Code Patch

Setting the UNIX/Linux Environment to Develop an AVR Application

By: Daniel Widyanto, ITB DSP Lab

Features

- Setting GNU Toolchain
- Setting Debugger and Simulator

Introduction

This application note will guide the UNIX/Linux user in building an AVR development environment.

Setting GNU Toolchain

The GNU toolchain consists of GCC (GNU Collection Compiler), binutils, and library. These packages are usually installed in your Linux box, but usually it is configured only to build host architecture software.

To make it compile AVR binaries, one must download and reconfigure these toolchains.

Package Name	Description	Download Location
Binutils-2.14 `binutils-2.14.tar.bz2`	Binary utilities	http://freshmeat.net/projects/binutils/
GCC-3.3.1 `gcc-3.3.1.tar.bz2` **OR** `gcc-core-3.3.1.tar.bz2` `gcc-g++-3.3.1.tar.bz2`	C compiler	http://gcc.gnu.org/mirrors.html
AVR-libc-1.0.4 `avr-libc-1.0.4.tar.bz2`	AVR C libraries	http://savannah.nongnu.org/download/avrlibc/
AVR libc - User Manual `avr-libc-user-manual-1.0.4.tar.bz2`	AVR C libraries user manual	http://savannah.nongnu.org/download/avrlibc/

Table 1. GNU Toolchain

The GNU Toolchain is a combination of GCC, binutils, and avr-libc versions. Not all of these combinations work. Please take a look at http://winavr.sourceforge.net/package.html for a working combination.

To start the installation, make a separate directory first. A separate directory makes it easy to remove.

```
$ su
$ mkdir /usr/local/atmel
```

Binutils-2.14

Extract binutils, configure, and install using UNIX/Linux default installation. Option --prefix is used to arrange its output directory.

```
$ cd /temprorary/working/directory
$ tar -xvjf binutils-2.14.tar.bz2
$ cd binutils-2.14
$ ./configure --target=avr --prefix=/usr/
    local/atmel \
   --disable-nls
$ make
$ make install (as root)
```

GCC-3.3.1

Only GCC above v.2.95 is available for AVR, except GCC-3.3.3, which is not work for AVR. GCC-3.3.1 is the only stable version of GCC when this application note was made. C++ is still in development in this version.

```
$ cd /temprorary/working/directory
$ tar -xvjf gcc-core-3.3.1.tar.bz2
$ tar -xvjf gcc-g++-3.3.1.tar.bz2
```

OR

```
$ tar -xvjf gcc-3.3.1.tar.bz2
$ mkdir gcc-build
$ cd gcc-build
$ ../gcc-3.3.1/configure --target=avr \
  --prefix=/usr/local/atmel \
  --enable-language=c,c++ --disable-nls
$ make
$ make install (as root)
```

AVR-libc-1.0.4

AVR libc has to be compiled within a new directory outside its source code directory. Otherwise, some of the AVR type library won't be compiled.

```
$ cd /temprorary/working/directory
$ tar -xvjf avr-libc-1.0.4.tar.bz2
$ mkdir/temprorary/working/directory/avr-libc
$ cd avr-libc
$ export PATH=/usr/local/atmel/bin:$PATH
$ ../avr-libc-1.0.4/configure --
    prefix=/usr/local/atmel
$ make
$ PATH=/usr/local/atmel/bin:$PATH
    make install (as root)
```

AVR-libc-1.0.4 User Manual

AVR-libc usually come with built-in documentation. You need to install Doxygen to build these documents. This

package contains only AVR-libc documentation in HTML format, just in case you don't have Doxygen to build the AVR-libc documents. Extract it to any place that you like.

```
$ tar -xvjf avr-libc-user-manual-1.0.4.tar.bz2
    -C /usr/local/atmel (as root)
```

Setting Environment

After these toolchains are installed, you need to add the path in your environment shell. Assume you're using BASH, add this in your `.bash_profile` file

```
PATH=$PATH:/usr/local/avr/bin
export PATH
```

Setting Debugger and Simulator

Debugger and simulator is useful to verify one's design. There are many free AVR debuggers and simulators around. This application note uses gdb and SimulAVR.

Package Name	Description	Download Location
GDB-6.1 gdb-6.1.tar.gz	GNU Debugger	http://ftp.gnu.org/gnu/gdb ftp://sources.redhat.com/pub/gdb/releases/
SimulAVR-0.1.2.1 simulavr-0.1.2.1.tar.bz2	AVR Simulator	http://savannah.nongnu.org/download/simulavr/

GDB-6.1 GDB

(GNU Debugger) is GNU's official debugger. GDB must be combined with SimulAVR to simulate AVR.

```
$ cd /temprorary/working/directory
$ tar -xvzf gdb-6.1.tar.gz
$ cd gdb-6.1
$ ./configure --target=avr --
    prefix=/usr/local/atmel
$ make
$ make install (as root)
```

SimulAVR-0.1.2.1

SimulAVR provides an AVR emulator in your computer. The AVR registers and data as provided by SimulAVR. You can read these registers and data using GDB.

```
$ cd /working/directory
$ tar -xvjf simulavr-0.1.2.1.tar.bz2
$ cd simulavr-0.1.2.1
$ ./configure --prefix=/usr/local/atmel
$ make
$ PATH=/usr/local/atmel/bin:$PATH
    make install (as root)
```

Code Patch

Using STK500 Under UNIX/Linux Environment

By: Daniel Widyanto, ITB DSP Lab

Introduction

This application note is a guide to use the STK500 features under a UNIX/Linux environment. To be able to use STK500 in Linux, one needs additional software. There are two software packages that can be used with STK500: AVRdude (formerly known as AVRprog) and UISP. It is recommended that you use UISP, since it can handle all of STK500 features.

AVRdude-4.3.0

Installation

AVRdude can be downloaded from http://savannah.nongnu.org/download/avrdude. To install it, extract the source first.

```
$ tar -xvjf avrdude-4.3.0.tar.bz2
```

Then, configure, compile, and install.

```
$ cd avrdude-4.3.0
$ ./configure
$ make
$ su
$ make install
```

Programming Devices on STK500

The typical command to use STK500 with avrdude is :

```
$ avrdude -c stk500 -p <device> \
  -U <memtype>:r|w|v:<filename>[:format]
     [other_option]
```

Option explanation :

-c : Use to identify the programmer that is used. Currently, AVRdude supports STK200, STK500, avrisp, AVR910, butterfly.

-p : AVR device. Currently these AVR <device> are supported :

t15	AT Tiny15	1200	AT90S1200
2313	AT90S2313	2323	AT90S2323
2333	AT90S2333	2343	AT90S2343
4414	AT90S4414	4433	AT90S4433
4434	AT90S4434	8515	AT90S8515
8535	AT90S8535	m163	ATMEGA163
m169	ATMEGA169	m128	ATMEGA128
m103	ATMEGA103	m16	ATMEGA16
m8	ATMEGA8		

-U : Memory operation. Multiple -U can be used to operate on multiple memories at single command.

<memtype> : memory type (flash or eeprom).

r | w | v : (r)ead memory to <filename>, (w)rite <filename> to memory, (v)erify memory against <filename>

[:format] : <filename> format. The option is:

- i - Intel-hex format
- s - Motorola S-record
- r - Raw binary, little endian byte order
- a - Auto
- m - Immediate, the value is entered directly

Reading Flash/EEPROM

To read flash from a device on STK500, use the following command (assume you are using AT90S8515):

```
$ avrdude -c stk500 -p 8515 \
    -U flash:r:8515_flash.hex:i
```

The AT90S8515 flash memory will be saved in 8515_flash.hex. This file is in Intel-hex format. To read EEPROM, change flash to eeprom.

Writing Flash/EEPROM

To write flash to the device on STK500, use the following command (assume you are using AT90S8515):

```
$ avrdude -c stk500 -p 8515 \
    -U flash:w:test.hex:i
```

AVRdude will automatically erase the device before it is programmed and verify the result. The test.hex file is in Intel-hex format. To write the EEPROM, change flash to eeprom.

Signature, Fuses, and LockBits

You must read the datasheet : fuse programming, lockbit programming, and signature reading section to understand the magic number that is used to program the signature, fuses, and lockbits.

NOTE: Not all AVR devices support fuses/lockbits read/write in ISP programming mode.

Signature, fuses, and lockbits are located in special memory section (depend on AVR type). To see, where they are located, type (assume you are using ATMega16):

```
$ avrdude -c stk500 -p m16 -v
```

You will see something like this :

(see table top of next page)

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

It means that the signature is located in signature memory, fuse in lfuse and hfuse memory, and lock in lock memory. It maybe different on different AVR devices. Usually, this name corresponds to the datasheet (eg. lfuse to describe *low-byte fuse* in ATMega16).

Programming Fuses/LockBits
To program lockbits/fuses, use lock/lfuse/hfuse as memory type, replacing flash/eeprom keyword in -U option. For example, to activate mode 2 BLB1 (Boot Lock Bit) in ATMega16, type (refer to AT Mega16 datasheet page 259)

```
$ avrdude -c stk500 -p m16 -U lock:w:0x2F:m
```

Notes: with m-modes, all values are located in filename section and are entered directly in the lockbit memory.

UISP
UISP is far more advanced than AVRdude. UISP can be used to set advanced features in STK500 (setting STK500 oscillator, VTARGET, AREF). It also can be used to program STK500 in high-voltage mode.

Installation
UISP can be downloaded from http://savannah.nongnu.org/download/uisp/. UISP only needs typical installation (extract, configure, compile, and install).

```
$ tar -xvjf uisp-20040311.tar.bz2
$ cd uisp-20040311
$ ./configure
$ make
$ su
$ make install
```

Programming STK500
Typical command to program STK500 is:

```
$ uisp -dserial=/dev/ttyS0 -dprog=stk500 \
  -dpart=ATxxx --[upload|download|verify] \
  --segment=flash|eeprom|fuse \
  if=<input_hex_file>
      [of=<output_hex_file>]
```

Option explanation:

-dserial :location where your STK500 is attached. / dev/ttyS0 for serial port 1, serial port 2 is / dev/ttyS1, and so on
-dprog :selecting STK500 board
-dpart :device name. Use complete device name (eg, -dpart=atmega16)

Reading Flash/EEPROM
To read flash, use (assume you are using AT90S8515):

```
$ uisp -dserial=/dev/ttyS0 -dprog=stk500 \
    -dpart=AT90S8515 --download --segment=
      flash \ of=8515_flash.hex
```

The flash will be saved in 8515_flash.hex. This is Motorola S-record format file. To read eeprom, replace --segment=flash with -- segment=eeprom

Writing Flash/EEPROM
To write flash, use (assume you are using AT90S8515):

```
$ uisp -dserial=/dev/ttyS0 -dprog=stk500 \
  -dpart=AT90S8515 --upload --segment=flash \
  if=test.hex
```

UISP can accept Intel 16-bit hex file or Motorola S-records (S1 and S2). The file type is automatically detected by UISP. It also erase the chip automatically.

Programming Fuses/LockBits
Under UISP, fuses and lockbits programming is far more easy than AVRdude.

NOTE: Not all AVR device support fuses/lockbits read/write in ISP programming mode.

There are several additional option to program fuses and lockbits:

--rd_fuses : Read all fuses and print values to screen / stdout
--wr_fuse_l=byte : Write fuse low byte
--wr_fuse_h=byte : Write fuse high byte
--wr_fuse_e=byte : Write fuse extended byte
--wr_lock=byte : Write lock bits. Argument is a byte where each bit is:
Bit5 -> BLB12
Bit4 -> BLB11
Bit3 -> BLB02
Bit2 -> BLB01
Bit1 -> LB2
Bit0 -> LB1

Here's an example of how to program mode 2 BLB1 (Boot Lock Bit) in ATMega16 (refer to AT Mega16 datasheet page 259):

```
$ uisp -dserial=/dev/ttyS0 -dprog=stk500 \
    -dpart=atmega16 --wr_lock=0x2F
```

STK500 Additional Features
STK500 has many additional features, such as high-voltage programming, on-board oscillator, VTARGET, AREF software controllable.

High Voltage Programming
Refer to page 3-13 of the STK500 User Manual for high-voltage programming setup.

Add option -dhiv to uisp, eg:

```
$ uisp -dserial=/dev/ttyS0 -dprog=stk500 \
    -dpart=AT90S8515 -dhiv --upload \
    --segment=flash if=test.hex
```

AREF setting
Refer to page 3-17 of the STK500 User Manual for AREF jumper setting.

Reading AREF
Use option: --rd_aref

NOTE: Due to a bug in the STK500 firmware, the read value is sometimes off by 0.1 from the actual value measured with a volt meter.

Writing AREF
Use option: --wr_aref=<value>

Valid values are 0.0 to 6.0 volts in 0.1 volt increments. Value can not be larger than the VTARGET value.

VTARGET setting
Refer to page 3-16 of the STK500 User Manual for AREF jumper setting.

Reading VTARGET
Use option: --rd_vtg

NOTE: Due to a bug in the STK500 firmware, the read value is sometimes off by 0.1 from the actual value measured with a volt meter.

Writing VTARGET
Use option: --wr_vtg=<value>

Valid values are 0.0 to 6.0 volts in 0.1 volt increments. Value can not be smaller than the AREF value.

Oscillator setting
Refer to page 3-20 STK500 User Manual for AREF jumper setting.

Reading oscillator
Use option: --rd_osc
The frequency is displayed in Hertz

Writing oscillator
Use option: --wr_osc=<value>
Set the oscillator frequency in Hertz, from 14.06 to 3686400.