МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Пермский государственный национальный
исследовательский университет»

Механико-математический факультет

УДК 004.422.833

Кафедра математического обеспечения вычислительных систем

Разработка средств автоматизации программирования устройств Интернета вещей на базе платформы SciVi

Выпускная квалификационная работа бакалавра

Работу выполнил студент группы ПМИ-1,2-2019 4 курса механико-математического факультета
Лукьянов Александр Михайлович

_____

«___» _____2023 г.

Научный руководитель:
кандидат физико-математических наук,
доцент кафедры МОВС
Рябинин Константин Валентинович

_____

«___» _____2023 г.

Пермь 2023

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ **2**

ВВЕДЕНИЕ **3**

1 Анализ наиболее популярных решений управления энергонезависимой памятью **6**

1.1 Требования к системе управления энергонезависимой памятью 6

1.2 Стандартная библиотека 7

1.3 Библиотека EEPROMEx 7

1.4 Библиотека EEManager 10

1.5 Библиотека EEPROMWearLevel 11

1.6 Вывод 14

2 Разработка библиотеки менеджера EEPROM **15**

2.1 Уточнение требований к разрабатываемой библиотеке 15

2.2 Разработка структуры библиотеки 17

2.2.1 Общая структура библиотеки 17

2.2.2 Внешний интерфейс библиотеки 18

2.2.3 EEPROM-переменные 20

2.2.4 Разделы памяти 21

2.2.5 Менеджер памяти 21

2.3 Разработка библиотеки 21

ЗАКЛЮЧЕНИЕ **24**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **25**

ПРИЛОЖЕНИЕ А **27**

ВВЕДЕНИЕ

Автоматизация процесса разработки и отладки программ для микроконтроллеров и для созданных на их основе устройств интернета вещей (англ. internet of things, IoT) является крайне важной и острой задачей для разработчиков. Это вызвано, в первую очередь, ограниченностью интерфейсов микроконтроллеров для взаимодействия с пользователем. Кроме того, разработку программ для микроконтроллеров в значительной степени затрудняет отсутствие высокоуровневых средств отладки. С указанными проблемами, в частности, сталкивается платформа научной визуализации и визуальной аналитики SciVi, разработанная сотрудниками Пермского государственного национального исследовательского университета [1; 2].

Данная платформа использует микроконтроллеры для исполнения пользовательских алгоритмов. Эти алгоритмы передаются на микроконтроллеры и хранятся на них в виде онтологий, сжатых с помощью набора программных средств EON [3; 4]. В дальнейшем эти онтологии, представляющие алгоритмы, интерпретируются и исполняются на самих микроконтроллерах.

Целью выпускной квалификационной работы, в рамках которой выполнена данная научно-исследовательская работа, является разработка требуемых платформой SciVi программных средств для автоматизации программирования микроконтроллеров.

В научно-исследовательской работе рассматривается один из аспектов автоматизации работы описанной выше системы — сохранение необходимых для интерпретации онтологий данных в энергонезависимой памяти микроконтроллеров с целью повышения уровня отказоустойчивости системы.

Энергонезависимая память — особый вид запоминающих устройств, способный хранить данные при отсутствии электропитания. Такая память используется в составе вычислительных устройств, в том числе для хранения данных, необходимых для их инициализации, и

конфигурационных данных между их запусками.

Задача хранения конфигурационных данных при отсутствии электропитания особо остро стоит при работе с микроконтроллерами. Это обусловлено, во-первых, уязвимостью таких устройств к перебоям электропитания и, во-вторых, особенностями условий их использования: устройства с микроконтроллерами обычно создаются для автономной работы, поэтому после временного отключения питания они должны самостоятельно восстанавливать своё прошлое состояние. В микроконтроллерах для решения этой задачи обычно используются электрически стираемые перепрограммируемые постоянные запоминающие устройства (ЭСППЗУ, англ. Electrically Erasable Programmable Read-Only Memory, EEPROM) — вид устройств энергонезависимой памяти, позволяющих электрическим импульсом стереть сохранённые данные, а затем, при необходимости, записать новые [5; 6].

В платформе SciVi уже реализовано сохранение информации в EEPROM, однако сделано это за счёт стандартных средств. Их низкоуровневость и ограниченность не позволяют использовать EEPROM удобно и, главное, расширять его применение хранением новых данных.

В основе данной научно-исследовательской работы лежит поиск решения указанных проблем в применении стандартных средств использования EEPROM микроконтроллеров.

Цель работы: разработать программный модуль с высокоуровневым интерфейсом, позволяющий удобно и без необходимости ручной настройки сохранять и считывать информацию из EEPROM микроконтроллеров, в соответствии с требованиями платформы SciVi.

Объект исследования данной работы: автоматизация периферийных вычислений.

Предмет исследования: средства платформы SciVi для организации онтологически-управляемых периферийных вычислений.

Для достижения цели работы были поставлены следующие задачи:

1. Составить требования к необходимому программному модулю.
2. Исследовать существующие средства для работы с энергонезависимой памятью и, в частности, EEPROM

микроконтроллеров.

3. При возможности, выбрать одно из таких средств для использования в качестве основы разрабатываемого модуля.
4. Разработать программный модуль для работы с EEPROM микроконтроллеров, соответствующий всем поставленным требованиям.
5. Провести тестирование и отладку разработанного программного модуля.
6. Интегрировать разработанный модуль в платформу SciVi.

1 Анализ наиболее популярных решений управления энергонезависимой памятью

1.1 Требования к системе управления энергонезависимой памятью

В платформе SciVi в основном применяются микроконтроллеры серии ESP8266. А для их программирования используются инструменты среды разработки Arduino IDE, позволяющие программировать микроконтроллеры, используя язык программирования C++, а также специальное дополнение к этой среде для работы с ESP8266 [7], содержащее, в частности, набор „стандартных“ библиотек.

Таким образом, необходимый программный модуль должен представлять собой библиотеку классов языка программирования C++, может использовать стандартный набор библиотек Arduino IDE и указанного дополнения к ней. Такая библиотека должна:

1. Предоставлять пользователю возможность сохранять и считывать данные из EEPROM микроконтроллера. При этом:
   1.1. Данные могут иметь произвольную структуру.
   1.2. Доступ к ним должен производиться по некоторым идентификаторам, уникальным для различных данных и без необходимости ручных манипуляций с адресами EEPROM со стороны пользователя.
2. Автоматически определять факт наличия в EEPROM данных с заданным идентификатором, определять адрес для записи новых данных, сохранять в EEPROM метаданные о хранящихся данных для их использования после перезапуска микроконтроллера.
3. Минимизировать количество операций записи в EEPROM, т.к. каждая ячейка такой памяти может быть перезаписана ограниченное количество раз (обычно производители гарантируют от 100.000 до 1.000.000 циклов перезаписи), после чего выходит из строя.
4. Выполняться на микроконтроллерах серии ESP8266 и, по возможности, на платформе Arduino, так как эта платформа

является наиболее популярной и распространённой.

Ключевым требованием является полная автоматизация работы с адресами EEPROM, это необходимо для создания возможности использования EEPROM в различных независимых программных модулях. В противном случае, таким модулям понадобилось бы каким-либо образом обмениваться информацией об используемых ими адресах для избежания чтения и записи разными модулями в одни и те же ячейки EEPROM.

1.2 Стандартная библиотека

В стандартный набор библиотек Arduino IDE уже входит библиотека для работы с EEPROM [8]. Однако она предоставляет только простые функции, такие как записать и считать байт по указанному адресу. Позже в неё были добавлены функции для чтения и записи данных произвольных типов, но также только по явно указанному адресу. Очевидно, это делает стандартную библиотеку нарушающей все поставленные требования, однако её функции можно использовать в качестве низкоуровневого интерфейса EEPROM в разрабатываемой библиотеке. Кроме указанных, стандартная библиотека содержит функцию-обёртку вокруг функции записи, производящую фактическую перезапись данных только тогда, когда они отличаются от хранящихся по указанному адресу в данный момент. В дальнейшем, в большинстве случаев, разумно использовать для записи именно эту функцию с целью уменьшения износа EEPROM.

1.3 Библиотека EEPROMEx

Библиотека EEPROMEx (от EEPROM Extended) — одна из первых разработок для работы с EEPROM микроконтроллеров в среде Arduiono IDE [9]. Данная библиотека была создана раньше, чем описанная выше, поэтому часть предоставляемых или возможностей совпадает, однако реализованы независимо друг от друга. EEPROMEx содержит функции для чтения и записи в EEPROM данных некоторых стандартных типов: целочисленных беззнаковых чисел длиной в 8, 16 и 32 бита и 32-х и 64-х битных чисел с плавающей точкой. В библиотеке также содержатся функции для чтений и записи отдельных битов и, как и в стандартной библиотеке, функции для работы с данными пользовательских типов и

аналоги всех функций записи, производящие запись только при отличии данных. Кроме того EEPROMEx содержит и уникальную возможность — с помощью класса EEPROMVar связывать переменные в коде программы для микроконтроллера и данные в EEPROM (их адреса), причём эти адреса назначаются автоматически, что косвенно соответствует части требований, описанных в пункте 1.1. Рассмотрим пример использования данной возможности, демонстрирующий большую часть возможностей данного класса:

```
1 EEPROMVar<float> eepromFloat(5.5);
2
3 floatVar.restore();
4
5 floatVar = 10.5;
6
7 float input = floatVar;
8
9 floatVar.save();
```

Как можно увидеть в строке 1 примера использования, при создании объекта класса EEPROMVar указывается тип данных, которые необходимо хранить (в примере — float), и значение по умолчанию, которое передаётся в конструктор в виде параметра (в примере — 5.5). Метод restore, вызываемый в строке 3, считывает из EEPROM данные, соответствующие данному объекту, и сохраняет эти данные в него. В строке 5 демонстрируется возможность изменения хранимых данных с помощью оператора присвоения, причём в качестве левостороннего значения выступает объект класса EEPROMVar, правостороннего — значение хранимого объектом типа пользовательских данных. Это достигается за счёт переопределения оператора присвоения для данного класса. В строке 7 показан обратный переход: объект класса EEPROMVar может быть переведёт в значение хранимого им типа. В последней строке показан вызов метода save, сохраняющего значение, хранящееся в объекте в EEPROM.

Также для класса EEPROMVar определены некоторые операторы, такие как оператор инкремента (++), присвоения со сложением (+=) и другие операторы, связанные с изменением хранимого значения. Такие операторы вызывают в своём теле аналогичные для хранимых данных, что, разумеется, требует их существования.

Описанный класс механизм имеет ряд значительных недостатка.

Первый из них заключается в способе связывания данных и адреса, по которому они хранятся в EEPROM. Обычно для этого данным выдаются уникальные имена, и каким-либо образом сохраняется информация, что данные с определённым именем находятся по определённому адресу. Как можно увидеть в приведённом примере, при использовании библиотеки EEEPROMEx этого не происходит. В этом случае данные располагаются в EEPROM друг за другом в порядке создания объектов класса EEPROMVar. Таким образом, для того, чтобы гарантировать, что при перезапуске микроконтроллера объекты класса EEPROMVar связываются с одними и теме же адресами в EEPROM, необходимо обеспечивать создание этих объектов в строго одинаковом порядке при каждом включении устройства. Эта работа ложится на плечи конечного пользователя — разработчика программы для микроконтроллера. Причём даже в случае, если разработчик не применяет описываемый класс напрямую, но он используется в импортируемых библиотеках.

Второй проблемой является невозможность определения первого создания объекта для каких-либо пользовательских данных, то есть момента в который эти данные отсутствуют в EEPROM и необходимо записать значение по умолчанию. И вновь разработчик конечной программы должен каким-либо образом самостоятельно определять, когда после создания объекта необходимо проводить сохранение в EEPROM значение по умолчанию, а когда наоборот считывание значения, уже хранящегося в памяти.

В конечном итоге можно сделать вывод, что библиотека EEPROMEx частично соответствует предъявляемым требования, но при этом её использование нельзя назвать удобным, так как оно требует от конечного пользователя большой работы, которая могла бы быть автоматизирована. При этом идея использовать переопределение операторов для получения и изменения пользовательских данных в EEPROM является крайне удачной и удобной в использовании. Такой вариант предпочтительнее использования для тех же целей отдельных методов, так как, во-первых, позволяет пользователю библиотеки писать меньше кода и, во-вторых, позволяет ему думать об объекте, описывающем данные в EEPROM, как о самих этих данных. В дальнейшем при разработке собственной библиотеки

имеет смысл реализовать аналогичный интерфейс.

1.4 Библиотека EEManager

Как и SciVi в данный момент, большая часть проектов, хранящих какие-либо данные в EEPROM, ограничивается использованием стандартной библиотеки. И до недавнего времени в открытом доступе отсутствовали более высокоуровневые альтернативы. Однако не так давно появилась новая библиотека для работы с EEPROM — EEManager [10]. Она имеет открытый исходный код (опубликован под лицензией MIT [11]) и документацию на русском языке.

Данная библиотека имеет следующие преимущества:

- Реализован механизм отложенной записи: по умолчанию данные записываются в EEPROM с заданной задержкой после последней команды на запись. Использование такого подхода имеет смысл в ситуациях, когда данные должны перезаписываться много раз за короткий промежуток времени, в действительности же, с таким механизмом данные в EEPROM будут записаны только в последний раз, что значительно замедлит износ памяти. В то же время этот механизм имеет значительный недостаток: если потеря питания произойдёт после команды записи, но до истечения задержки, новые данные записаны не будут. Это делает использование такого механизма оправданным только в устройствах, для которых гарантия записи не является обязательной и точность восстановления состояния после потери питания не представляет критической важности.
- Библиотека также реализует „механизм ключа первой записи“. Вместе с каждым блоком данных в EEPROM хранится специальный однобайтовый ключ. При обращении к блоку данных пользователь указывает придуманный им ключ, который не должен изменяться от запуска к запуску, а из EEPROM считывается записанное значение ключа. Если они совпадают, значит необходимые данные уже находятся в EEPROM и их необходимо только считать, иначе данные никогда не были записаны, в этом случае данные должны быть сохранены в

EEPROM.

Работа с библиотекой осуществляется следующим образом:

1. Создаётся переменная в энергозависимой памяти, значение которой необходимо хранить в EEPROM.
2. Создаётся специальный объект, описывающий блок EEPROM. При этом пользователь указывает переменную в энергозависимой памяти, значение которой необходимо хранить, и адрес в EEPROM, начиная с которого должна быть записана эта переменная.
3. С помощью механизма ключа первой записи либо в EEPROM записывается значение переменной по умолчанию, либо наоборот сохранённое в EEPROM значение считывается в переменную.
4. В дальнейшем, по необходимости, текущее значение переменной записывается в EEPROM. Для этого у описанного выше объекта существует два метода: для немедленной записи и для запуска таймера записи с задержкой.

Несмотря на указанные преимущества перед стандартной библиотекой, библиотека EEManager так же не может быть использована в готовом виде, так как обращение к блокам данных в ней производится только по их адресам. Это делает невозможным независимое использование данной библиотеки из различных программных модулей.

1.5 Библиотека EEPROMWearLevel

В большинстве микроконтроллеров, в частности, на платформе Arduino, используется EEPROM с возможностью перезаписи отдельного байта, таким образом одни ячейки памяти могут изнашиваться быстрее других. Для уменьшения скорости общего износа памяти имеет смысл как можно равномернее распределять количество циклов перезаписи по всем ячейкам EEPROM. Классический способ такого распределения — использование кольцевого буфера [12].

Идея этого метода заключается в перезаписи данных не по тому же адресу, а по новому, с некоторым сдвигом вперёд. Когда такой сдвиг становится невозможным из-за недостаточного объёма памяти, запись снова производится по первоначальному адресу. При этом, чтобы иметь

доступ к данным, необходимо постоянно хранить их текущий адрес или счётчик циклов перезаписи, и определять адрес по его значению. В самом простом случае, если все данные хранятся в одном блоке, это можно сделать, умножив номер цикла на размер этого блока данных. Однако, если хранить их по фиксированному адресу в EEPROM, соответствующие ячейки будут изнашиваться быстрее, что сделает использование кольцевого буфера бессмысленным.

Один из способов эффективной реализации счётчика — использование двух кольцевых буферов: одного для данных, второго — для счётчика. Если при этом каждый раз при прохождении целого цикла записи (записи данных по изначальному адресу) для второго буфера, заполнять его нулевыми значениями, то после перезапуска микроконтроллера в этом буфере можно найти актуальное значение счётчика тривиальным образом — это будет последнее ненулевое значение в нём.

Другой способ основывается на использовании ещё одной особенности EEPROM микроконтроллеров: после стирания байта все его биты устанавливаются равными единице, и стандартная библиотека Arduino IDE предоставляет возможность устанавливать отдельным битам нулевое значение, без стирания всего байта. За счёт этого можно хранить счётчик в своеобразной унарной системе счисления: десятичное значение счётчика равно количеству нулей в его побитовой записи. Это позволит стирать ячейки EEPROM, хранящие счётчик, только при прохождении полного цикла записи в буфере данных.

Кольцевой буфер с последним из описанных механизмом эффективного счётчика реализует открытая библиотека EEPROMWearLevel [13] (опубликована под лицензией Apache 2.0 [14]). EEPROMWearLevel предоставляет интерфейс, аналогичный стандартной библиотеке, за исключением функции инициализации, в которую в данной библиотеке необходимо подать требуемое количество блоков EEPROM, дополняя этот интерфейс возможностью обращения к блокам данных по их индексам. Указанное отличие в функциях инициализации вызвано тем, что данная библиотека создаёт отдельный кольцевой буфер для каждого блока данных, при этом весь объём EEPROM делится на буферы поровну

между всеми хранящимися блоками данных. Такое решение является неэффективным в случае хранения блоков разного размера. В худшем случае размер одного или нескольких блоков может превышать размеры выделенного буфера, однако такой случай обрабатывается библиотекой и приведёт к выводу соответствующей ошибки и отмене записи такого блока. Целостность других блоков при этим нарушена не будет.

Индексы блоков данных, используемые в этой библиотеке для доступа к ним, можно считать уникальными идентификаторами, описанными в требованиях, приведённых в начале главы. Однако необходимость общей инициализации с указанием суммарного количества блоков данных делает невозможным применение данной библиотеки в независимых программных модулях. Такой необходимости можно избежать, например, введя иерархическую структуру разделения EEPROM: сначала разделить весь объём EEPROM на крупные разделы (по одному или несколько на модуль), а затем внутри каждого из них — на кольцевые буферы, аналогично уже реализованному принципу. Чтобы гарантировать, что потребности всех модулей в использовании EEPROM учтены, необходимо:

- Либо проводить разделение EEPROM только после того, как все модули (посредством вызова некоторой специальной функции) сообщат библиотеке управления EEPROM их потребности в использовании EEPROM. Однако такой подход требует обязать конечного пользователя вызывать функцию инициализации библиотеки управления EEPROM строго после инициализации всех модулей, которым она необходима. А разработчиков этих модулей — вызывать функцию, описанную выше, при их инициализации. Такой подход, очевидно, является неудобным, так как требует от конечного пользователя знания о том, необходимо ли используемым им модулям хранить данные в EEPROM.
- Либо реализовать возможности динамического добавления новых разделов EEPROM, при этом уменьшая размер уже созданных разделов. Такой подход лишён недостатков первого, однако его использование значительно усложнит работу библиотеки из-за необходимости уменьшения размеров каждого кольцевого буфера

при добавлении нового раздела и, возможно, сдвига данных, если эти данные в момент добавления выходят за уменьшенные границы буфера.

Можно заключить, что существует возможность дополнения данной библиотеки таким образом, чтобы она удовлетворяла всем поставленным требованиям. В то же время данная библиотека не может напрямую использоваться с микроконтроллерами ESP8266, так как она содержит платформозависимый код, выполнение которого возможно только микроконтроллерами с архитектурой AVR, к которым ESP8266 не относится. Кроме того, микроконтроллеры ESP8266 используют иной тип EEPROM — flash память [15; 16], которая не позволяет стирать и перезаписывать отдельные байты, а только их большие группы (обычно от 512 байт) [17]. Основное назначение этой памяти в данных микроконтроллерах — хранение исполняемых программ, однако её часть специально выделена для хранения пользовательских данных. При этом в ходе анализа исходного кода стандартной библиотеки [15] для работы с EEEPROM для ESP8266 было установлено, что для данных микроконтроллеров размер такой группы байт равен всему объёму flash-памяти, выделенному для пользовательских данных.

Таким образом, оказалось, что использование в программах для ESP8266 любых механизмов, подобных кольцевым буферам, не только не приводит к замедлению износа памяти, но и способно ускорять её износ из-за дополнительных операций перезаписи счётчиков и других метаданных.

1.6 Вывод

В ходе первого этапа работы были изучены особенности устройства и работы EEPROM. Были составлены требования к программному модулю для работы с EEPROM микроконтроллеров. С учётом этих требований были проанализированы существующие в открытом доступе решения. На основе анализа было принято решение о разработке собственного модуля, а также об использовании в нём стандартной библиотеки для работы с EEPROM и заимствования некоторых механизмов из других библиотек.

2 Разработка библиотеки менеджера EEPROM

2.1 Уточнение требований к разрабатываемой библиотеке

На основе требования об обеспечении независимости работы с EEPROM из различных программных модулей и с целью уменьшения количества коллизий пользовательских идентификаторов данных, было принято решение реализовать следующее:

1. Ввести иерархию идентификаторов блоков данных: не хранить все идентификаторы в едином пространстве имён, а создавать множество таких пространств и реализовать механизм обращения к ним по особым идентификаторам, которые должны быть уникальны между собой.
2. Обязать разработчиков программных модулей, использующих разрабатываемую библиотеку, создавать для этих модулей отдельные пространства имён EEPROM, идентификаторы которых должны быть связаны с названием модуля, а непосредственно используемые блоки данных описывать внутри этих пространств имён.
3. Обязать разработчиков самостоятельно поддерживать уникальность идентификаторов внутри отдельных модулей.

Такой шаг позволит изолировать друг от друга идентификаторы, используемые различными разработчиками в различных программных модулях. Важно отметить, что вероятность коллизий идентификаторов самих пространств имён является небольшой, с учётом привязки этих идентификаторов к названиям модулей, в которых они создаются. Таким образом, разработчики смогут применять данную библиотеку, не задумываясь о других программных модулях, использующих её, что выполняет основное требование к разрабатываемой библиотеке. В дальнейшем будем называть такие пространства имён разделами, а пользовательские блоки данных, хранящиеся в EEPROM — EEPROM-переменными.

На основе дополненного списка требований к библиотеке можно составить диаграмму прецедентов работы с ней. Она представлена на

рисунке 2.1.

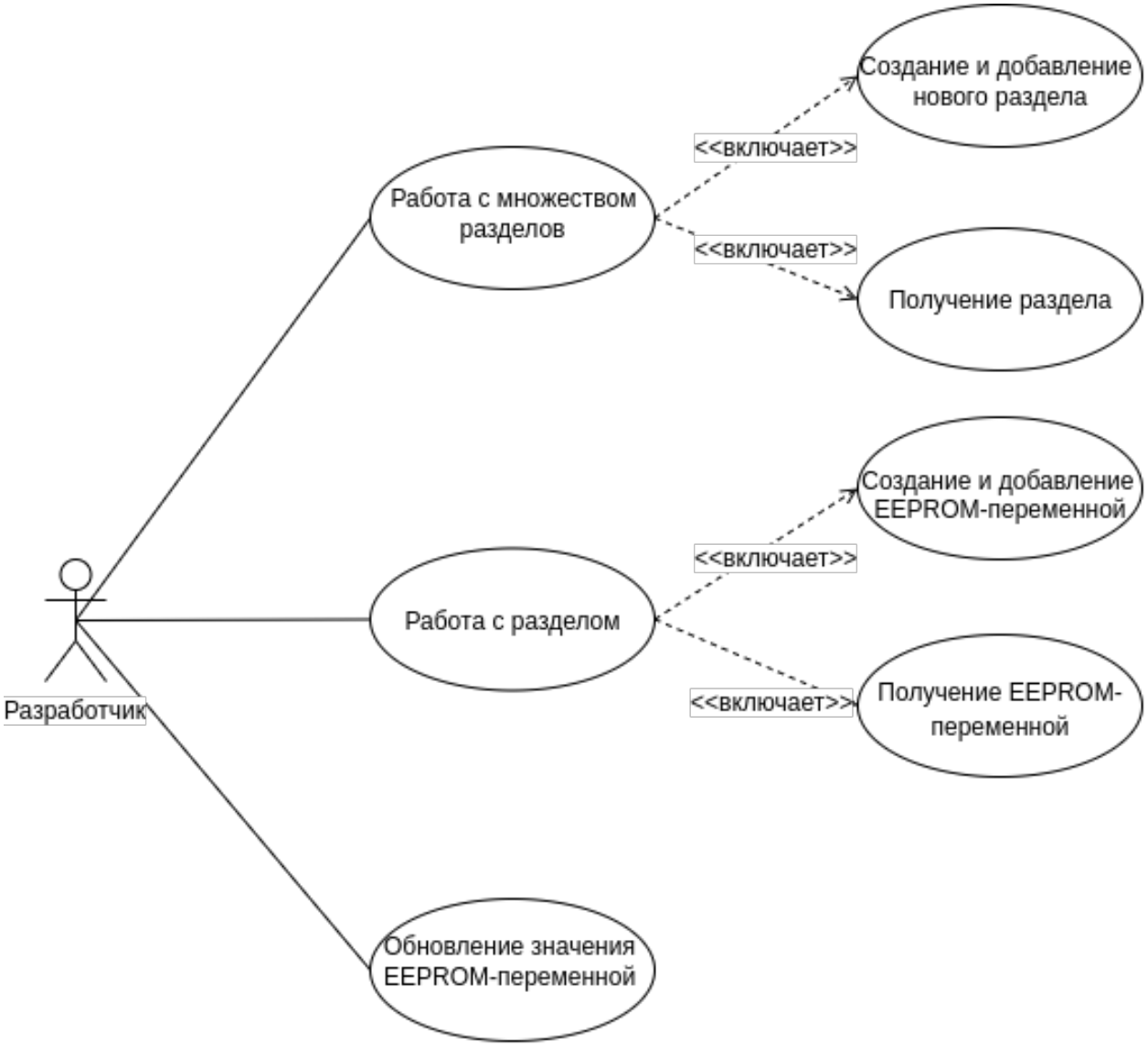

Рисунок 2.1 – Диаграммы прецедентов использования библиотеки

На диаграмме отсутствует прецедент удаления переменной в связи с тем, что, как писалось выше, основная цель применения EEPROM в программах для микроконтроллеров — сохранение состояния устройства между его запусками. Как правило, для запуска микроконтроллера с одной и той же программой необходимы одни и те же данные, следовательно, удалять их нет необходимости.

2.2 Разработка структуры библиотеки

2.2.1 Общая структура библиотеки

В соответствии с общепринятым стилем, библиотека должна быть написана с использованием объектно-ориентированной парадигмы программирования. Основные понятия, которыми оперирует разрабатываемая библиотека — раздел и EEPROM-переменная, следовательно, в библиотеке должны содержаться классы, соответствующие этим понятиям. Кроме того, необходим отдельный класс, оперирующий EEPROM в целом. Назовём его классом менеджера памяти.

Для хранения EEPROM-переменных внутри разделов было решено использовать механизм, схожий с механизмами, используемыми в файловых системах. В описание раздела входит адрес первой EEPROM-переменной этого раздела, а в описание каждой переменной — адрес следующей в этом же разделе, либо специальное значение, если переменная является последней в разделе. Информация о каждом разделе сохраняется в EEPROM за счёт создания отдельной EEPROM-переменной, связанной с объектом, описывающим этот раздел. Для хранения таких переменных создаётся специальный системный раздел, который находится всегда по одному и тому же адресу в EEPROM и хранит в себе все EEPROM-переменные, связанные с другими разделами. Кроме того, в системном разделе можно хранить последний использованный адрес для создания возможности добавлять новые разделы и переменные после перезапуска устройства.

Также в энергонезависимой вать его. Это необходимо для избежания чтения „мусорных“ данных, находящихся в EEPROM до первого использования менеджера. Также изменение этого значения в последующих версиях библиотеки можно использовать для принудительной переинициализации менеджера в случае изменения структуры хранения данных при выходе новой версии. Логическая схема размещения данных в EEPROM показана на рисунке 2.2.

Для идентификации разделов и EEPROM-переменных было решено использовать произвольные строковые выражения — имена.

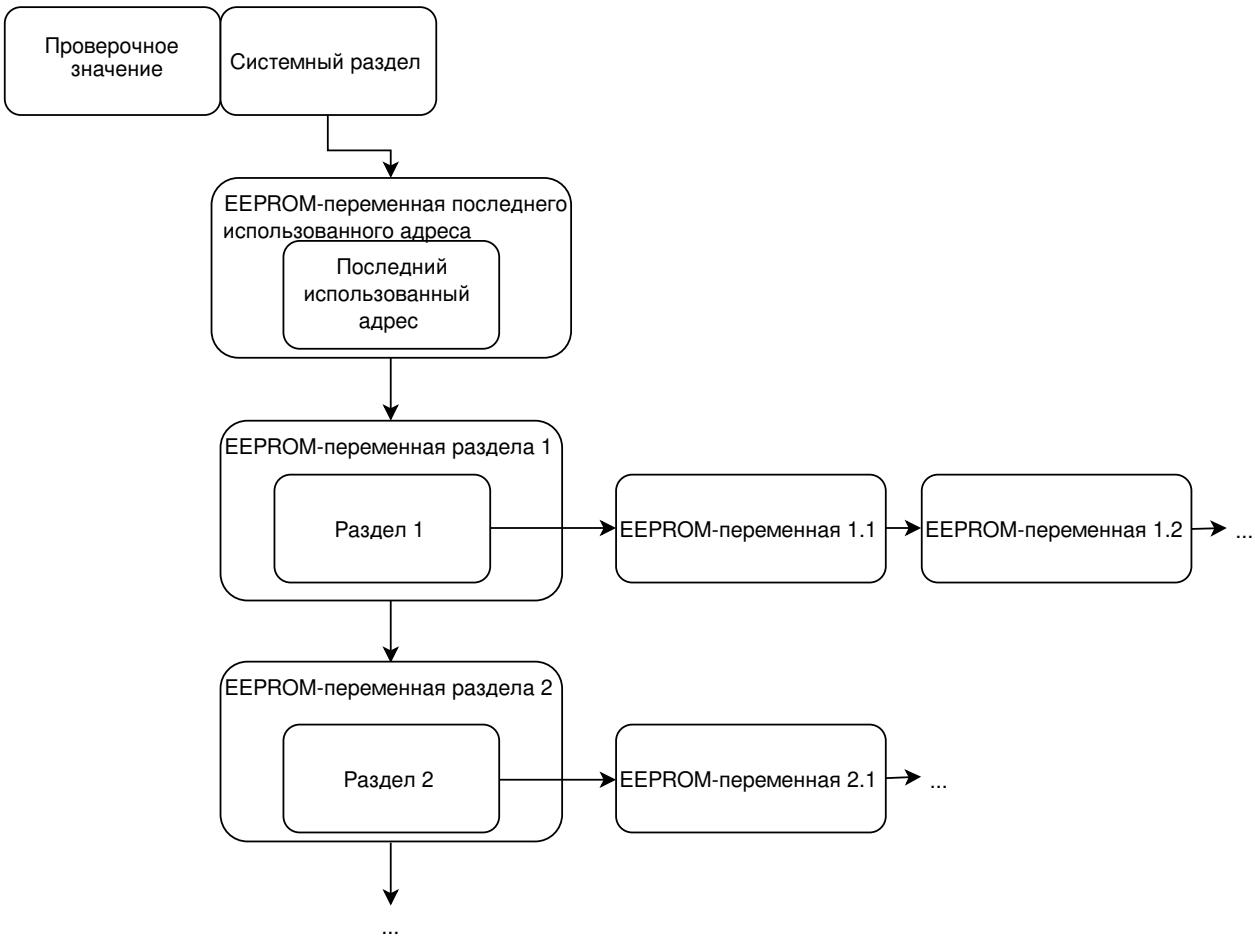

Рисунок 2.2 – логическая схема размещения данных в EEPROM

2.2.2 Внешний интерфейс библиотеки

С точки зрения пользователя библиотеки, её классы должны выглядеть как показано на рисунке 2.3. На данном рисунке EEPROMVar — класс, описывающий EEPROM-переменную. Его публичные методы взяты из соответствующего класса библиотеки EEManager (пункт 1.4). Описание методов:

1. updateNow — мгновенная запись нового значения в EEPROM.
2. update — запланировать запись нового значения, то есть запустить таймер отложенной записи.
3. tick — метод, который необходимо вызывать регулярно, если используется отложенная запись: реальная запись нового значения EEPROM произойдёт при первом вызове метода tick() после истечения задержки записи.
4. getTimeout — вернуть текущее значение задержки записи.
5. setTimeout — установить новое значение задержки записи.

Класс MemPart описывает раздел EEPROM. Его метод getVar должен возвращать EEPROM-переменную с именем, равным значению параметра name, и связывать её с локальной переменной произвольного типа T, хранящейся в оперативной памяти устройства, указателем на которую является параметр data. В случае, если EEPROM-переменная с указанным названием уже содержится в данном разделе, её значение должно копироваться в локальную переменную по тому же адресу, преобразуясь в значение типа T. Иначе, должна создаваться новая EEPROM-переменная, а значение локальной переменной записываться в неё. При этом важно учитывать, что, так как локальная переменная связывается с EEPROM-переменной, область видимости связываемой локальной переменной должна быть не уже области видимости объекта класса EEPROMVar, полученного вызовом метода getVar.

Класс EEMemManager описывает менеджер памяти и имеет два статических метода:

1. init — метод для инициализации менеджера.
2. getMemPart — метод, аналогичный методу getVar класса MemPart, но работающий с разделами, а не с переменными.

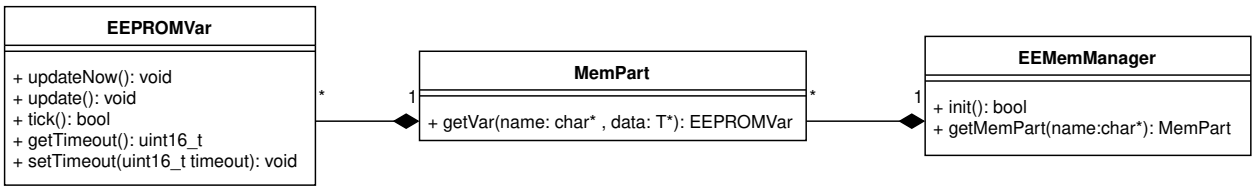

Рисунок 2.3 – Диаграмма классов библиотеки с точки зрения её пользователя

Итоговый алгоритм взаимодействия пользователя с библиотекой должен выглядеть следующим образом:

1. Вызов метода EEMemManager.init при инициализации программного модуля, использующего менеджер EEPROM.
2. Получение/создание раздела памяти вызовом метода EEMemManager.getMemPart с именем раздела, связанным с названием модуля.
3. Создание переменных в оперативной памяти, значение которых необходимо сохранять в EEPROM.
4. Получение/создание необходимых EEPROM-переменных и их

связывание с переменными в оперативной памяти с помощью метода getVar полученного раздела.

5. Вызов методов обновления значений EEPROM-переменных (updateNow или update и tick) по необходимости.
6. Предполагается, что для передачи синхронизируемых через EEPROM данных между частями программного модуля будут использоваться переменные из оперативной памяти. Однако при необходимости, повторным вызовом метода MemPart.getVar возможно считать актуальное значение EEPROM-переменной.

2.2.3 EEPROM-переменные

Помимо указателя на данные в оперативной памяти, которые необходимо сохранять, описание переменной должно включать значение задержки отложенной записи и мета-данные о записи переменной в EEPROM. Эти данные приведены в таблице 2.1.

Таблица 2.1 – Мета-данные EEPROM-переменной

Размер	Название поля	Описание
4 байта	nameHash	Хэш-значение имени переменной. Используется для фиксации объёма памяти, занимаемого именем. Используется хэш-функция CRC32 [18].
2 байта	dataSize	Размер данных, хранящихся в переменной.
2 байта	nextVarAddr	Адрес следующей переменной в этом же разделе.
2 байта	addr	Адрес в EEPROM, по которому хранится переменная.

Все указанные выше поля переменной, кроме адреса, необходимо хранить в EEPROM, адрес же можно узнать в процессе поиска переменной

и хранить только в оперативной памяти. Для удобства записи и чтения мета-данных, те из них, которые необходимо хранить в EEPROM, следует вынести в отдельную структура — VariableInfo.

2.2.4 Разделы памяти

Для удобства чтения и записи информации о разделе в EEPROM, все данные о разделе, которые необходимо сохранять между запусками, было решено вынести в отдельный класс — MemPartInfo, а класс MemPart унаследовать от него. В данной версии библиотеки к таким данным относится только адрес первой переменной раздела.

2.2.5 Менеджер памяти

Классу менеджера EEPROM необходимы следующие поля:

1. lastAddr — последний использованный адрес.
2. lastAddrVar — объект EEPROM-переменной для обновления последнего использованного адреса.
3. metaMemPart — объект, описывающий указанный в подпункте 2.2.1 системный раздел памяти.
4. startAddr — стартовый адрес в EEPROM, используемый менеджером. По умолчанию равен нулю, то есть используется весь доступный объём EEPROM.

В энергонезависимой памяти из этих значений необходимо хранить только последний использованный адрес: стартовый адрес и адрес, по которому хранится описание системного раздела, фиксируется в момент компиляции.

2.3 Разработка библиотеки

Итоговая диаграмма классов разрабатываемой библиотеки представлена ниже на рисунке 2.4.

Библиотека была реализована в соответствии с представленной диаграммой. В данном отчёте не приводятся детали реализации отдельных методов, однако исходный код разработанной библиотеки находится в открытом доступе в специальном репозитории, вместе с примерами использования и тестирующими библиотеку программами для

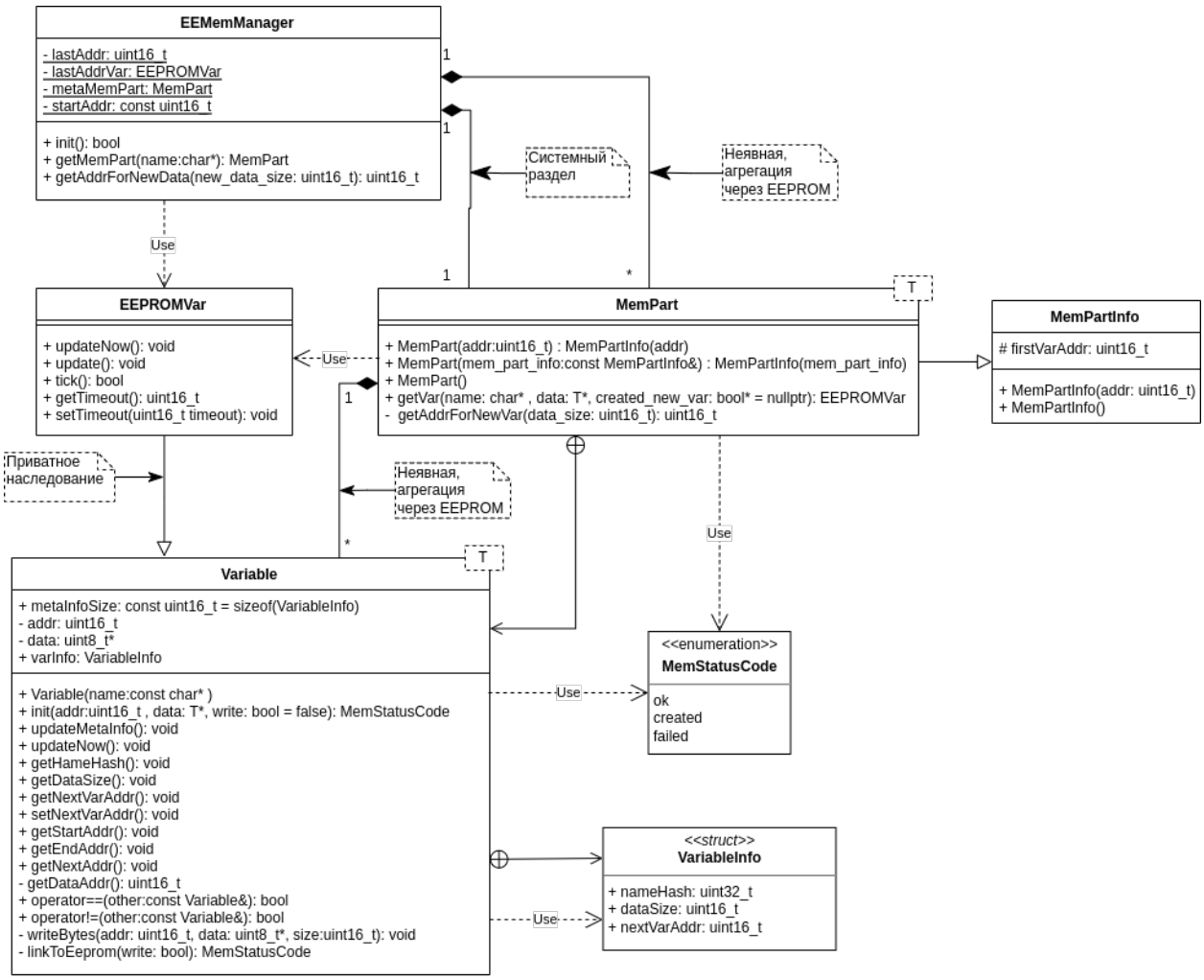

Рисунок 2.4 – Диграмма классов библиотеки

микроконтроллеров [19]. Код содержит комментарии, необходимые для генерации документации средствами Doxygen [20].

Работа библиотеки была протестирована на микроконтроллерах серии ESP8266 и платформы Arduino. Проведённое тестирование показало, что логика библиотеки работает в соответствии с поставленными требованиями. Однако, её интерфейс проявил себя крайне неудобным и недостаточно высокоуровневым. Общий вид этого интерфейса был продиктован идеей сохранить частичную обратную совместимость с библиотекой EEManager и переиспользовать её код, что привело к предоставлению пользователю слишком низкоуровневых функций для работы с EEPROM-переменными. В то же время более высокоуровневые функции реализованы не были.

В дальнейшем необходимо, используя часть реализованных низкоуровневых функции, разработать обновлённую версию библиотеки,

предоставляющую пользователю интерфейс более высокого уровня и не завязанную на использование библиотеки EEManager. В обновлённой версии необходимо:

1. Объединить в цельную сущность переменные в оперативной памяти и объекты, описывающие связанные с ними EEPROM-переменные. При использовании такой сущности первоначальное считывание значения из EEPROM (или запись в него значения по умолчанию) возможно производить в конструкторе класса, соответствующего такой сущности. Дальнейшее же получение из неё пользовательских данных и их обновление в EEPROM реализовать за счёт переопределения операторов. Это позволить одновременно (с точки зрения пользователя) работать и с данными в оперативной памяти, и с их представлением в EEPROM.
2. Заменить явное использование разделов неявным и позволить пользователю использовать большую глубину вложенности пространств имён.
3. Реализовать механизм отложенной записи для всего EEPROM в целом, а не для каждой EEPROM-переменной по отдельности.
4. Реализовать возможность немедленного обновления данных в EEPROM в случае необходимости.

ЗАКЛЮЧЕНИЕ

В ходе работы были составлены требования платформы SciVi к программному модулю управления энергонезависимой памятью микроконтроллеров. Были проанализированы существующие в этой области решения. По итогам анализа было установлено, что эти решения не удовлетворяют поставленным требованиям. Было принято решение о разработке собственного решения в виде библиотеки классов C++. Первая версия такой библиотеки была спроектирована, реализована и протестирована на целевых платформах. Тестирование выявило недостаточною высокоуровневость интерфейса разработанной библиотеки и необходимость её дальнейшей доработки. Однако, низкоуровневые функции разработанной библиотеки успешно прошли тестирование и могут быть использованы в её последующих версиях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ryabinin K.*, *Chuprina S.* Adaptive Scientific Visualization System for Desktop Computers and Mobile Devices // Procedia Computer Science. — 2013. — Т. 18. — С. 722—731.
2. Ontology-Driven Visual Analytics Platform for Semantic Data Mining and Fuzzy Classification / R. Konstantin [и др.] // Frontiers in Artificial Intelligence and Applications. — 2022. — Т. 358. — С. 1—7.
3. *Ryabininl K.*, *Chuprina S.* Ontology-Driven Edge Computing // Computational Science – ICCS 2020. — Springer, 2020. — С. 312—325.
4. *Konstantin R.*, *Svetlana C.*, *Ivan L.* Tackling IoT Interoperability Problems with Ontology-Driven Smart Approach. — 2021.
5. *Tarui Y.*, *Hayashi Y.*, *Nagai K.* Proposal of electrically reprogrammable non-volatile semiconductor memory // Proceedings of the 3rd Conference on Solid State Devices. — Tokyo : The Japan Society of Applied Physics, 1971. — С. 155—162.
6. *Tarui Y.*, *Hayashi Y.*, *Nagai K.* Electrically reprogrammable nonvolatile semiconductor memory // IEEE Journal of Solid-State Circuits. — 1972. — Т. 7, вып. 5. — С. 369—375.
7. Arduino core for ESP8266 WiFi chip / I. Grokhotkov [и др.]. — URL: https://github.com/esp8266/Arduino.
8. *Arduino Software*. EEPROM Library. — URL: https://docs.arduino.cc/learn/built-in-libraries/eeprom.
9. *Elenbaas T.* Arduino-EEPROMEx. — 2012. — URL: https://github.com/thijse/Arduino-EEPROMEx.
10. *AlexGyver*. EEManager. — 2021. — URL: https://github.com/GyverLibs/EEManager.
11. *Massachusetts Institute of Technology*. MIT License. — 1988. — URL: https://opensource.org/license/mit/.

12. *Atmel Corporation.* AVR101: High Endurance EEPROM Storage. — 2002. — URL: http://ww1.microchip.com/downloads/en/appnotes/doc2526.pdf.

13. *Rosenberg P.* EEPROMWearLevel. — 2016. — URL: https://github.com/PRosenb/EEPROMWearLevel.

14. *The Apache Software Foundation.* Apache License, Version 2.0. — 2004. — URL: https://www.apache.org/licenses/LICENSE-2.0.

15. *Grokhotkov I.* ESP8266 EEPROM Library. — 2014. — URL: https://github.com/esp8266/Arduino/tree/master/libraries/EEPROM.

16. *Hirnschall S.* ESP8266: Read and Write from/to EEPROM (Flash Memory). — 2021. — URL: https://blog.hirnschall.net/esp8266-eeprom/.

17. A new flash E2PROM cell using triple polysilicon technology / F. Masuoka [и др.] // 1984 International Electron Devices Meeting. — San Francisco, CA, USA : IEEE, 1984. — С. 464—467.

18. *Peterson W. W., Brown D. T.* Cyclic Codes for Error Detection // Proceedings of the IRE. — 1961. — Т. 49, вып. 1. — С. 228—235.

19. *Lukianov A.* EEManager. — 2023. — URL: https://github.com/almiluk/EEManager/.

20. Doxygen / D. van Heesch [и др.]. — 1997. — URL: https://www.doxygen.nl/.

ПРИЛОЖЕНИЕ А

Пример использования разработанной библиотеки

```
#include <EEManager.h>


struct my_datatype {
    uint8_t a = 1;
    char[10] = { 0 };
};

void setup() {
    Serial.begin(115200);

    // Инициализация библиотеки
    EEMemManager::init();
    // Получение разделов
    MemPart part1 = EEMemManager::GetMemPart("part1");
    MemPart part2 = EEMemManager::GetMemPart("part2");

    // Получение переменной
    // Вывод порядкового номера запуска
    bool created;
    int loads_num = 1;
   EEPROMVar loads_num_var = part1.getVar("loads_num", &loads_num, &create
    Serial.print("Load #");
    Serial.println(loads_num);
    if (created)
        Serial.println("First run. Counter was created in EEPROM.");
    else
        Serial.println("Not first run. Counter was found in EEPROM.");
```

```
    // Обновление порядкового номера запуска
    loads_num++;
    loads_num_var.updateNow();

    // Получение переменной сложного типа
    my_datatype my_var;
    EEPROMVar next_load_num_var = part1.getVar("testVar", &my_var);

    // Получение переменной с таким же именем, но из другого раздела
    int x = 4;
    EEPROMVar x_var = part2.getVar("testVar", &x);
    Serial.print("x ");
    Serial.println(x);

    x++;
    x_var.update();

    while (!x_var.tick()){
        Serial.println("Waiting...");
        delay(1);
    }
    Serial.println("Variable has been updated");

   // Чтение нового значения в другую переменную (локальная переменная x, в люб
    int y;
    EEPROMVar x_var = part2.getVar("testVar", &y);
    Serial.print("New value: ");
    Serial.println(y);
}

void loop() {

}
```