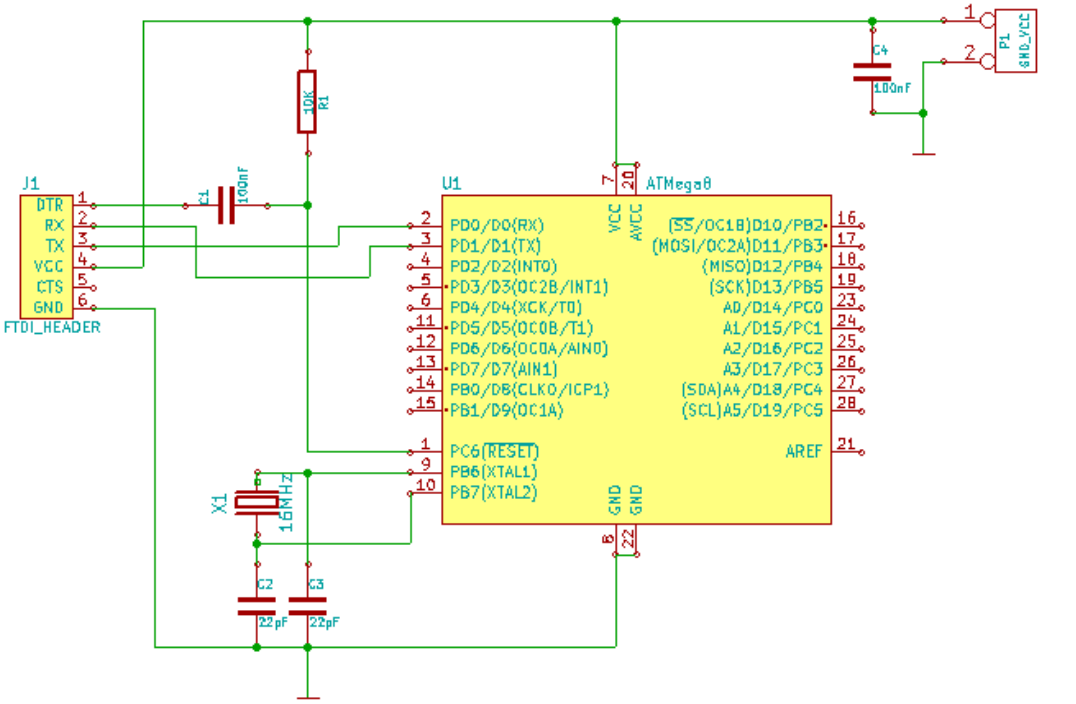
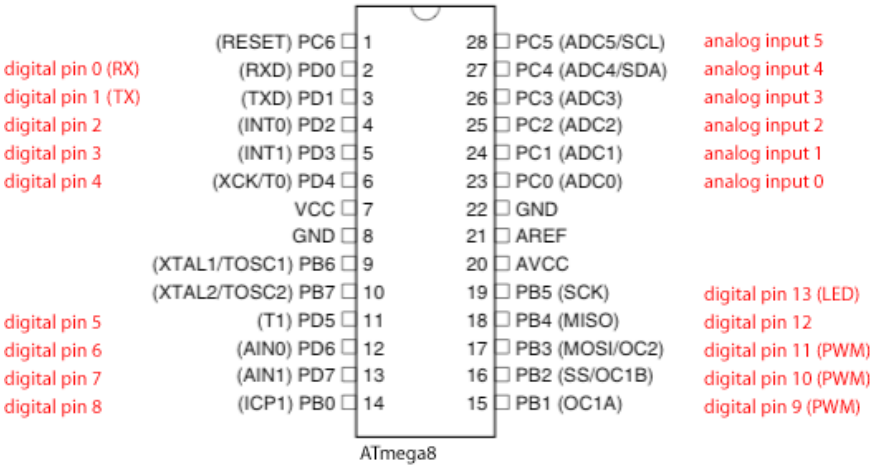
Arduinize a Mega8

As I began to get used to Arduino, and as I still had many Mega8 chips lying around, I had the idea to turn them into rudimentary Arduinos for some simple projects.

1. Schematic



Arduino Pin Mapping www.arduino.cc



Take care: the pin numbers used in the Arduino code are not the pin numbers of the Mega8, but the Arduino pins marked in red in the above pin mapping!

2. Software

In contrast to arduinizing a Mega32 or Mega16, no special library has to be installed for the Mega8. The old Arduino NG contained a Mega8, so Board: Arduino NG is OK.

(Note that a Mega168 or Mega328 is a better choice, as it has more memory. But I wanted to use my old Mega8 chips for some Arduino projects.)

The programming has to follow 2 steps:

- Burn a bootloader. This has to be done once.
  After this, the Mega8 is ready to be programmed via the serial interface (RxD, TxD).
- Upload a program with the Arduino IDE.

3. Burn bootloader

To do this, an ISP programmer must be connected to the pins SCK, MISO, MOSI and RESET\ .

Arduino uses avrdude for the programming with the following steps:

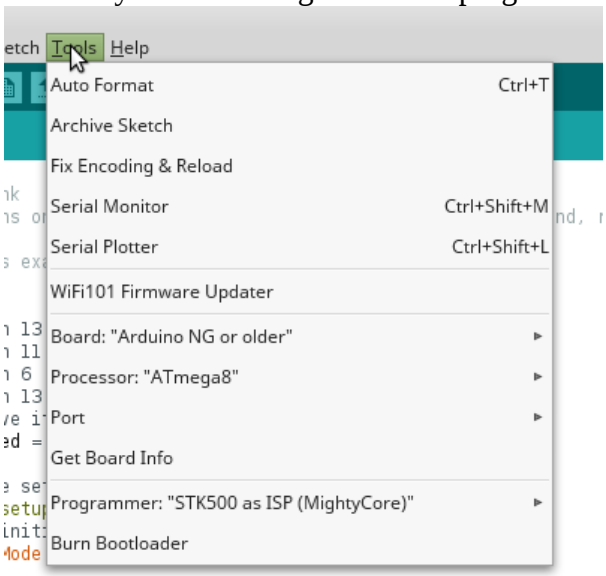
- Set the Unlock fuse bits to free the bootloader section of the Mega8
- Set the fuses High, Low and Extended
- Load bootloader
- Set the Lock fuse bits to protect the bootloader.

There are 2 possibilities:

- Use an ISP programmer or
- Use an Arduino Uno as programmer

Use an ISP programmer

I used mySmartUSBLight with the progrmmer setting STK500 in the Arduino menu.



Use an Arduino board as programmer

Instead of this an Arduino Uno board can be used as ISP programmer, with the following connections:

10 - RSET\
11 – MOSI
12 – MISO
13 – SCK

In addition to this, a 10µF capacitor is inserted between RESET and ground. This is to buffer the RESET pin so that it does not receive a LOW pulse from the DTR pin of the Arduino FTDI chip (Otherwise the Arduino would start its own bootloader instead of transferring the bootloader to the target).

The menu setting is: Programmer – Arduino as ISP
To burn the Bootloader using the Arduino as programmer, hold <Shift> and push <Upload>

4. Upload program

All Arduino boards have a builtin USB - Serial interface.
To program our Mega8, we need an external interface of this kind, eventually called an FTDI breakout bord, as usually a chip FTDI232 is used.

The interface must provide the signals RxD, TxD **and DTR**.
(Not all of the cheap boards provide DTR!)

RxD and TxD are uses for the communication, while DTR provides a short RESET pulse to tell the Mega8 that it should upload a program.

For test purposes a blink program from the Arduino examples is always a good choice.

5. Problems while uploading?

If there is a synchronize error, check the file boards.txt in <arduino folder> /hardware/arduino/avr:

```
atmegang.name=Arduino NG or older

atmegang.upload.tool=avrdude
atmegang.upload.protocol=arduino
atmegang.upload.speed=19200

atmegang.bootloader.tool=avrdude
atmegang.bootloader.unlock_bits=0x3F
atmegang.bootloader.lock_bits=0x0F

atmegang.build.mcu=atmegang
atmegang.build.f_cpu=16000000L
atmegang.build.board=AVR_NG
atmegang.build.core=arduino
atmegang.build.variant=standard
```

Check if the crystal frequency of the controller fits the defined CPU frequency, in our case 16MHz. Eventually this setting could be edited in the boards.txt file, if another crystal is used.

Fuses

For special purposes the fuses must be set via ISP programming.
This requires the use of avrdude as a command line tool.

Under Linux, and with the mySmartUSB programmer, the fuses could be read like this:

```
avrdude -c stk500v2 -P /dev/ttyUSB0 -p m8
```

Result:

```
avrdude: AVR device initialized and ready to accept instructions
Reading | ################################################## | 100% 0.01s
avrdude: Device signature = 0x1e9307
avrdude: safemode: Fuses OK (H:FF, E:D9, L:A4)
avrdude done.  Thank you.
```

The avrdude options have the following meaning:

```
-c    set programmer, here stk500v2
-P    virtual serial port via USB, here /dev/ttySUSB0
-p    set chip, here m8 = Mega8
-v    verbose output
```

The option -v is recommended to have a very detailed output.

A good tool to find the right fuse settings or to translate the fuse values to a readable form is the fuse calculator:

http://www.engbedded.com/fusecalc or
http://eleccelerator.com/fusecalc/

L fuses:

- **Internal oscillator or external crystal**
  (fuses CKSEL0 to CKSEL3 combined with SUT0, SUT1)
- **Brown out detection**
  (fuses BODLEVEL, BODEN)

H fuses:

- BOOTRST sets the starting address from 0 to the staring address of the **bootloader**
- BOOTSZ0, BOOTSZ1 set the **size of the bootloader** area in memory
- EESAVE decides if the EEPROM is deleted when programming.
- CKOPT: oscillator gain, should be set if crystal f > 8MHz
- WDTON: Watchdog timer on boot
- RSTDISBL: Reset disable switches off the Reset pin to have one more port pin.
  But this disables the possibility of SPI programming! Do not touch!

Lock bits:
These protect the controller memory from being read out. Wrong settings have as consequence that the controller can not be programmed anymore. Better not touch! Default is $Ffh.

Take care:

- A fuse is set when the corresponding bit is 0!
- a fuse setting EXTERNAL CRYSTAL requires a crystal and two 22pF capacitors
  A new Mega8 is set to INTERNAL OSC 1MHz

Do not touch:

- RSTDISBL that disables the reset pin
- SPIEN that enables the controller to be programmed via SPI
- Lock bits

Fuses link:
https://www.heise.de/make/artikel/Arduino-Uno-als-In-System-Programmer-2769246.html?seite=3

6. Links and further information

https://todbot.com/blog/2009/05/26/minimal-arduino-with-atmega8/

https://www.heise.de/make/artikel/Arduino-Uno-als-In-System-Programmer-2769246.html

- An Arduino can be used to program a PIC:
  https://www.heise.de/make/artikel/Arduino-Uno-als-In-System-Programmer-2769246.html?seite=4

7. Disadvantages and advantages of using a bootloader

Advantages:

- Sometimes there is some fumbling with ISP programmers before you get them to work, as there are so many different variants. (Under Linux there is rarely a problem, in contrast to Windows). The bootloader method however is very reliable, as there is always just one method of uploading, via Rx and Tx of the virtual serial port. The success of the Arduino boards is mainly due to this feature.

- Communication and programming is done via one USB interface.

- The program can be uploaded from far away, if a (vitual) serial connection is available

Disadvantage:
The Arduino bootloader waits for several (about 5) seconds for DTR to go low, indicating that a program should be loaded. This means that during this time the selfmade Arduino is not alive and does nothing but waiting.
This is not really fine for a controller that could be readyy immediately after switching it on.

8. What about dismissing the bootloader and directly programming the chip?

This is also possible, using a programmer like the mySmartUSBLight:

- Menu Tools – Programmer - <select programmer>
- Menu Sketch – Upload using Programmer

The advantage is that about 1K of flash RAM is saved.
The disadvantage is that, if you want serial communication, there are two devices to connect to the controller: the programmer and the FTDI serial adapter.

9. Appendix

Typical avrdude output while writing the bootloader:

```
avrdude -C <...>avrdude.conf -v -p atmega8 -c stk500 -P /dev/ttyUSB0
-Uflash:w:<...>arduino185/hardware/arduino/avr/bootloaders/atmega8/ATmegaBOOT-prod-firmware-2009-11-
07.hex:i -Ulock:w:0x0F:m

avrdude: Version 6.3, compiled on Jan 17 2017 at 11:00:16
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch
...
         Using Port                    : /dev/ttyUSB0
         Using Programmer              : stk500
Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of lock written
avrdude: verifying lock memory against 0x3F:
avrdude: load data lock data from input file 0x3F:
avrdude: input file 0x3F contains 1 bytes
avrdude: reading on-chip lock data:

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

avrdude: verifying ...
avrdude: WARNING: invalid value for unused bits in fuse "lock", should be set to 1 according to
datasheet
This behaviour is deprecated and will result in an error in future version
You probably want to use 0xff instead of 0x3f (double check with your datasheet first).
avrdude: 1 bytes of lock verified
avrdude: reading input file ""
avrdude: writing efuse (0 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 0 bytes of efuse written
avrdude: verifying efuse memory against :
avrdude: load data efuse data from input file :
avrdude: input file  contains 0 bytes
```

```
avrdude: reading on-chip efuse data:

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

avrdude: verifying ...
avrdude: 0 bytes of efuse verified
avrdude: reading input file "0xca"
avrdude: writing hfuse (1 bytes):

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

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

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

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified
avrdude: reading input file "0xdf"
avrdude: writing lfuse (1 bytes):

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

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

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

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

avrdude done.  Thank you.

         AVR Part                      : ATmega8
         Chip Erase delay              : 10000 us
         PAGEL                         : PD7
         BS2                           : PC2
         RESET disposition             : dedicated
         RETRY pulse                   : SCK
         serial program mode           : yes
         parallel program mode         : yes
         Timeout                       : 200
         StabDelay                     : 100
         CmdexeDelay                   : 25
         SyncLoops                     : 32
         ByteDelay                     : 0
         PollIndex                     : 3
         PollValue                     : 0x53
         Memory Detail                 :

                                  Block Poll               Page                       Polled
           Memory Type Mode Delay Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
           ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
           eeprom         4    20   128    0 no        512    4      0  9000  9000 0xff 0xff
           flash         33    10    64    0 yes      8192   64    128  4500  4500 0xff 0x00
           lfuse          0     0     0    0 no          1    0      0  2000  2000 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  2000  2000 0x00 0x00
           efuse          0     0     0    0 no          0    0      0     0     0 0x00 0x00
           lock           0     0     0    0 no          1    0      0  2000  2000 0x00 0x00
           calibration    0     0     0    0 no          4    0      0     0     0 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00

         Programmer Type : STK500V2
         Description     : Atmel STK500
         Programmer Model: STK500
         Hardware Version: 3
         Firmware Version Master : 2.10
         Topcard         : Unknown
         Vtarget         : 0.0 V
         SCK period      : 17.4 us
         Varef           : 0.0 V
         Oscillator      : Off
```

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

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9307 (probably m8)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "<...>arduino185/hardware/arduino/avr/bootloaders/atmega8/ATmegaBOOT-
prod-firmware-2009-11-07.hex"
avrdude: writing flash (8170 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 8170 bytes of flash written
avrdude: verifying flash memory against
<...>arduino185/hardware/arduino/avr/bootloaders/atmega8/ATmegaBOOT-prod-firmware-2009-11-07.hex:
...

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

avrdude: verifying ...
avrdude: 8170 bytes of flash verified
avrdude: reading input file "0x0F"
avrdude: writing lock (1 bytes):

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

avrdude: 1 bytes of lock written
avrdude: verifying lock memory against 0x0F:
avrdude: load data lock data from input file 0x0F:
avrdude: input file 0x0F contains 1 bytes
avrdude: reading on-chip lock data:

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

avrdude: verifying ...
avrdude: WARNING: invalid value for unused bits in fuse "lock", should be set to 1 according to
datasheet
This behaviour is deprecated and will result in an error in future version
You probably want to use 0xcf instead of 0x0f (double check with your datasheet first).
avrdude: 1 bytes of lock verified

avrdude done.  Thank you.
```

After programming the bootloader, the fuses read out with avrdude are set like this:

```
avrdude: safemode: lfuse reads as DF
avrdude: safemode: hfuse reads as CA
avrdude: safemode: Fuses OK (H:FF, E:CA, L:DF)
```

According to the fuse calculator, this means:

- External HF crystal with CKOPT set
- Brown out detection 2.7V
- Boot reset vector enabled, start address $0E00h, 512 words = 1kByte
- SPI enabled