

## Что такое Fuse и Lock биты?

Слово "Fuse" (Фьюз) с английского переводится как "предохранитель", а слово "Lock" - блокировка. Fuse и Lock биты в AVR микроконтроллере можно представить себе как внутримикросхемные переключатели, которые в одном состоянии контактов дают 1, а в противоположном - 0.

Здесь также неплохо работает ассоциация со словом "Fuse" (плавкий предохранитель): если нить плавкого предохранителя цела, то это - 1 (ток течет), а если она перегорела (вообразим что мы ее программно сожгли, установили фьюз) - 0 (ток не течет).

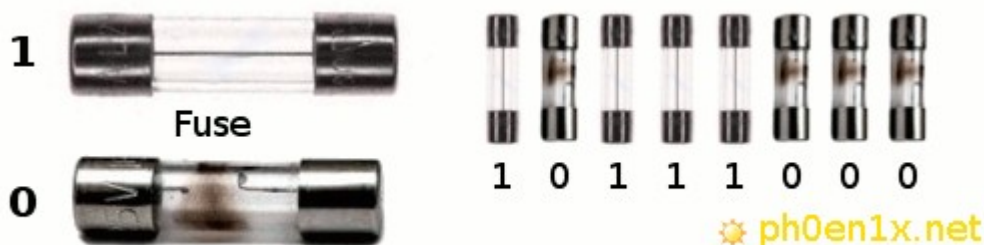


Рис. 1. Что такое фьюзы (fuses), пример кодирования с плавкими предохранителями.

Важно помнить что для AVR микроконтроллеров все фьюзы и биты блокировки имеют следующие значения:

- 1 - не установлен (не запрограммирован);
- 0 - установлен (запрограммирован).

Начиная работать с AVR МК многие не могут понять зачем было ломать принятые и вполне логичные нормы что 1=установлен, а 0=не установлен. Но если исходить из названия "fuse", с пониманием как работает предохранитель, то все становится закономерно и вполне логично!

Итак, сброшенный в ноль бит является активным (установленным). Чтобы в будущем не забывать эту особенность, достаточно вспомнить пример с плавкими предохранителями, который был приведен выше.

## Для чего нужны биты конфигурации и блокировки

В AVR микроконтроллерах Fuse и Lock биты содержатся в специально отведенной для этого области памяти. При помощи Fuse-битов мы можем установить различные режимы работы микроконтроллера, его пинов, задать источник тактового сигнала и его параметры, превратить вывод "Reset" в обычный порт ввода-вывода, а также многое другое.

Некоторые названия и описания часто используемых Fuse-битов:

- **RSTDSBL** (ReSeT DiSaBLe) - "запретить ресет", превращает пин для сброса МК в обычный порт ввода/вывода;

- **CKSEL0..3** (ClocK SElect) - четыре бита для установки параметров и источника тактового сигнала МК (внешний кварц, внутренний RC-генератор, делитель частоты и т.п.);
- **CKDIV8** (ClocK DIVision 8) - если этот бит установлен то тактовая частота от внутреннего RC-генератора будет делиться на 8;
- **CKOPT** (ClocK OPTimization) - задает размах сигнала (амплитуду) с выходного тактового генератора, оптимизация потребляемого тока, влияет на помехоустойчивость;
- **SUT0..1** (Start Up Time) - установка временной задержки запуска программы после подачи питания или перезапуска МК;
- **SPIEN** (Serial Programming Interface ENable) - разрешение/запрещение программирования МК через последовательный программный интерфейс;
- **JTAGEN** (JTAG ENable) - разрешает/запрещает использование JTAG интерфейса;
- **EESAVE** (EEProm SAVE) - если этот бит установлен, то содержимое энергонезависимой памяти будет сохранено после стирания кристалла (опция -е в [AVRDude](#));
- **WDTON** (Watch Dog Timer ON) - отключение программного управления сторожевым таймером, запуск таймера автоматически при подаче питания на МК;
- **BODEN** (Brown-Out Detection ENabled), **BODLEVEL** (Brown-Out Detection LEVEL) - биты для включения и настройки мониторинга за напряжением питания МК;
- **BOOTRST** (BOOT ReSeT) - выполнять запуск через загрузчик (Boot Loader), микроконтроллер начнет выполнение программы не с адреса 0x0000 (по умолчанию), а с адреса где расположен загрузчик.

Важно заметить что при установке фьюза RSTDSBL теряется возможность перепрошивки МК через ISP-интерфейс. Тем не менее, с использованием высоковольтного (+12В) параллельного программатора перепрошивка все же возможна.

Биты блокировки позволяют установить режимы доступа (запись/чтение) к внутренней Flash-памяти и/или EEPROM, причем направление доступа можно ограничить как изнутри микроконтроллера, так и снаружи (при использовании [ISP-интерфейса](#)). Данная возможность может быть полезна для защиты от копирования/изменения вашей программы, а также хранящихся в энергонезависимой памяти данных.

## Структура конфигурационного и блокировочного байта

Фьюзы (фьюз-биты) содержатся в трех байтах:

- Fuse Low Byte - младший байт;
- Fuse High Byte - старший байт;
- Fuse Extended Byte - байт с опциями расширенных функций.

Блокировочные биты (Lock Bits) микроконтроллера расположены в отдельном блокировочном байте. Не редко можно слышать что их относят к фьюзам, но это не так, не стоит путать, тем более что названы они так чтобы можно было их четко различать.

У каждой модели микроконтроллера свой набор доступных к изменению фьюзов и блокировочных битов, поэтому чтобы более детально ознакомиться со структурой байтов содержащих эти биты в интересующем вас МК, нужно обратиться к официальной документации (даташиту).

**Ниже приведен пример структуры Fuse-байтов для микроконтроллера ATTiny13.** В первой строке идет номер бита, во второй - название, а в третьей - значение по умолчанию. Помним что значение 0 = бит установлен (запрограммирован).

Младший Fuse-байт (Fuse Low Byte):

7	6	5	4	3	2	1	0
SPIEN	EESAVE	WDTON	CKDIV8	SUT1	SUT0	CKSEL1	CKSEL0
0	1	1	0	1	0	1	0

Как видим, интерфейс последовательного программирования (SPI) по умолчанию разрешен, частота внутреннего тактового генератора делится на 8 (CKDIV8), в качестве источника тактового сигнала используется внутренний RC-генератор (CKSEL0).

Старший Fuse-байт (Fuse High Byte):

7	6	5	4	3	2	1	0
-	-	-	SELFPRGEN	DWEN	BODLEVEL1	BODLEVEL0	SRTDISBL
1	1	1	1	1	1	1	1

В данном фьюз-байте все биты неактивны (не запрограммированы).

Структура блокировочного байта (Lock Byte):

7	6	5	4	3	2	1	0
-	-	-	-	-	-	LB2	LB1
1	1	1	1	1	1	1	1

Установка бита под номером 0(LB1) в 0(установлен) запретит программирование внутренней Flash и EEPROM памяти. Если еще дополнительно установить бит под номером 1(LB1) то это дополнительно заблокирует возможность считывания данных с памяти МК ATTiny13.

Данную информацию я легко получил, скачав даташит на ATTiny13 в формате PDF, для этого достаточно выполнить поиск в поисковой системе Google или другой по запросу "ATTiny13 datasheet download".

Открыв документ ищем раздел "Memory Programming". Как правило PDF-файл с даташитом на микроконтроллер занимает от 1 до 10МБ. Документы от ATMEL хорошо структурированы, содержат хорошую внутреннюю навигацию со ссылками и содержанием, очень удобно искать нужную информацию.

## Работа с фьюзами и Lock-битами

Выполнять установку фьюз-битов и битов блокировки нужно очень и очень аккуратно, с уверенным пониманием того что и для чего выполняется! Невнимательность и лень в изучении документации может стать причиной неработоспособности используемого AVR микроконтроллера.

Самая безопасная стратегия при записи фьюзов и блокировочных битов:

1. Читаем значение нужного байта из МК
2. Устанавливаем в полученном байте нужные биты
3. Записываем результирующий байт в МК

Почему именно так, а не сразу запись в МК? - потому что преследуя некоторую цель с установкой одного бита, можно нарушить состояние других битов, что может повлечь за собой, к примеру переключение МК на источник тактового сигнала, который не предусмотрен и микроконтроллер просто не запустится.

Важно помнить что в случае выполнения операции стирания (опция "-e" в AVRDUDE) все Fuse и Lock биты будут восстановлены по умолчанию, а записанная во Flash-память программа уничтожена.

## Пример установки Fuses с использованием AVRDUde

К примеру, нам нужно убрать [фьюз CKDIV8 в младшем байте](#) чтобы микроконтроллер ATTiny13 начал работать на частоте 8МГц вместо 1МГц.

Попробуем выполнить чтение байта с фьюзами, изменим один бит (установим фьюз), а потом выполним запись в МК при помощи AVRDUde на примере малыша ATTiny13. Если возникнут вопросы при работе с дудкой - читаем [документацию по AVRDUde](#).

Итак, читаем документацию по микроконтроллеру и [подключаем его к программатору](#).

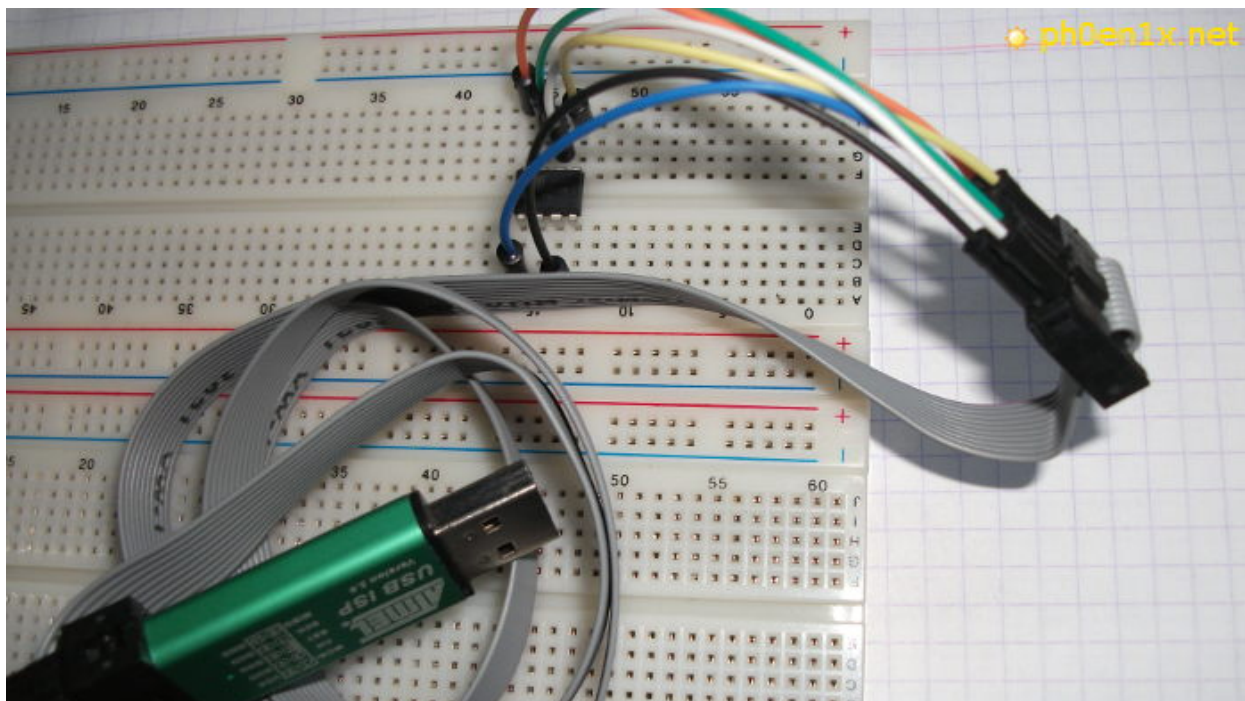


Рис. 2. Микроконтроллер ATTiny13 подключен к программатору USB ISP.

По умолчанию, при запуске AVRDUde выводит на экран значения всех трех байтов с фьюзами, достаточно подключить МК к программатору и выполнить команду с нужными настройками (в данном случае МК - ATTiny13, программатор - USB ISP):

```
avrdude -p t13 -c usbasp
```

Bash

Вывод команды:

```
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
```

```
avrdude: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.00s
```

```
avrdude: Device signature = 0x1e9007
```

```
avrdude: safemode: Fuses OK (E:FF, H:FF, L:6A)
```

```
avrdude done. Thank you.
```

None

Как видим, значение расширенного Fuse-байта (E) - FF (0xFF), старшего (H) - FF, младшего (L) - 6A (0x6A).

Также значение любого из фьюз-байтов можно считать и сохранить в файл. К примеру, запишем значение младшего фьюз-байта в файл "lfuse.txt" и выведем его содержимое на экран:

```
avrdude -p t13 -c usbasp -U lfuse:r:lfuse.txt:h  
cat lfuse.txt
```

Bash

Значение младшего байта с фьюзами сейчас - 0x6A, что в двоичном представлении равно 01101010 - именно то, что я приводил в [структуре младшего фьюз-байта для ATTiny13](#) по умолчанию.

Теперь нам нужно изменить значение бита SKDIV8 в байте 01101010 на 1 (сбросить его), получаем двоичное число 01111010, которое равно 0x7A. Раньше я уже описывал как [работать с битами](#), а также [как удобно переводить числа из одной системы счисления в другую](#).

Записываем новое значение в байта с фьюзами в микроконтроллер:

```
avrdude -p t13 -c usbasp -U lfuse:w:0x7a:m
```

Bash

Значение параметра "-U" - "lfuse:w:0x7a:m" означает что выполняем операцию с памятью (-U), а именно с младшим байтом который содержит фьюзы (lfuse), будем записывать (:w) значение "0x7a", указав его прямо в командной строке (:m).

Результат выполнения команды:

```
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
```

```
avrdude: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.00s
```

```
avrdude: Device signature = 0x1e9007
avrdude: reading input file "0x7a"
avrdude: writing lfuse (1 bytes):
```

```
Writing | ##### | 100% 0.01s
```

```
avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0x7a:
avrdude: load data lfuse data from input file 0x7a:
avrdude: input file 0x7a contains 1 bytes
avrdude: reading on-chip lfuse data:
```

```
Reading | ##### | 100% 0.00s
```

```
avrdude: verifying ...
avrdude: 1 bytes of lfuse verified
```

```
avrdude: safemode: Fuses OK (E:FF, H:FF, L:7A)
avrdude done. Thank you.
```

None

Как видим, все удачно записалось и прочиталось - "Fuses OK (E:FF, H:FF, L:7A)"! Теперь микроконтроллер должен работать на частоте 8МГц.

## Пример установки Lock Bits с использованием AVRDUde

Для **установки битов блокировки** в микроконтроллере используем ту же самую стратегию. Считываем текущее значение байта с битами блокировки (Lock Bits) в файл lock.txt:

```
avrdude -p t13 -c usbasp -U lock:r:lock.txt:h
cat lock.txt
```

Bash

Результат работы команд:

```
avrdude: warning: cannot set sck period. please check for usbasp firmware
update.
avrdude: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.00s
```

```
avrdude: Device signature = 0x1e9007
avrdude: reading lock memory:
```

```
Reading | ##### | 100% 0.00s
```

```
avrdude: writing output file "lock.txt"
avrdude: safemode: Fuses OK (E:FF, H:FF, L:7A)
avrdude done. Thank you.
```

0x3f

None

Текущее значение байта с Lock Bits - 0x3F, что в двоичном представлении равно 00111111. В даташите на ATTiny13 указано что для изменения значимы только два бита - 0011111**1**.

Установив эти биты в 0 мы их активируем, чем заблокируем возможность программирования и чтения внутренней Flash + EEPROM. В результате наш байт будет выглядеть вот так - 00111100, а в шестнадцатеричном представлении - 0x3C.

Для записи Lock-байта нужно выполнить следующую команду:

```
avrdude -p t13 -c usbasp -U lock:w:0x3c:m
```

Bash

Будьте предельно внимательны с этой командой, поскольку изменение Lock-битов может повлечь за собою необратимые последствия!

## Выполняем расчет битов в байте вручную

Расчет значения байта с установленными нужными в нем битами для AVR микроконтроллера можно выполнять в различных онлайн-калькуляторах, ссылка на один из таких приведена в конце статьи.

Тем не менее, важно уметь выполнить такие расчеты "вручную", то есть изучив документацию по микроконтроллеру самому составить нужную последовательность бит и перевести состоящий из них байт в двоичный (Binary) или шестнадцатеричный (HEX) вид.

Допустим что изучив [структуру фьюзов микроконтроллера ATTiny13](#) нам нужно в младшем Fuse-байте выполнить следующее:

- Сбросить бит 4 (CKDIV8) - установить в 1;
- Активировать бит 6 (EESAVE) - установить в 0.

Итак, у нас есть новенький микроконтроллер и значение младшего Fuse-байта по умолчанию - 0x6a (это число в HEX-формате мы узнали считав значение байта при помощи AVRDUDE, а также из документации).

Теперь нам нужно его перевести в двоичное представление, это можно сделать используя специализированный математический калькулятор, а можно просто разделить значение на две части и воспользоваться табличкой что ниже.

**Binary** 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001

**HEX** 0 1 2 3 4 5 6 7 8 9

**Binary** 1010 1011 1100 1101 1110 1111

**HEX** A B C D E F

Разделяем значение "6A" (0x6a) на две части - "6" и "A", ищем в табличке соответствующие бинарные представления - 0110 (6) и 1010 (A). Значение 0x6a в двоичной системе счисления - 01101010.

Теперь установим 4-й бит в 1, а 6-й бит - в 0: [7й бит->] 00111010 [<-0й бит].

Разделяем полученное бинарное представление 00111010 на две равные части: 0011 и 1010. Конвертируем эти части в HEX, используя табличку выше: 0011 = 3, 1010 = A.



Соединяем полученные цифры в шестнадцатеричном формате: 0011 1010 = 3A. Получается, для того чтобы сбросить фьюз 4 (CKDIV8) = 1, а также активировать фьюз 6 (EESAVE) = 0 нужно записать в младший фьюз-байт значение "0x3a" (0x - значит что значение представлено в HEX-формате).

## Заключение

В действительности, работа с Fuses в AVR-микроконтроллере - это очень просто. Главное здесь не спешить и не лениться сверяться с официальной документацией. Всю рутину по записи и считыванию значений берет на себя "швейцарский нож из мира AVR" по имени AVRDude, нам лишь остается только правильно выполнить расчет нужного байта с фьюзами.

Удачного и результативного программирования!