Надежное хранение и обновление данных во флэш памяти микроконтроллера STM32

Часто возникает задача сохранить изменяемые данные, например конфигурацию, во флэш памяти микроконтроллера. Решение кажется простым, однако обеспечить надежность обновления данных при условии, что питание может отключиться в любой момент, оказывается весьма нетривиально, и даже использование контрольных сумм не решает проблему полностью. Из этой статьи вы узнаете

- как устроена флэш память
- к каким проблемам приводит выключение питания в момент записи или стирания
- как эти проблемы решаются

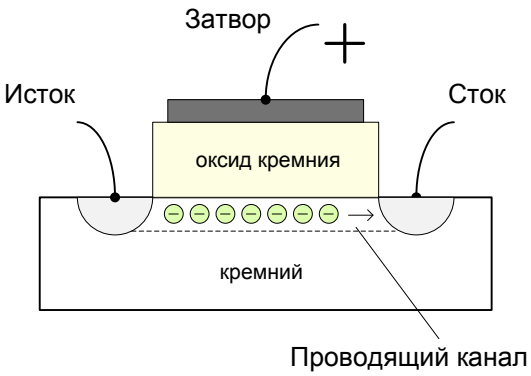
Классический подход к проблеме заключается в том, чтобы писать данные на флэш, сопровождая их контрольными суммами, чтобы можно было проверить целостность данных при чтении. Именно на таком подходе основана схема, предложенная автором для микроконтроллеров MSP430. Однако, она имеет 2 недостатка — сложность, обусловленная стремлением сэкономить память за счет хранения данных по частям, которые могут обновляться независимо, и отсутствие строгих гарантий целостности данных при отключении питания в момент записи. Под целостностью мы здесь понимаем следующее:

- данные оказываются либо записаны либо нет
- статус операции не меняется со временем, т.е. если данные записаны, они доступны всегда, если же нет, то они вдруг не появятся в будущем

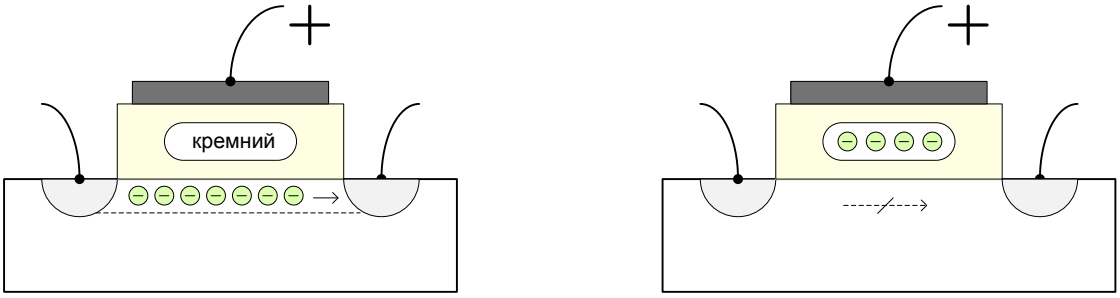
Поэтому, при разработке очередного устройства на базе STM32 было решено сделать вторую попытку строгого решения этой задачи, свободного от упомянутых недостатков. Но сначала мы рассмотрим, как устроена флэш память, чтобы понять суть проблем, с которыми мы имеем дело.

Как работает флэш память

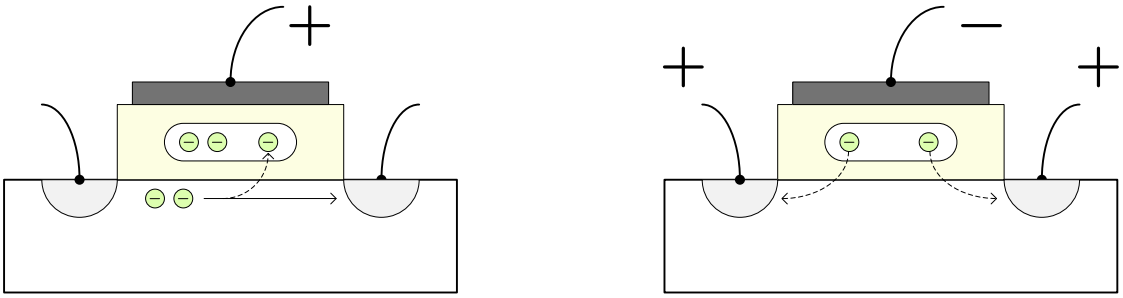
В основе флэш памяти лежит особая модификация транзистора с изолированным затвором (МОП-транзистора). Классический МОП-транзистор формируется на кремниевой пластине, покрытой слоем окисла, который играет роль изолятора. Поверх окисла напыляется электрод, называемый затвором. Подачей напряжения на этот электрод, можно управлять током, текущим между двумя электродами на кремниевой пластине — стоком и истоком. Происходит это потому, что положительный заряд затвора притягивает электроны и под затвором образуется проводящий канал из электронов. Если убрать напряжение с затвора, проводящий канал пропадает.



Во флэш памяти используются транзисторы с плавающим затвором. Они имеют изолированный от всего островок кремния в толще окисла между затвором и каналом. Если островок не заряжен, транзистор работает так же, как и обычный. Однако, если мы поселим на островке некоторое количество электронов, то они скомпенсируют положительный заряд затвора и проводящий канал пропадет.



Электроны попадают на плавающий затвор в процессе записи данных, туннелируя через изолятор. Этот процесс наглядно показан в фильме Чародеи — главное хорошо разогнаться, видеть цель и не замечать препятствий. Разгоняются электроны при пропускании тока в канале.



Запись Стирание

Со стиранием сложнее — ведь нам нужно не поселить электроны на затворе, а убрать их оттуда, значит разогнать их никак не получится. Поэтому мы просто формируем положительный потенциал в канале и ждем, когда электроны притянутся и протуннелируют в канал. Вот почему стирание занимает на несколько порядков большее время, чем запись. Для нашего STM32 это время от долей секунд до секунд. Более сложные устройства вроде SSD дисков поддерживают некоторый запас стертых транзисторов, но если они заканчиваются, время выполнения операций записи радикально увеличивается.

Чтобы сэкономить время, стирают память большими блоками — секторами. В случае STM32 минимальный размер — 16 килобайт — имеют 4 сектора, расположенные по младшим адресам. Записывать наш STM32 умеет по одному байту, по два, или четыре. Стертый транзистор читается как логическая единица. Соответственно, при записи мы поселяем электроны на затворы тех транзисторов, которые соответствуют логическим нулям в записываемых данных. Отсюда интересное наблюдение — мы можем выставлять в ноль биты в одном и том же байте один за другим, а не все сразу. Обратная операция — выставить нулевой бит в единицу — невозможна без стирания. При записи единичного бита содержимое памяти не изменяется.

Проблема стабильности чтений

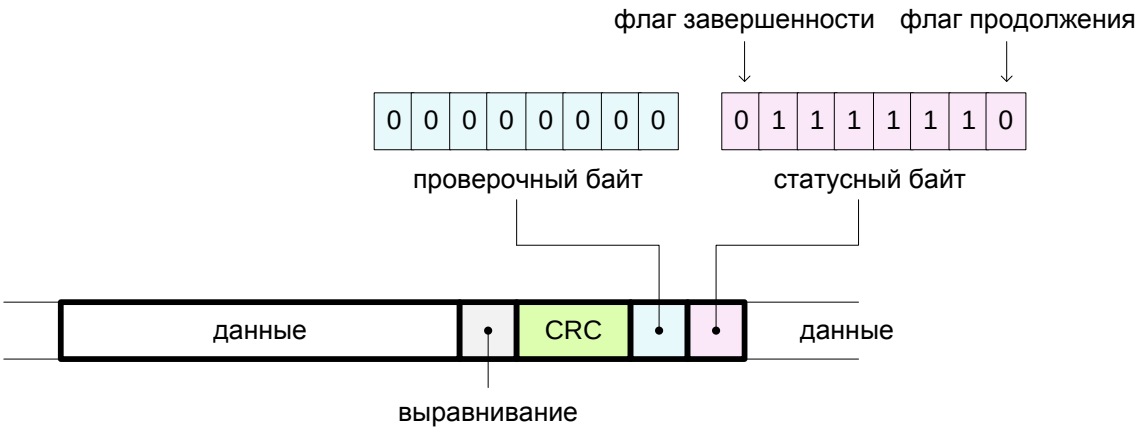
Что же произойдет, если мы выключим питание в момент записи данных? Понятно, что часть данных окажется незаписанной. А что призойдет с тем байтом или словом, которое мы записывали в момент выключения питания? При записи на плавающий затвор может попасть разное количество электронов. Много электронов читается как 0, мало — как 1, значит есть и некоторое пограничное количество электронов. Если до выключения питания на затвор попадет количество электронов, близкое к пограничному, то при чтении мы можем получать как 0, так и 1 в зависимости от совершенно случайных факторов. Со временем заряд будет стекать с затвора, так что вероятность прочитать 1 будет расти. Эта крайне неприятная особенность делает ненадежной даже схему с использованием контрольной суммы. Если питание отключилось в момент записи последнего слова нашего пакета с данными, которые мы можем сопроводить любым количеством проверочной информации, то мы можем сегодня прочитать наши данные, а завтра нет, или наоборот. Более того, нас ждут неприятности и при записи в область, которую мы считаем стертой, потому что она сегодня читается как все единицы — ведь завтра там могут проступить нули и испортить наши данные.

Аналогичные проблемы возникают и при выключении питания в момент стирания. При этом мы получаем совершенно непредсказуемое содержимое памяти с непредсказуемым поведением в будущем. Значит, такую ситуацию нужно уметь детектировать и проводить повторное стирание. Вот почему код, имеющий дело с записью во флэш память, должен писаться в состоянии обостренной паранойи, причем никогда нельзя сказать, достаточна ли степень этой паранойи или нет.

Реализация с гарантией целостности данных

Теперь мы готовы рассмотреть схему хранения данных, которая гарантирует целостность данных в смысле, обсуждавшемся выше. Поскольку STM не скупится на размер флэша, было решено упростить конструкцию, отказавшись от экономии, и использовать модель, где все данные объединены в единственную структуру фиксированного размера. При обновлении данных мы записываем всю структуру целиком. Разные версии данных записываются последовательно в предварительно стертую область флэш памяти. Актуальными считаются данные, записанные последними.

Система разбита на 2 уровня, предоставляющих различные гарантии относительно целостности данных. На нижнем уровне находится пул данных, позволяющий записывать данные последовательно в предварительно стертый сектор. Ниже показан формат пакета с данными на этом уровне.



После собственно данных следует выравнивание до 32 битного слова, после которого записывается контрольная сумма. После контрольной суммы следует проверочный байт, куда мы просто записываем нулевые биты. Эту часть пакета с данными мы записываем байт за байтом, поэтому, если при чтении мы видим хотя бы один нулевой бит в проверочном байте, мы можем быть уверены в том, что контрольная сумма записана правильно и ее содержимое не будет меняться со временем. Следующий байт после проверочного — статусный. Здесь есть нулевой бит, который маркирует пакет, как завершенный. Если при чтении мы обнаружили этот нулевой бит, это означает, что проверочный байт тоже был записан правильно и его содержимое не будет меняться со временем. То есть, мы можем считать данные полностью записанными и наше мнение не изменится со временем. Если при чтении мы не обнаружили флаг завершенности, но проверочный байт имеет нулевые биты, мы просто перезаписываем последние 2 байта. Если же в проверочном байте читаются все единичные биты, мы считаем, что данные не были записаны правильно независимо от контрольной суммы.

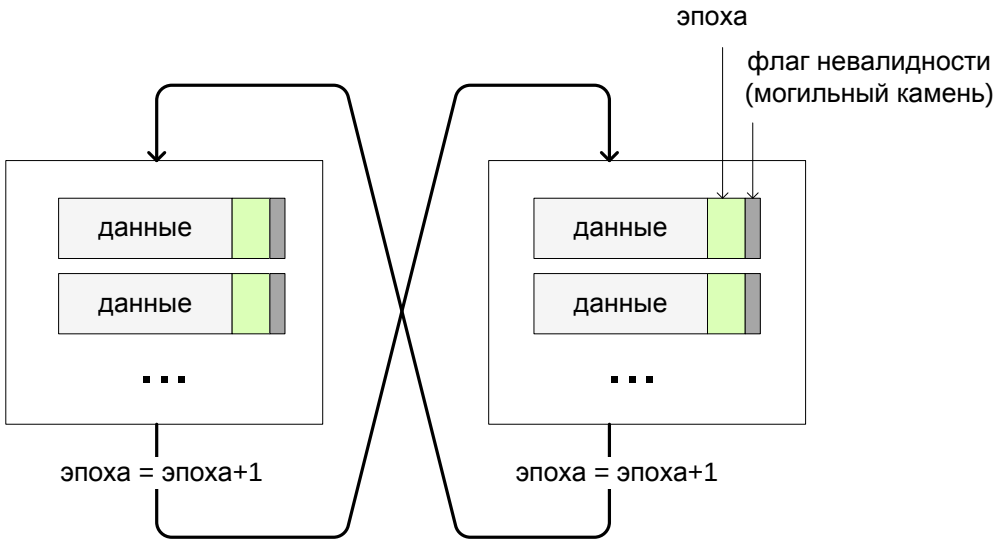
К чему такие сложности, может спросить любознательный читатель. Ведь мы можем просто переписать контрольную сумму, и она гарантированно не будет меняться со временем. Да, действительно, но нам придется делать это каждый раз. Цель состоит в том, чтобы

- Не делать лишних записей
- Иметь возможность понять, почему не совпадает контрольная сумма. Если при этом проверочный байт правильный, то несовпадение контрольной суммы однозначно указывает на то, что содержимое сектора повреждено или не конца стерто.

Второй статусный бит — флаг продолжения — позволяет определить, можно ли считать стертой память, из которой читаются все единицы. Перед тем, как записывать следующий блок данных, мы устанавливаем этот флаг (сбрасываем бит в 0). Если при чтении мы видим в этом бите 1, значит мы никогда не пытались писать в следующий байт. Ну а как быть, если сектор изначально пуст — можем мы считать его стертым или нет? Конечно же нет! Но с этим рецедивом паранойи справиться легче всего — мы просто сотрем его еще раз перед тем, как что то записать.

Итак, мы можем гарантировать, что будучи однажды прочитанными, данные будут читаться и дальше. Однако, с прочими свойствами надежного хранилища данных все не так радужно. Очевидна проблема с необходимостью периодически стирать сектор, когда там заканчивается место. Если при этом выключится питание, мы не только не запишем

новые данные, но и потеряем старые. Несколько менее очевидна проблема с неправильно записанными данными (в результате отключения питания во время записи). Мы не можем гарантировать, что со временем там не проступят отсутствующие биты и мы не начнем читать эти данные как правильные. Может показаться, что дополнительные статусные биты, маркирующие запись как неправильную, могут спасти положение, однако это не так. Ведь питание может пропасть и при записи этих дополнительных бит, и в итоге проблем станет только больше. Схема, описанная выше, успешно использует корректирующие записи только потому, что они записывают ровно те же данные, что и первоначальная запись, поэтому при любой последовательности отключений питания последняя успешная запись переводит флэш в стабильное состояние. Конечно, и в таком виде описанное хранилище может использоваться в приложениях не предъявляющих повышенные требования к надежности хранения. Но оказывается, что на базе двух хранилищ описанного типа можно создать более надежный вариант, лишенный описанных недостатков. Схема такого хранилища показана на следующем рисунке.



Два пула данных вышеописанного типа хранят пользовательские данные (в 2-х различных секторах флэша), дополненные служебным байтом. В нем хранится номер эпохи и флаг невалидности данных (называемый часто 'могильным камнем'). Если в текущем пуле заканчивается место, мы увеличиваем номер эпохи на единицу и начинаем писать в следующий. Отключение питания уже не грозит уничтожением всех наших данных, ведь мы не стираем пул с данными, которые были записаны последними. Номер пула, куда будет происходить очередная запись, равен младшему биту номера эпохи. На старте системы мы сравниваем номера эпох (на числовой окружности), чтобы определить пул, записанный последним. Проблема стабильности незавершенных записей решается тоже довольно просто. Если на старте мы обнаруживаем запись, которую считаем неправильной, то мы можем ее просто 'похоронить', сделав новую запись с актуальными данными, если они есть, либо с 'могильным камнем', если таковых нет.