Разработка средств автоматизации программирования устройств Интернета вещей на базе платформы SciVi

Лукьянов Александр Михайлович

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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 [1; 2].

Микроконтроллеры, в частности, используются платформой научной визуализации и визуальной аналитики SciVi, разработанной сотрудниками Пермского государственного национального исследовательского университета [3; 4]. В SciVi уже реализовано сохранение настроечной 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 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 [5], 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 [6]. 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 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 [7]. Она имеет открытый исходный код (опубликован под лицензией MIT [8]) и документацию на русском языке.

Эта библиотека также требует ручного манипулирования адресами, однако 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 данной 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.

1.4 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 [9].

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 [10] (опубликована под лицензией Apache 2.0 [11]). 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 [12; 13], которая не позволяет стирать и перезаписывать отдельные байты, а только их большие группы (обычно от 512 байт) [14]. 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 [12] 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.

1.5 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 механизма отложенной записи из библиотеки EEManager.

2 Разработка библиотеки менеджера EEPROM

2.1 Требования к библиотеке

2.2 Выбор необходимых программных средств

2.3 Разработка структуры библиотеки

2.3.1 Разработка внешнего интерфейса библиотеки

2.3.2 Переменные

2.3.3 Разделы памяти

2.3.4 Менеджер памяти

2.4 Разработка библиотеки

2.5 Использование библиотеки

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *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.
2. *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.
3. *Ryabinin K.*, *Chuprina S.* Adaptive Scientific Visualization System for Desktop Computers and Mobile Devices // Procedia Computer Science. — 2013. — Т. 18. — С. 722—731.
4. 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.
5. Arduino core for ESP8266 WiFi chip / I. Grokhotkov [и др.]. — URL: `https://github.com/esp8266/Arduino`.
6. *Arduino Software*. EEPROM Library. — URL: `https://docs.arduino.cc/learn/built-in-libraries/eeprom`.
7. *AlexGyver*. EEManager. — 2021. — URL: `https://github.com/GyverLibs/EEManager`.
8. *Massachusetts Institute of Technology*. MIT License. — 1988. — URL: `https://opensource.org/license/mit/`.
9. *Atmel Corporation*. AVR101: High Endurance EEPROM Storage. — 2002. — URL: `http://ww1.microchip.com/downloads/en/appnotes/doc2526.pdf`.
10. *Rosenberg P.* EEPROMWearLevel. — 2016. — URL: `https://github.com/PRosenb/EEPROMWearLevel`.
11. *The Apache Software Foundation*. Apache License, Version 2.0. — 2004. — URL: `https://www.apache.org/licenses/LICENSE-2.0`.

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

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

14. 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.

ПРИЛОЖЕНИЕ