IST Knowledge

AVR BootLoader в вопросах и ответах. Часть 1

Вы планируете создать собственный загрузчик? Тогда этот документ содержит ответы на многие вопросы, которые в связи с этим у вас появятся. Большинство понятий, охватываемых здесь не затронуты достаточно подробно в даташитах на микроконтроллеры, но тем не менее эти советы важны для проектирования надежных бутлоадеров. Большая часть представленной здесь информации можно найти на avrfreaks.net (http://www.avrfreaks.net/) разбросанной по разным веткам. Приведенные ниже вопросы упорядочены от простого к сложному и для создания жизнеспособного загрузчика, вы, вероятно, захотите понять ответы, по крайней мере на первые 11 вопросов. Для успешного овладения материалом предполагается, что вы уже знаете для чего нужен загрузчик, владеете програмированием на языке C, и знакомы с созданием обычных 8-разрядных приложений AVR. Этот документ основан на инструменте AVR-GCC с применением библиотек AVR-Libc и на типовом Makefile. Если вы используете иной инструмент, то все что описано в данном документе остается в силе, но и код Makefile примеры должны быть
соответствующим образом адаптированы под ваш инструмент.
Также обратите внимание, что примеры основаны на AT90USB162 и должны быть приспособлены к вашей AVR до их повторного использования.

С чего начать?

По сути, загрузчик — это просто обычное приложение AVR, которое расположено в специальной области флэш-памяти. В простейшей форме, приложение можно сделать загрузчиком AVR путем указания дополнительного флага компоновщику, который необходимо добавить в свой Makefile:

```
LDFLAGS += -Wl,--section-start=.text=0x3000
```

Этот флаг указывает начальный адрес байта из загрузчика. Вы можете найти это значение в даташите для своего типа AVR. Убедитесь, что вы используете адрес байта, а не адрес слова (большинство даташитов от Atmel определяют адреса флеш-памяти списком из слов, но некоторые списками байт). Вместо жесткого указанного начального адреса загрузчика я предлагаю объявить константу в файлеMakefile:

```
BOOTSTART = 0x3000
LDFLAGS += -Wl,--section-start=.text=$(BOOTSTART)
```

Что такое области NRWW и RWW?

Это описано в даташитах, но название этих терминов может запутать. NRWW и RWW обозначают разные области флэш-памяти микроконтроллеров и определяют что происходит с процессором в то время как к указанной области памяти происходит обращение на чтение или запись. Загрузчик всегда находится в в области no-read-while-write или NRWW (буквально не читать пока писать). Очень часто область NRWW резервируют полностью под загрузчик, но это вовсе не обязательно. Когда происходит стирание или запись в области памяти NRWW процессор останавливается, потому что задействуется режим «не-чтения» (no-read). Сами приложения, как правило, хранятся в области read-while-write (RWW) что буквально означает читать пока писать. Пока производится запись или стирание в области RWW процессор может продолжать работать до тех пор, пока процессор исполняет код расположенный в разделе NRWW (в области загрузчика). Пока область памяти RWW перепрограммируется любые попытки считать из нее данные будут возвращать 0xFF. Прежде чем область RWW снова станет доступна для чтения после программирования область необходимо переактивировать (подробнее об этом в вопрос № 3). Вот что вам действительно необходимо знать об особенностях областей RWW и NRWW:

- Загрузчик может перепрограммировать приложения, расположенные в области RWW, при этом исполнение загрузчика не прерывается;
- Загрузчик не может обновлять свой код также просто как код приложения;
- Приложения очень редко обновляют загрузчик;
- Приложения могут обновить себя, но лучше чтобы это делал загрузчик.

Как загрузчик обновляет приложение?

Каким образом загрузчик получит программу для перепрограммирования контроллера зависит от вас. Распространенными каналами связи являются UART и USB. Протокол обмена данными по каналу может быть собственный или стандартный, наподобие AVR109 или DFU. При реализации стандартных протоколов можно использовать уже существующие инструменты, такие как AVR Studio, работающий по протоколу AVR109 или Atmel's Flip работающий по протоколу DFU. Тем не менее эти стандартные протоколы как правило немного раздуты, а при реализации простого пользовательского протокола загрузчик как правило будет меньше размером и чище. Правда в таком случае вам понадобится также разработать собственное решение для передачи данных загрузчику. Обычно загрузчику передается за раз одна страница памяти. Загрузчику необходимо записать эти страницы в область флэш, предназначенную для приложения, то есть в RWW. AVR-Libc

предоставляет заголовочный файл<avr/boot.h> (http://www.nongnu.org/avr-libc/user-manual/group__avr__boot.html) который имеет все необходимое для этого. Эти функции очень хорошо документированы и сопровождены примерами. Несколько советов:

- Используйте удобный макрос `_safe` , чтобы не забыть применять ожидание готовности MCU;
- Убедитесь в том, что каждая страница стирается прежде чем в нее производится запись;
- `boot_page_fill` принимает адрес в байтах, но пишет слово за один раз. В цикле необходимо
- увеличить адрес на 2 байта;
- После программирования не забудьте снова включить область RWW с помощью макроса `boot_rww_enable`
- макрос. Делайте это до чтения или запуска приложения, иначе раздел RWW будет состоять из 0xFF;

Может ли загрузчик обновлять значения фьюзов?

С AVR это невозможно. Для этого вам потребуется внешний программатор.

Для чего нужен BOOTLOADER_SECTION из <avr/boot.h>?

Несмотря на свое название макрос BOOTLOADER_SECTION не является полезным при написании загрузчика. Этот макрос просто помогает вам переназначить позицию вашей функции в раздел NRWW флеш-памяти. Вероятно, для этого макроса более подходящим названием является что-то вроде NRWW_SECTION. Макрос BOOTLOADER_SECTION может потребоваться если приложению необходима функция перепрограммирования собственного приложения, т.к. такая функция будет работать только если она будет выполнена из разделаNRWW. BOOT_LOADER_SECTION на самом деле просто создает специальную секцию для кода с названием «.bootloader» (другое название лучше не использовать), а затем дополнительный флаг компоновщика позиционирует данную секцию где-то в неиспользуемой области памяти раздела NRWW. Такой подход не очень часто применяется на практике. Некоторые люди склонны использовать BOOTLOADER_SECTION для написания приложения и загрузчика сразу в одной программе, но это не очень хорошая идея. Загрузчик хорош в виде автономных программ, которые никак не зависят от самого приложения. Это будет более надежным решением, при том что если загрузчик будет отдельным приложением, то и создавать его тоже будет проще.

Как прошить загрузчик для микроконтроллеров?

Так как загрузчику не просто обновить самого себя, а приложения практически никогда не изменяют загрузчик, то вам потребуется внешний программатор для записи загрузчика в микроконтроллер. В качестве примера программаторов можно привести STK500, AVRISP, AVR Dragon, JTAGICE MKII и т.д. Режимы программирования вы будете задавать в зависимости от вашего типа AVR. Вам не нужно делать ничего особенного, чтобы прошить в микроконтроллер загрузчик. Если у вас есть рабочий внешний программатор, который может заливать прошивки и менять фьюзы, то он вполне подойдет и для заливки загрузчика. Программатор просто читает файл прошивки с загрузчиком и записывает его по указанному адресу. Программатору все равно, что вам посчастливилось писать в область NRWW флеш-памяти.

Как прошить сразу приложение и загрузчик?

Так как, надеюсь, вы создали отдельно приложение и загрузчик отдельно, то в конечном итоге у вас появится два отдельных шестнадцатеричных файла с прошивками (например, app.hex и boot.hex). Возникает вопрос: как залить их обе в AVR. Существует несколько вариантов:

В два этапа: Залейте загрузчик с помощью внешнего программатора как описано в Вопросе № 6. Возможно, вам при этом потребуется также установить требуемые значения фьюзов включая `BOOTSZ` и `BOOTRST` . После этого можно просто использовать обычный механизм связи загрузчика для передачи и прошивки в микроконтроллер приложения. **В один этап:** Объединить файлы app.hex и boot.hex в один и использовать внешний программатор для записи объединенного файла в микроконтроллер. Возможно, вам при этом потребуется также установить требуемые значения фьюзов включая `BOOTSZ` и `BOOTRST` .

Я думаю, что самый простой способ объединить шестнадцатеричные файлы с помощью командной строки `srec_cat` . Этот инструмент входит в набор инструментов `srecord` . Он поставляется с WinAVR и очень легко устанавливается на Unix-подобных ОС. Вот команда, которую вы должны при этом использовать:

```
srec_cat app.hex -I boot.hex -I -o combined.hex -I
```

Также вы можете вручную объединить файлы app.hex и boot.hex файлов с небольшими правками: Каждый шестнадцатеричный файл имеет одну последнюю запись, в которой говорится, что «файл заканчивается». Таким образом, после ручного объединения нужно отредактировать объединенный шестнадцатеричный файл — найти в нем запись окончания файла, расположенную в конце app.hex (после объединения где-то в середине файла) и удалить ее. Запись имеет тип 01. Байт типа записи в шестнадцатеричном формате Intel по порядку является 4-м байтом, поэтому запись на самом деле будет выглядеть примерно так: «:00000001FF». Вся линия (та что в середине, а не в конце файла) должна быть удалена.

Можно ли в загрузчике использовать прерывания?

Да, для этого достаточно сказать процессору, чтобы тот использовал вектор прерываний, расположенный в области загрузчика а не в области приложения. Для этого где-то в начале кода загрузчика (до использования прерываний), надо добавить строки вроде этих:

```
MCUCR = (1<<IVCE);
MCUCR = (1<<IVSEL);
```

После чего во время прерываний программный счетчик будет переведен в соответствующее положение согласно таблице прерываний загрузчика. Для нормальной работы, приведенная выше последовательность кода должна быть скомпилирована с включенной оптимизацией, также вручную требуется удостовериться, что в результирующую сборку записаны команды, выполняемые за 4 машинных цикла.

Что должно загружаться в первую очередь — загрузчик или приложение?

Почти всегда лучше всего будет установить фьюз BOOTRST для того чтобы загрузчик запускался первым после перезагрузки микроконтроллера. Когда загрузчик запускается после перезагрузки первым вы сможете перезалить программу независимо от того поврежден ли код приложения или он вовсе отсутствует.

Для лучшей работы загрузчиков необходимо чтобы они представляли собой небольшие, надежные и редко изменяемые программы. С того дня как вы отправили свое устройство в эксплуатацию вам действительно больше не захочется изменить на нем загрузчик. Приложение же наоборот требует больше свободы в плане обновления. Если в приложении возникает ошибка, приводящая к отказу или зависанию, или во время заливки прошивки произойдет сбой питания надежный загрузчик, который будет запускаться первым сможет без проблем исправить данную ситуацию. Если после сброса устройства запускается приложение, то для того чтобы исправить проблемы вы можете использовать внешний программатор. Но если у устройства не предусмотрен интерфейс для подключения программатора, то устройство скорее всего придется выкинуть или перепаивать.

Как узнать загрузчику когда запускать приложение?

Есть много возможностей. Если устройство имеет кнопку или другой механизм ввода, вы можете послать соответствующий сигнал нажатием. Тогда загрузчик, как правило, запустит приложение сразу. А в случает если кнопка будет не нажата во время сброса, то загрузчик продолжает исполнение. Это пример того как работает загрузчик чипа STK500. Другим распространенным решением для загрузчика является проверка внешнего канала связи на специальный символ во время запуска. Опять же, загрузчик, как правило, запустит приложение сразу, а продолжает исполняться только после проверки внешнего канала связи и

получения определенного ответа или поиска ожидаемых данных. Примером такого подхода является Atmel Butterfly.
В нашем случае, ни одно из этих решений не подходят. Наше устройство не имеет кнопок и мы хотели бы чтобы приложение запускалось в обычных условиях очень быстро. Единственный внешний канал связи является USB, к сожалению, устройство USB требует некоторое время (в процессор выражении) прежде чем будет возможен обмен данными с хостом.
Так вот что мы сделали: Если предположить, что AVR имеет EEPROM, вы можете хранить в нем байтовое значение, которое указывает, когда загрузчик должен продолжать работать, а когда запустить приложение. У этого подхода два этапа. Сначала где-то в начале кода загрузчика необходимо добавить что-то вроде этого:

```
// Bootloader code
const uint8_t app_run = eeprom_read_byte(ADDR_APP_RUN);
if(app_run == APP_RUN)
{
    // In case the app is faulty, clear the eeprom byte so that
    // the BL will run next time. A properly running app should
    // set this back to APP_RUN.
    eeprom_write_byte(ADDR_APP_RUN, 0xFF);
    run_application();
}
// Only run the bootloader once, then go back to the app
// (comment out the next line during app development)
eeprom_write_byte(ADDR_APP_RUN, APP_RUN);
```

Затем где-то подальше в приложении, в той части кода, при обработке которого вы уже точно уверены, что приложение работает правильно добавьте следующее:

```
// Application code
eeprom_write_byte(ADDR_APP_RUN, APP_RUN);
```

Это работает следующим образом: если приложение было с успехом исполнено ранее, то загрузчик будет читать
байт APP_RUN подготавливать к переходу к исполнению приложения. Однако незадолго до запуска приложения байт APP_RUN очищается. Приложение затем снова
пишет байт APP_RUN в тот момент когда она считает, что она работает правильно. Таким образом последовательность повторяется. Если с приложением до сброса произошел сбой до того момента как оно производит сброс байта APP_RUN, то после перезагрузки загрузчик будет оставаться на исполнении. Если вы закомментируете указанную выше строку в коде загрузчика, то загрузчик будет оставаться на исполнении только один раз, а в следующие разы после сброса будет запускать приложение. Я также предлагаю способ принудительно запустить загрузчик из приложения. Вам просто необходимо очистить байт APP_RUN и предотвратить его перепись позже. Мы реализовали это в нашем приложении, а делается это по команде, отправленной через USB с программы, запущенной на компьютере. Один из недостатков такого подхода является то, что EEPROM компании Atmel на большинстве контроллеров имеет ресурс в 100000 циклов записи/стирания. Поскольку этот подход производит два цикла стирания/записи во время каждого пуска, то получается ресурс

устройства ограничен 50 000 циклами перезагрузки. Для нашего приложения я прикинул, что продолжительность жизни устройства составляет в среднем 13,7 лет при 10 перезагрузках день. Этот предел можно легко расширить если износ ячеек EEPROM распределить.

Как загрузчик запускает приложение?

Существует на самом деле только один способ, вы должны переместить программный счетчик в начало приложения (которое, как правило, является вектором сброса). Если у вас есть приложение на основе
обычной библиотеки AVR-Libc, то в начале вашего приложения будет расположен рантайм языка с, который инициализирует стек и глобальные переменные, а затем
располагается ваша функция «main». Вы можете переместить счетчик программы с помощью ассемблерной команды jump или вызовом функции, расположенной по нулевому адресу. Я выбрал команду jump:

```
asm("jmp 0000");
```

Но это еще не все. Когда вы непосредственно запустите приложение указанным способом микроконтроллер не будет сброшен в исходное состояние как это происходит после перезагрузки. Значения регистров не очищаются, а периферия не выключается. Есть два распространенных способа справиться с этим. Первый подход заключается в намеренном провоцировании сброса с помощью сторожевого таймера. Это приведет к сбросу микроконтроллера в первоначальное состояние и возобновит загрузчик (если у вас установлен бит BOOTRST). Если условия, что вызвало загрузчик больше нет (например, вы больше не удерживаете кнопку), то загрузчик должен немедленно запустить приложение. Необходимо начать исполнение приложения как можно быстрее, пока изменения коснулись как можно меньше ресурсов микроконтроллера. Вот пример:

```
Disable_interrupt();
Wdt_change_16ms();
while(1);
```

Потом где-то ближе к началу загрузчика сделать переход на адрес 0.
Второй подход состоит в том, что загрузчик прежде чем перейти к запуску приложения, очищает за собой все и переходит к исполнению программы сразу без перезагрузки. Линейка загрузчиков Atmel USB работает именно таким образом. С применением USB-загрузчика имеет смысл отключить USB и вернуть вызов прерываний обратно в область приложения перед переходом к приложению:

```
Disable_interrupt();
// Shutdown USB cleanly
Usb_detach();
Usb_disable();
Stop_pll();
// Put interrupts back in app land
MCUCR = (1<<IVCE);
MCUCR = 0;
// Run the applicat
```

Приложение затем должно явно сконфигурировать все ресурсы что оно собирается использовать.

Продолжение: AVR BootLoader в вопросах и ответах. Часть 2 (/informacionnaa-lenta/avr-bootloader-v-voprosah-i-otvetah-2/)

Микроконтроллеры AVR (https://istknowledge.wordpress.com/category/mikrokontrollery-avr/)

Zashibon
30 августа, 201219 декабря, 2019

AVR /
bootloader /
C /
загрузчик /
уроки