avrprog2 - A Linux frontend for the mikroElektronika AVRprog2 Programming Hardware

Andreas Hagmann, ahagmann@ecs.tuwien.ac.at

August 9, 2011

This documentation is mainly for programmers who want to extend the functionality of *avrprog2* [1] or want to reuse code in their own projects.

Contents

[1] http://sourceforge.net/projects/avrprog2

avrprog2 is a linux frontend for the mikroElektronika (http://mikroe.com) AVRprog2 programming hardware.

1 Compilation and Installation

This project uses the *GNU autoconf* toolchain.
If you want to use the source directly from the repository start with (in this the newest trunk version is used):

```
svn export https://avrprog2.svn.sourceforge.net/svnroot/avrprog2 avrprog2
cd avrprog2/trunk
autoreconf --install
```

To build the application from a downloaded source package start with:

```
tar xfvz avrprog2-1.0.0.tar.gz
cd avrprog2-1.0.0
```

The following steps are equal in both cases and will compile the application in the build directory.

```
mkdir build
cd build
../configure
make
```

To install the application type

```
sudo make install
```

To build the doxygen reference execute

```
make doc
```

2 Program flow

The program does all its tasks in the following order.

- Parse the command line arguments and check the correctness of them.
- Do all things which do not need a connection to the hardware device.
  - Print version information.
  - List all supported devices.
  - Read input files.
- Connect to the programming hardware.
- If not specified on the command line try to auto detect the following parameters.
  - Auto detect programming pins.
  - Auto detect device by the device signature.
  - Auto detect device frequency.
- Perform the actions according to the command line parameters in the following order.
  - Perform a chip erase.
  - Perform Fuses actions (write, read, verify).
  - Perform Flash memory actions (write, read, verify).
  - Perform EEPROM memory actions (write, read, verify).
- Close the connection to the programming hardware.

3 Auto detection Mechanisms

This section describes the mode of operation of the implemented autodetection mechanisms.

3.1 Programming Pins

Some mcu cards for the mikroElektronika development boards have more than one mcu socket. Hence it is necessary to specify the right one before one tries to communicate with the device.
The procedure to detect the right pins is quite simple. It selects the first socket, tries to detect a device and stops if a device answers. If not, the same procedure is done with the second socket.

3.2 Device

If no device type is specified on the command line the programmer tries to identify the device itself. Todo so it reads the device signature and looks in the config directories for the according configuration file. To avoid unintended damage of the target device, this feature is deactivated when fuse bytes are written.

3.3 Device frequency

If no frequency is specified at the command line the programmer tries to detect the device frequency in the following way.

- It sets the frequency to the lowest possible value (128kHz).
- Then it reads the device signature.
- Now the frequency is increased up to 16Mhz and after each step the device signature is read again. If it does not coincide with the first read signature, this frequency was too high and the programmer returns to the last working one.

4 Device Configuration Files

Default locations for device description files are /etc/avrprog2 (system wide) and ∼/.avrprog2 (per user).
Device description files contain the following information about a AVR device.

- name
- device signature
- size of Flash memory
- size of EEPROM memory
- number of Fuse bytes
- package (used to select the programming pins, this is optional)

This information is stored in ∗.xml files, which look like this one:

```
<device>
    <name>ATmega 128</name>
    <signature>0x1e9702</signature>
    <flashSize>4096</flashSize>
    <eepromSize>4096</eepromSize>
    <numOfFuses>3</numOfFuses>
    <package>TQFP64</package>
</device>
```

If the *package* is not specified, avrprog2 tries to auto detect a device and to select the right programming pins.

5 Binary Files

Avrprog2 can read files in *elf* and *ihex* format. It determines the type by the file extension. Where ∗.elf files are opened as elf, whereas ∗.hex, ∗.ihex and ∗.eep files are treated as intel hex files.
In the following the behavior of the file parsers is described. It depends mainly on the type of memory (EEPROM, Flash or Fuses) to which the file should be written.
When reading content from memory to a file, this application writes always files in ihex format.

5.1 ihex Parser

When a hex file is read, all sections of the file are copied to the programming buffer. If the target memory is flash memory, the *lma* entries in each section are considered. For other memory types the entry is ignored and all sections are written sequentially.

5.2 elf Parser

When a elf file is read it depends on the target memory which sections are copied to the internal buffer. If the target is Flash memory, then the .text and .data sections are copied with consideration of the *lma* entries.
If the target is EEPROM memory, the .eeprom section is copied to the internal buffer without processing the *lma* entry. Hence the content will start at EEPROM address 0x0000. This allows programming of elf files, without the need to invoke *objcopy*.
When the target of the programming operation are fuse bytes, then the .fuse section of the elf file is read. Again the *lma* entry is ignored.

6 Libraries

The following libraries are used by this programmer.

- *libbfd* is used for reading and writing binary files.
- *libboostfilesystem* is used to traverse the config directories.
- *libusb* is used for usb communication. Here a version greater 1 is necessary since in older versions the isochronous transfer mode is no implemented.
- *libboostpropertytree* is used for reading the xml configuration files.

7 Further Documentation

More implementation details are documented directly in the source code. A class reference can be build with Doxygen (Todo so use the *doc* make target.) The used low level commands for the communication with the programming hardware are documented in CAvrProgCommands.cpp.