ELECTRONS.PSYCHOGENIC.COM NEWS RAVES & RANTS GAMES CONTACT US

HOME ACCOUNT PRIVATE MESSAGE

Main Menu
- Home
- Latest News
- Raves & Rants
- Articles
  - Add Article
  - My Articles
  - Newest
  - Top Rated
  - Most Popular
- Frozen Bubble
- Flash Games

Login
Username:
Password:
User Login
Secure Login
Lost Password?

Register now!

Electrons :: Articles :: AVR :: AVR Fuses HOWTO Guide AVR Go

AVR Fuses HOWTO Guide

Description: Details the nature and function of AVR fuses and describes how to determine and program new values for a chip.

- 1. **AVR Fuses HOWTO Guide**

AVR Fuses HOWTO Guide

For an AVR microcontroller, the closest thing to a config file is its set of **fuses**. Fuses are used to configure important system parameters.

While you may completely ignore a number of the more esoteric options most of the time, you will undoubtedly need access to a few of these settings (such as the clock source fuse bits, which allow you to specify the source and/or speed of the chip clock) in the course of hacking AVRs.

You will need to know how to program these fuse bytes in order to get the most out of your microcontroller (or get it working at all!) and this simple tutorial will get you started right.

Fuse Bits and Bytes

As a quick reminder (or a crash course, if all this is new to you): numbers are usually represented in decimal notation (e.g. "13" is thirteen) but others representations are possible, such as in binary where thirteen is "1101" or hexadecimal where the same value is represented as 0x0D. Any number below 256 can be represented using only 8 binary digits (this is where "bits" come from, they are just **bi**nary digi**ts**), for instance "1111 1111" is equal to 255 (in regular decimal notation). Each such group of 8 bits is called a *byte*.

Members of the AVR family can have one or more fuse bytes. How many, and what they tell the chip to do, depends on the specific microcontroller (e.g. a ATMega8 has both high and low fuse bytes, for a total of 16 configuration bits, while an ATtiny12 has only eight fuse bits).

As a quick reference, here are a few PDFs which provide the list of low, high and extended fuse bytes for AVR microcontrollers. Though some of the bits are common to multiple MCUs, the specific functions and settings to use for each should be verified in the relevant datasheet.

How to read the datasheet

The datasheet for each microcontroller contains (excruciatingly?) detailed information on the fuse bits available and their function, but often leaves a bit to be desired in terms of an overview of the actual process of setting these fuses. Matters are made worse by the fact that, for AVR micros, having a "**1**" in a bit means it is *unprogrammed* while having a "**0**" is considered to be *programmed* and that the documentation will often use language such as "... clear bit 7 to enable blah blah" which can leave you wondering whether this means you should set it to 0 or not.

To find the available fuses for your microcontroller, do a search in the relevant datasheet for "Memory Programming", "LockBits" "Fuse low byte" or something similar.

Programming Fuse Bits

The process of programming fuse bits involves:

- Reading the datasheet to discover the location and settings for the bits of interest,
- Determining the byte value for the affected fuse byte(s),
- Actually programming the fuses on the chip.

We'll go through an example of the process and covers each step.

Bits of Interest

The ATmega8 ships with safe default settings for its clock source and startup time, which specify that an internal 1 MHz RC oscillator is to be used. This ensures that you can get the chip working with minimal setup but lets assume you want to take advantage of a faster/more stable/more useful clock through the use of an external crystal.

By checking the *Clock Sources* section of the datasheet, you can see that the clocking options are set using 4 *CKSEL* bits that specify the type of clock source the chip will use, and the *CKOPT* bit (which tells the chip to pay attention to our CKSEL bits and influences startup time).

The actual bits you use depend on the hardware you've got. Here we'll say we're using a 2MHz crystal. In this case, we determine that we should set CKSEL3..0 (this notation, used in the datasheets, indicates CKSEL bits 3 to 0, starting at 3) to 1100 and we should also ensure the CKOPT bit is 1.

Now that we know which bits we wish to set we need to know where they fit in, in the grand fuse byte scheme of things. To do this, have a look at the following tables, which indicate the position and function of each bit in the low and high fuse bytes (for the ATmega8).

Fuse Low Byte

Bit Number	Name	Description	Default
7	BODLEVEL	Brown out detector trigger level	1 (unprogrammed)
6	BODEN	Brown out detector enable	1 (unprogrammed, no BOD)
5	SUT1	Select start-up time	1 (unprogrammed)
4	SUT0	Select start-up time	0 (programmed)
3	CKSEL3	Select Clock source	0 (programmed)
2	CKSEL2	Select Clock source	0 (programmed)
1	CKSEL1	Select Clock source	0 (programmed)
0	CKSEL0	Select Clock source	1 (unprogrammed)

News
- Russia tries to ban human DNA expo... (2007/5/30) Russia has banned the shipment of medical specimens abro...
- Cookies In Your Head (2007/2/7) Current iris scanning systems require a person to stand ...
- Trans Goes Straight To Tummy (2007/2/7) Eating a diet consisting largely of fast food could caus...
- Folic Acid as Cancer Prevention (2006/6/12) Folic acid supplements may prevent cancer progression an...
- Don't Bother Touching That Dial (2006/4/18) Philips would like to take advantage of Multimedia Home ...
- Human Lab-Grown Organ Success (2006/4/4) The first human recipients of laboratory-grown organs we...
- Dogs beat CATs at detecting cancer (2006/1/7) Dogs with three weeks of training can best the latest CA...
- Stem Cells Medicate Brain (2005/12/15) Scientists have managed to protect and regenerate the pa...
- NAC vs. Alzheimer's (2005/12/15) A study conducted at the San Francisco VA Medical Center...
- Buckyballs Could Disrupt DNA (2005/12/15) A new study, conducted at Vanderbilt by chemical enginee...

Fuse High Byte			
Bit Number	**Name**	**Description**	**Default**
7	RSTDISBL	Select if PC6 is I/O pin or RESET pin	1 (unprogrammed PC6 is RESET-pin)
6	WDTON	WDT always on	1 (unprogrammed, WDT enabled by WDTCR)
5	SPIEN	Enable Serial Program and Data Downloading	0 (programmed, SPI programming enabled)
4	CKOPT	Oscillator options	1 (unprogrammed)
3	EESAVE	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
2	BOOTSZ1	Select Boot Size	0 (programmed)
1	BOOTSZ0	Select Boot Size	0 (programmed)
0	BOOTRST	Select Reset Vector	1 (unprogrammed)

As you can see, our CKSEL settings are to be set in bits 0 to 3 of the low fuse byte, while CKOPT is bit 4 of the high fuse byte. Even though we only wish to affect the CKSEL and CKOPT bits, we must program the **entire** low and high fuse bits. We want to set:

```
Low fuse bits: XXXX 1100
High fuse bits:XXX1 XXXX
```

without disturbing the bits in the "X" positions.

Byte Values

Determining the byte values required for the low and high fuses first involves discovering the current values for the "X" bits above. Your hardware programmer can allow you to do this.

If you are using uisp you'd use something like:

```
$ uisp -dprog=stk500 -dserial=/dev/ttyS0 -dspeed=115200 -dpart=atmega8 --rd_fuses

Fuse Low Byte = 0xE1
Fuse High Byte = 0x91
Calibration Byte = 0x00 -- Read Only
Lock Bits = 0xff
```

You can convert these hex values to binary (for use below) using any decent calculator program.

If you've been using avrdude, you can extract the high and low fuse bytes like so (adjusting the programmer, device and port options to match your hardware and setup):

```
$ avrdude -c stk500 -p m8 -P /dev/ttyS0 -U hfuse:r:high.txt -U lfuse:r:low.txt

  avrdude: Device signature = 0x003d04

  avrdude: reading hfuse memory:
  Reading | ################################################## | 100% 0.01s
  avrdude: writing output file "high.txt"

  avrdude: reading lfuse memory:
  Reading | ################################################## | 100% 0.00s
  avrdude: writing output file "low.txt"

  avrdude done.  Thank you.
```

Avrdude doesn't make getting to the fuse info as simple... the bytes are actually saved raw in the high.txt and low.txt files. You can use a hex editor to look at the value of the byte in each file. If you are lucky enough to use a Unix workstation, like Linux, then you can just run this one-liner for each file to view the binary representation of the bytes:

```
$ od -d high.txt  | head -1 | sed -e 's/0000000 *//' | \
  xargs -i perl -e '$str= unpack("B32", pack("N",{})); $str =~ s/.*([01]{4})([01]{4})$/$1 $2/; \
  print "$str\n";'

1101 1001



$ od -d low.txt  | head -1 | sed -e 's/0000000 *//' | \
  xargs -i perl -e '$str= unpack("B32", pack("N",{})); $str =~ s/.*([01]{4})([01]{4})$/$1 $2/; \
  print "$str\n";'

1110 0001
```

Note that the Atmel documentation names the bits in order of significance, so from bit 7 to 0:

```
low fuse bits: 1110 0001
low fuse pos.: 7654 3210
```

So in our example, CKSEL3..0 is 0001 (a setting for the calibrated internal RC oscillator, according to the data sheet). To determine the byte values we wish to have, we apply the bit "mask" developed above and overwrite relevant bits of each fuse byte.

```
Low original bits:   1110 0001
Low fuse mask:       XXXX 1100
Low new bits:        1110 1100

High original bits:  1101 1001
High fuse mask:      XXX1 XXXX
High new bits:       1101 1001
```

Once that is done, we can convert the new bytes back to hexadecimal for use with the hardware programmer.

Note that, as the CKOPT bit was already unprogrammed (1) in this example, the high fuse byte has not changed and doesn't need to be reprogrammed. The low fuse has changed, though, to a hex value of 0xECand must therefore be written to the microcontroller memory.

Lighting Fuses

Now that we have determined the new value for the low fuse byte, our work is done. Simply performing the inverse operation using the programmer will write the new byte to the chip:

```
$ avrdude -c stk500 -p m8 -P /dev/ttyS0 -U lfuse:w:0xEC:m
```

As you can see, programming the chip is easy. The majority of the work involves determining which options you want to set, how to set them and where the new bits actually go. And that means getting to know the relevant Atmel datasheet.

Final things To Remember

The fuses aren't erased when the AVR memory is erased, so reprogramming the fuses everytime the device is reprogrammed is **not** required. Since the fuses are not cleared by a memory erase, it can cause problems if incorrect settings are selected.

It may be possible to disabled In-System Programming (through the SPIEN fuse) while performing in-system programming. If you do this, you will no longer be able to program the chip in this manner (you need to use parallel programming to change this setting, which may mean removing the chip from the circuit and using another hardware programmer, such as the STK500).

It may also be possible to disable the RESET pin (RSTDISBL fuse). If this happens, the RESET pin must be pulled very high (12V) to program the chip and the circuit must tolerate this.

Some fuses just can't be changed through ISP Programming. If fuses cannot be changed through ISP Programming, Parallel Programming is required to alter the fuses.

The lock bits are fuses that can be used to lock down the chip, but this can lock you out as well!

You can use the `-v` switch to avrdude while reading the fuse bytes. The verbose output includes lots of neat info concerning the chip and current settings. Here is the output for an ATmega162, which shows that this chip has extended fuse (efuse) and lock byte, in addition to the low and high fuse bytes:

```
$ avrdude -v -c stk500 -p m162 -P /dev/ttyS1 -U hfuse:r:high.txt:r -U lfuse:r:low.txt:r
avrdude Version 4.4.0

  Using Port            : /dev/ttyS1
  Using Programmer      : stk500
  AVR Part              : ATMEGA162
  Chip Erase delay      : 9000 us
  PAGEL                 : P00
  BS2                   : P00
  RESET disposition     : dedicated
  RETRY pulse           : SCK
  serial program mode   : yes
  parallel program mode : yes
  Memory Detail         :

                              Page                       Polled
    Memory Type Paged  Size   Size #Pages MinW  MaxW   ReadBack
    ----------- ------ ------ ---- ------ ----- ----- ---------
    flash       yes     16384  128   128  4500  4500 0xff 0xff
    eeprom      no        512    0     0  9000  9000 0xff 0xff
    lfuse       no          1    0     0 16000 16000 0x00 0x00
    hfuse       no          1    0     0 16000 16000 0x00 0x00
    efuse       no          1    0     0 16000 16000 0x00 0x00
    lock        no          1    0     0 16000 16000 0x00 0x00
    signature   no          3    0     0     0     0 0x00 0x00
    calibration no          1    0     0     0     0 0x00 0x00

  Programmer Type : STK500
  Description     : Atmel STK500
  Hardware Version: 2
  Firmware Version: 1.14
  Vtarget         : 5.1 V
  Varef           : 5.1 V
  Oscillator      : 3.686 MHz
  SCK period      : 1.1 us
```

You are now ready to play with your AVR's fuses, just remember to double-check those values before performing the write!

Enjoy.

Level: Article

Additional Article Data

Level: Article

Jump to section

AVR | Go