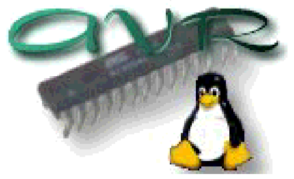
Programming the AVR Microcontroller with GCC



by Guido Socher (homepage)

About the author:

Guido loves Linux not only because it is fun to discover the great possibilities of this systems but also because of the people involved in its design.

Abstract:

Note: An updated version of this article is now available at: ../November2004/article352

The AVR 8-Bit RISC Microcontroller from Atmel is a very common Microcontroller It’s a single integrated circuit with EEPROM, Ram, Analog to Digital converter, a lot of digital input and output lines, timers, UART for RS 232 communication and many other things.

The best is however that a complete programming environment is available under Linux: You can program this Microcontroller in C using GCC. In this article I will explain how to install and use GCC. I will as well explain how to load the software into the Microcontroller. All you need for this are an AT90S4433 Microcontroller, a 4Mhz crystal, some cable and a few other very cheap parts.

This article shall be only an introduction. In a later article we will build a LCD display with a few push buttons, analog and digital inputs, hardware watchdog and LEDs. The idea is that this will be a general purpose control panel for a Linux Server but first we will learn how to setup the programming environment and that is what this article is about.

Software installation: What you need

To use the GNU C development environment you need the following software:

binutils-2.11.2.tar.bz2	Available from: ftp://ftp.informatik.rwth-aachen.de/pub/gnu/binutils/ or ftp://gatekeeper.dec.com/pub/GNU/binutils/
gcc-core-3.0.3.tar.gz	Available from: ftp://ftp.informatik.rwth-aachen.de/pub/gnu/gcc/ or ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr-libc-20020106 .tar.gz	The AVR C-library is available from: http://www.amelek.gda.pl/avr/libc/ You can as well download it from this server: download page
uisp-20011025.tar.gz	The AVR programmer is available from: http://www.amelek.gda.pl/avr/libc/ You can as well download it from this server: download page

We will install all the programs to /usr/local/atmel. This is to keep the program separate from your normal Linux C compiler. Create this directory with the command:

```
mkdir /usr/local/atmel
```

Software installation: GNU binutils

The binutils package provides all the low-level utilities needed for building object files. It includes an AVR assembler (avr-as), linker (avr-ld), library handling tools (avr-ranlib, avr-ar), programs to generate object files loadable to the Microcontroller’s EEPROM (avr-objcopy), disassembler (avr-objdump) and utilities such as avr-strip and avr-size.

Run the following commands to build and install the binutils :

```
bunzip2 -c binutils-2.11.2.tar.bz2 | tar xvf -
cd binutils-2.11.2
./configure --target=avr --prefix=/usr/local/atmel
make
make install
```

Add the line /usr/local/atmel/lib to the file /etc/ld.so.conf and run the command /sbin/ldconfig to rebuild the linker cache.

Software installation: AVR gcc

avr-gcc will be our C compiler.

Run the following command to build and install it:

```
tar zxvf gcc-core-3.0.3.tar.gz
cd gcc-core-3.0.3
./configure --target=avr --prefix=/usr/local/atmel --disable-nls --enable-language=c
make
```

```
make install
```

Software installation: The AVR C-library

The C-library is still under development. The installation might still change a bit from release to release. I recommend to use the version as shown in the above table if you want to follow the instructions step by step. I have tested this version and it works fine for all the programs that we will write in this and the following articles.

Set some environment variables (syntax is for bash):

```
export CC=avr-gcc
export AS=avr-as
export AR=avr-ar
export RANLIB=avr-ranlib
export PATH=/usr/local/atmel/bin:${PATH}

./configure --prefix=/usr/local/atmel --target=avr --enable-languages=c --host=avr
make
make install
```

Software installation: The Programmer

The programmer software loads the specially prepared object code into the EEPROM of our Microcontroller.

The uisp programmer for Linux is a very good programmer. It can be used directly from within a Makefile. You just add a "make load" rule and you can compile and load the software in one go.

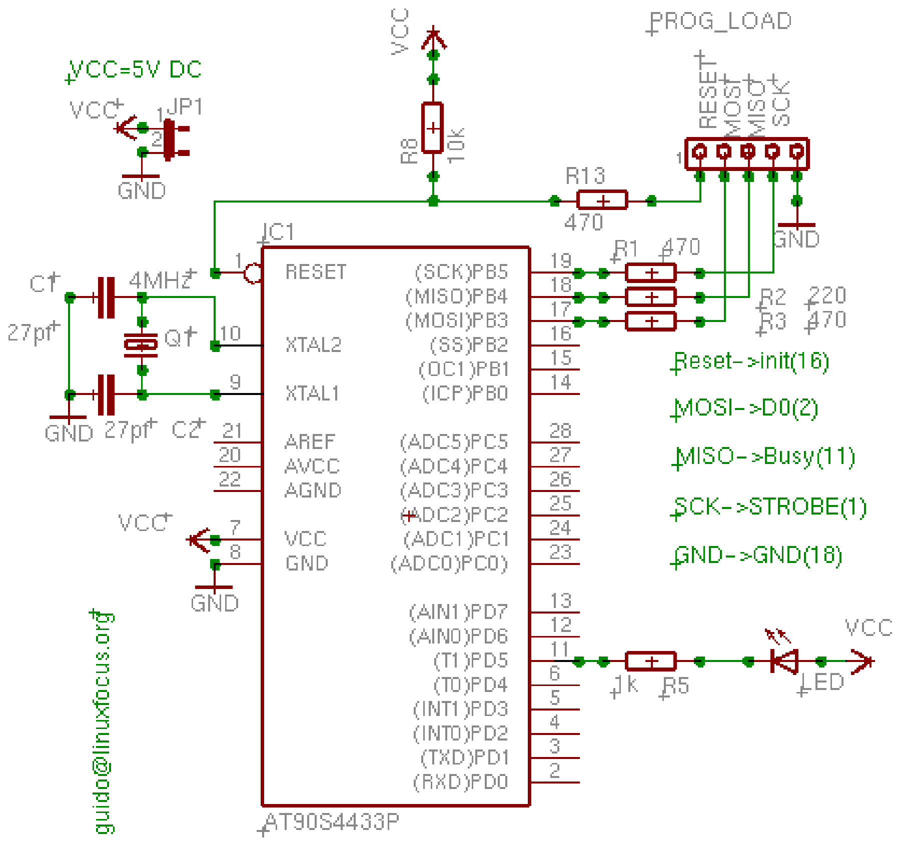
uisp is installed as follows:

```
tar zxvf uisp-20011025.tar.gz
cd uisp-20011025/src
make
cp uisp /usr/local/atmel/bin
```

A small test project

We will start with a small test circuit. The purpose of this circuit is just to test our development environment. We can use it to compile, download and test a simple program. The program will just cause a LED to blink.

I suggest to make a small printed circuit board for the Microcontroller. You can later on extent this circuit to do your own experiments. A good idea is to use a breadboard for this. You should however not try to put the AVR with it’s 4Mhz crystal directly onto the breadboard. It is better to use a few short wires to connect input and output lines with the breadboard since such breadboards are not made for fast digital circuits. The 4Mhz crystal and the capacitors should be physically very close to the Microcontroller.

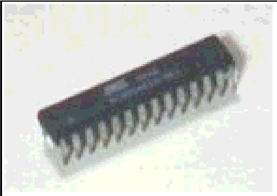
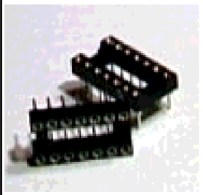
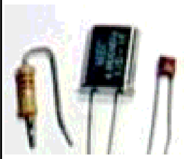
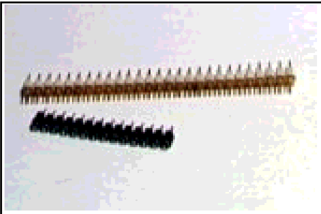
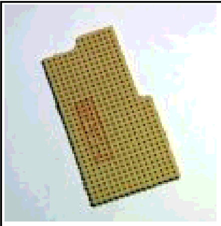


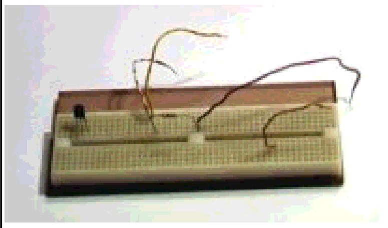
The resistors on the connector for the programmer are actually not needed in our case. You need them only if you plan to use the port-B input/output lines for other purposes.

Udo Puetz has provided a possibly more newbie friendly schematic which you find here: avr_layout_newbiefriendly.gif.

Needed Hardware

You need the parts listed in the table below. All of them are very common and cheap. Only the Microcontroller is a bit more expensive, about 7.50 Euro. Although it is a very common Microcontroller it might not be available in every local radio shop but bigger distributors for electronic components like (www.reichelt.de (germany), www.conrad.de (germany), www.selectronic.fr (france), etc..., probably there are similar sites in your country) have them all in stock.

	1 x AT90S4433, Atmel 8 bit Avr risc processor.
	2 x 14 pin IC socket or 1 x 28 pin 7.5mm IC socket The 28 pin socket is a bit more difficult to get. Usually the 28 sockets are 14mm wide but we need a 7.5mm socket.
	1 x 10K resistor (color code: brown,black,orange) 3 x 470 Ohm resistor (color code: yellow,purple,brown) 1 x 1K resistor (color code: brown,black,red) 1 x 220 Ohm resistor (color code: red,red,brown) 1 x 4Mhz Crystal 2 x 27pf ceramic capacitor
	Any kind of 5 pin connector/socket for the programmer. I usually buy these strips of connectors and break off 5 of them.
	matrix board
	1 x DB25 connector to plug into the parallel port.
	1 x LED

	A breadboard. We don't use it here but it is very useful if you want to do further experiments with the AVR. I suggest you leave the Microcontroller together with the crystal and the capacitors on the matrix board and connect the input/output lines via short cables to the breadboard.

In addition to the above parts you need a 5V electronically stabilized DC power supply or you can use a 4.5V battery as power supply.

Building the programmer hardware

The AT90S4433 allows for in circuit programming (ISP).
That is: you can do not need to remove the Microcontroller form the board to program it. You will see that you can buy ready made programmer hardware for 50-150 Euro. You do not need to invest that much in a programmer. With Linux, the uisp software and a free parallel port you can build a very good and simple AVR programmer. It's a simple cable. The wiring for the programmer cable must be as follows:

pin on AVR	Pin on parallel port
Reset (1)	Init (16)
MOSI (17)	D0 (2)
MISO (18)	Busy (11)
SCK (19)	Strobe (1)
GND	GND (18)

The cable should not be longer than 70cm.

Writing software

The AT90S4433 can be programmed in plain C with the help of gcc. To know some AVR assembler can be useful but it is not needed. The AVR libc comes with an avr-libc-reference which documents most of the functions. Harald Leitner has written a document with a lot of useful examples on how to use the AVR and GCC (haraleit.pdf, 286Kb, originally from http://www.avrfreaks.net/AVRGCC/). From Atmel's website, (www.atmel.com, go to: avr products -> 8 bit risc-> Datasheets), you can download the complete data sheet (local copy: avr4433.pdf, 2361Kb) . It describes all the registers and how to use the CPU.

One thing to keep in mind when using the 4433 is that it has only 128Byts of Ram and 4K EEPROM. That means you must not declare large data structures or strings. Your program should not use deeply nested function calls or recursion. Writing a line like
char string[90];
will already be too much. An integer is 16 bit. If you need a small integer then use

```
unsigned char i; /* 0-255 */
```
You will however be surprised how big programs you can write. It’s a really powerful processor!

Much better than all theory is a real example. We will write a program that causes our LED to blink in 0.5 seconds intervals. Not very useful but good to get started and to test the development environment and the programmer.

```
void main(void)
{
    /* enable PD5 as output */
    sbi(DDRD,PD5);
    while (1) {
        /* led on, pin=0 */
        cbi(PORTD,PD5);
        delay_ms(500);
        /* set output to 5V, LED off */
        sbi(PORTD,PD5);
        delay_ms(500);
    }
}
```

The above code snipt shows how simple it is to write a program. You see only the main program the delay_ms function is included in the full listing (avrledtest.c). To use pin PD5 as output you need to set the PD5 bit in the data direction register for port D (DDRD). After that you can set PD5 to 0V with the function cbi(PORTD,PD5) (clear bit PD5) or to 5V with sbi(PORTD,PD5) (set bit PD5). The value of "PD5" is defined in io4433.h which is included via io.h. You don’t have to worry about it. If you have already written programs for multi user / multi tasking systems such as Linux you know that one must never program a non blocking endless loop. This would be a waste of CPU time and slow the system very much down. In the case of the AVR this is different. We don’t have several tasks and there is no other program running. There is not even an operating system. It is therefore quite normal to busy loop forever.

Compiling and loading

Before you start make sure that you have /usr/local/atmel/bin in the PATH. If needed edit your .bash_profile or .tcshrc and add:

```
export PATH=/usr/local/atmel/bin:${PATH} (for bash)
setenv PATH /usr/local/atmel/bin:${PATH} (for tcsh)
```

We use the parallel port and uisp to program the AVR. Uisp uses the ppdev interface of the kernel. Therefore you need to have the following kernel modules loaded:

```
# /sbin/lsmod
parport_pc
ppdev
parport
```

Check with the command /sbin/lsmod that they are loaded other wise load them (as root) with

```
modprobe parport
modprobe parport_pc
modprobe ppdev
```

It is a good idea to execute these commands automatically during startup. You can add them to a rc script (e.g for Redhat /etc/rc.d/rc.local).
To use the ppdev interface as normal user root needs to give you write access by running once the command

```
chmod 666 /dev/parport0
```

Make as well sure that no printer daemon is running on the parallel port. If you have one running then stop it before you connect the programmer cable. Now everything is ready to compile and program our Microcontroller.

The package for our test program (avrledtest-0.1.tar.gz) includes a make file. All you need to do is type:

```
make
make load
```

This will compile and load the software. I will not go into the details of all the commands. You can see them in the Makefile and they are always the same. I can my self not remember all of them. I just know that I need to use "make load". If you want to write a different program then just replace all occurrences of avrledtest in the Makefile with the name of your program.

Some interesting binutils

More interesting than the actual compilation process are some of the binutils.

```
avr-objdump -h avrledtest.out
```

Shows the size of the different sections in our program. .text is the instruction code and loads into the flash EEPROM. .data is initialized data such as
static char str[]="hello";
and .bss is uninitialized global data. Both are zero in our case. The .eeprom is for variables stored in eeprom. I never had any use for this. stab and stabstr is debugging info and will not make it into the AVR.

```
avrledtest.out:     file format elf32-avr

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000008c  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00800060  0000008c  00000120  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00800060  0000008c  00000120  2**0
                  ALLOC
  3 .eeprom       00000000  00810000  00810000  00000120  2**0
                  CONTENTS
  4 .stab          00000750  00000000  00000000  00000120  2**2
                   CONTENTS, READONLY, DEBUGGING
  5 .stabstr       000005f4  00000000  00000000  00000870  2**0
                   CONTENTS, READONLY, DEBUGGING
```

You can as well use the command avr-size to get this in a more compressed form:

avr-size avrledtest.out

```
   text    data     bss     dec     hex filename
    140       0       0     140      8c avrledtest.out
```

When working with the AVR you need to watch out that text+data+bss is not more than 4k and data+bss+stack (you can not see the size of the stack, it depends on how many nested function calls you have) must not be more than 128 Bytes.

As well interesting is the command

avr-objdump -S avrledtest.out

It will generate an assembler listing of your code.

Conclusion

Now you know enough to start your own projects with the AVR hardware and GCC. There will as well be further articles in LinuxFocus with more complex and more interesting hardware.

References

- Libc and uisp: /www.amelek.gda.pl/avr/libc/
- GCC and binutils: ftp://gatekeeper.dec.com/pub/GNU/
- avrfreaks (watch out some people on that site are still using windows !?): http://www.avrfreaks.net/
- the tavrasm assembler for Linux: www.tavrasm.org
- AVR webring: R.webring.com/hub?ring=avr&list
- Pre-compiled versions of gcc: combio.de/avr/
- All software and documents mentioned in this article
- The atmel website: www.atmel.com/