr[2]);
WriteControlReg(MAADR4, macAddr[3]);
WriteControlReg(MAADR5, macAddr[4]);
WriteControlReg(MAADR6, macAddr[5]);
WriteControlReg(MACON1, MACON1_TXPAUS_BIT | MACON1_RXPAUS_BIT |
MACON1_MARXEN_BIT);
WriteControlReg(MACON3, MACON3_PADCFG0_BIT | MACON3_TXCRCEN_BIT |
MACON3_FRMLNEN_BIT);
WriteControlRegPair(MAIPGL, ENC28J60_NBB_PACKET_GAP);
WriteControlReg(MABBIPG, ENC28J60_BB_PACKET_GAP);
```

```
WriteControlRegPair(MAMXFLL, ENC28J60_FRAME_DATA_MAX);
// PHY resisters
WritePhyReg(PHCON2, PHCON2_HDLDIS_BIT);
ENC28J60_StartReceiving();
}
uint16_t ENC28J60_ReceiveFrame(ENC28J60_Frame* frame)
{
uint16_t dataSize = 0;
uint8_t packetsNum = ReadControlReg(EPKTCNT);
if (packetsNum > 0)
{
WriteControlRegPair(ERDPTL, curPtr);
ReadBufferMem((uint8_t*)frame, ENC28J60_HEADER_SIZE);
curPtr = frame->nextPtr;
if ((frame->status & ENC28J60_FRAME_RX_OK_MASK) != 0)
{
dataSize = frame->length - ENC28J60_CRC_SIZE;
if (dataSize > ENC28J60_FRAME_DATA_MAX)
{
dataSize = ENC28J60_FRAME_DATA_MAX;
}
ReadBufferMem((uint8_t*)&(frame->data[0]), dataSize);
ReadBufferMem((uint8_t*)&(frame->checkSum), ENC28J60_CRC_SIZE);
}
uint16_t nextPtr = frame->nextPtr - 1;
if (nextPtr > ENC28J60_RX_BUF_END)
{
nextPtr = ENC28J60_RX_BUF_END;
}
WriteControlRegPair(ERXRDPTL, nextPtr);
BitFieldSet(ECON2, ECON2_PKTDEC_BIT);
}
return dataSize;
}
void ENC28J60_TransmitFrame(uint8_t *data, uint16_t size)
{
while((ReadControlReg(ECON1) & ECON1_TXRTS_BIT) != 0)
{
if((ReadControlReg(EIR) & EIR_TXERIF_BIT) != 0)
{
BitFieldSet(ECON1, ECON1_TXRST_BIT);
BitFieldClear(ECON1, ECON1_TXRST_BIT);
}
}
WriteControlRegPair(EWRPTL, ENC28J60_TX_BUF_START);
uint8_t controlByte = 0x00;
WriteBufferMem(&controlByte, 1);
WriteBufferMem(data, size);
WriteControlRegPair(ETXSTL, ENC28J60_TX_BUF_START);
WriteControlRegPair(ETXNDL, ENC28J60_TX_BUF_START + size);
```

```
BitFieldSet(ECON1, ECON1_TXRTS_BIT);
}
```

ENC28J60.H

```
#include "stm32f1xx_hal.h"
#define ENC28J60_CS_PORT GPIOB
#define ENC28J60_CS_PIN GPIO_PIN_3
#define ENC28J60_RESET_PORT GPIOB
#define ENC28J60_RESET_PIN GPIO_PIN_6
#define MAC_ADDRESS_BYTES_NUM 6
#define IP_ADDRESS_BYTES_NUM 4
#define ENC28J60_SPI_TIMEOUT 10
#define ENC28J60_OP_CODE_OFFSET 5
#define ENC28J60_REG_BANK_OFFSET 5
#define ENC28J60_REG_TYPE_OFFSET 7
#define ENC28J60_TX_BUF_START 0x0000
#define ENC28J60_RX_BUF_START 0x0600
#define ENC28J60_RX_BUF_END 0x1FFF
#define ENC28J60_FRAME_RX_OK_MASK 0x80
#define ENC28J60_REG_BANK_MASK 0x60
#define ENC28J60_REG_TYPE_MASK 0x80
#define ENC28J60_REG_ADDR_MASK 0x1F
#define ENC28J60_BUF_COMMAND_ARG 0x1A
#define ENC28J60_RESET_COMMAND_ARG 0x1F
#define ENC28J60_FRAME_DATA_MAX 1024
#define ENC28J60_BB_PACKET_GAP 0x15
#define ENC28J60_NBB_PACKET_GAP 0x0C12
#define ENC28J60_BANK_0_BITS (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_1_BITS (BANK_1 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_2_BITS (BANK_2 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_3_BITS (BANK_3 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_BANK_COMMON_BITS (BANK_0 << ENC28J60_REG_BANK_OFFSET)
#define ENC28J60_COMMON_REGS_ADDR 0x1B
#define ENC28J60_ETH_REG_BIT (ETH_REG << ENC28J60_REG_TYPE_OFFSET)
#define ENC28J60_MAC_MII_REG_BIT (MAC_MII_REG << ENC28J60_REG_TYPE_OFFSET)
// Common bank registers
#define EIE (0x1B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define EIR (0x1C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ESTAT (0x1D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ECON2 (0x1E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define ECON1 (0x1F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_COMMON_BITS)
#define EIR_PKTIF_BIT (1 << 6)
#define EIR_DMAIF_BIT (1 << 5)
#define EIR_LINKIF_BIT (1 << 4)
#define EIR_TXIF_BIT (1 << 3)
#define EIR_TXERIF_BIT (1 << 1)
#define EIR_RXERIF_BIT (1 << 0)
#define ECON2_AUTOINC_BIT (1 << 7)
#define ECON2_PKTDEC_BIT (1 << 6)
#define ECON2_PWRSV_BIT (1 << 5)
#define ECON2_VRPS_BIT (1 << 3)
#define ECON1_TXRST_BIT (1 << 7)
#define ECON1_RXRST_BIT (1 << 6)
```

```
#define ECON1_DMAST_BIT (1 << 5)
#define ECON1_CSUMEN_BIT (1 << 4)
#define ECON1_TXRTS_BIT (1 << 3)
#define ECON1_RXEN_BIT (1 << 2)
#define ECON1_BSEL1_BIT (1 << 1)
#define ECON1_BSEL0_BIT (1 << 0)
// Bank 0 registers
#define ERDPTL (0x00 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERDPTH (0x01 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EWRPTL (0x02 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EWRPTH (0x03 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXSTL (0x04 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXSTH (0x05 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXNDL (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ETXNDH (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXSTL (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXSTH (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXNDL (0x0A | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXNDH (0x0B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXRDPTL (0x0C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXRDPTH (0x0D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXWRPTL (0x0E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define ERXWRPTH (0x0F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMASTL (0x10 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMASTH (0x11 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMANDL (0x12 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMANDH (0x13 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMADSTL (0x14 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMADSTH (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMACSL (0x16 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
#define EDMACSH (0x17 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_0_BITS)
// Bank 1 registers
#define EHT0 (0x00 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT1 (0x01 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT2 (0x02 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT3 (0x03 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT4 (0x04 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT5 (0x05 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT6 (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EHT7 (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM0 (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM1 (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM2 (0x0A | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM3 (0x0B | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM4 (0x0C | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM5 (0x0D | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM6 (0x0E | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMM7 (0x0F | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMCSL (0x10 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMCSH (0x11 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
```

```
#define EPMOL (0x14 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define EPMOH (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define ERXFCON (0x18 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
#define ERXFCON_UCEN_BIT (1 << 7)
#define ERXFCON_ANDOR_BIT (1 << 6)
#define ERXFCON_CRCEN_BIT (1 << 5)
#define ERXFCON_PMEN_BIT (1 << 4)
#define ERXFCON_MPEN_BIT (1 << 3)
#define ERXFCON_HTEN_BIT (1 << 2)
#define ERXFCON_MCEN_BIT (1 << 1)
#define ERXFCON_BCEN_BIT (1 << 0)
#define EPKTCNT (0x19 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_1_BITS)
// Bank 2 registers
#define MACON1 (0x00 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACON1_TXPAUS_BIT (1 << 3)
#define MACON1_RXPAUS_BIT (1 << 2)
#define MACON1_PASSALL_BIT (1 << 1)
#define MACON1_MARXEN_BIT (1 << 0)
#define MACON3 (0x02 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACON3_PADCFG2_BIT (1 << 7)
#define MACON3_PADCFG1_BIT (1 << 6)
#define MACON3_PADCFG0_BIT (1 << 5)
#define MACON3_TXCRCEN_BIT (1 << 4)
#define MACON3_PHDRLEN_BIT (1 << 3)
#define MACON3_HFRMEN_BIT (1 << 2)
#define MACON3_FRMLNEN_BIT (1 << 1)
#define MACON3_FULDPX_BIT (1 << 0)
#define MACON4 (0x03 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MABBIPG (0x04 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAIPGL (0x06 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAIPGH (0x07 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACLCON1 (0x08 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MACLCON2 (0x09 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAMXFLL (0x0A | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MAMXFLH (0x0B | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MICMD (0x12 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MICMD_MIIRD_BIT (1 << 0)
#define MIREGADR (0x14 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIWRL (0x16 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIWRH (0x17 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIRDL (0x18 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
#define MIRDH (0x19 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_2_BITS)
// Bank 3 registers
#define MAADR5 (0x00 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR6 (0x01 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR3 (0x02 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR4 (0x03 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR1 (0x04 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MAADR2 (0x05 | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTSD (0x06 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
```

```
#define EBSTCON (0x07 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTCSL (0x08 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EBSTCSH (0x09 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define MISTAT (0x0A | ENC28J60_MAC_MII_REG_BIT | ENC28J60_BANK_3_BITS)
#define MISTAT_BUSY_BIT (1 << 0)
#define EREVID (0x12 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define ECOCON (0x15 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EFLOCON (0x17 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EPAUSL (0x18 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
#define EPAUSH (0x19 | ENC28J60_ETH_REG_BIT | ENC28J60_BANK_3_BITS)
// PHY registers
#define PHCON1 (0x00)
#define PHSTAT1 (0x01)
#define PHID1 (0x02)
#define PHID2 (0x03)
#define PHCON2 (0x10)
#define PHCON2_FRCLNK_BIT (1 << 14)
#define PHCON2_TXIS_BIT (1 << 13)
#define PHCON2_JABBER_BIT (1 << 10)
#define PHCON2_HDLDIS_BIT (1 << 8)
#define PHSTAT2 (0x11)
#define PHIE (0x12)
#define PHIR (0x13)
#define PHLCON (0x14)
#define PHLCON_LACFG3_BIT (1 << 11)
#define PHLCON_LACFG2_BIT (1 << 10)
#define PHLCON_LACFG1_BIT (1 << 9)
#define PHLCON_LACFG0_BIT (1 << 8)
#define PHLCON_LBCFG3_BIT (1 << 7)
#define PHLCON_LBCFG2_BIT (1 << 6)
#define PHLCON_LBCFG1_BIT (1 << 5)
#define PHLCON_LBCFG0_BIT (1 << 4)
#define PHLCON_LFRQ1_BIT (1 << 3)
#define PHLCON_LFRQ0_BIT (1 << 2)
#define PHLCON_STRCH_BIT (1 << 1)
#define MISTAT_BUSY_BIT (1 << 0)
#define ENC28J60_HEADER_SIZE 6
#define ENC28J60_CRC_SIZE 4
typedef enum
{
READ_CONTROL_REG,
READ_BUFFER_MEM,
WRITE_CONTROL_REG,
WRITE_BUFFER_MEM,
BIT_FIELD_SET,
BIT_FIELD_CLEAR,
SYSTEM_RESET,
COMMANDS_NUM,
} ENC28J60_Command;
typedef enum
```

```
{
CS_LOW = 0,
CS_HIGH = 1,
} ENC28J60_CS_State;
typedef enum
{
BANK_0,
BANK_1,
BANK_2,
BANK_3,
} ENC28J60_RegBank;
typedef enum
{
ETH_REG,
MAC_MII_REG,
} ENC28J60_RegType;
typedef struct ENC28J60_Frame
{
uint16_t nextPtr;
uint16_t length;
uint16_t status;
uint8_t data[ENC28J60_FRAME_DATA_MAX];
uint32_t checkSum;
} ENC28J60_Frame;
extern void ENC28J60_Init()
```