

All software versions and URLs in this document valid as of dec.13, 2001.

Installing and using AVRGCC with AVRstudio

This is a comprehensive introduction on how to install AVRstudio and the GNU AVR C compiler “*avr-gcc*”, and making them work together. This introduction leads step by step to the successful build of sample code, and programming of whichever AVR part you have chosen, on the STK500 development board.

Used in this introduction:

- AVRstudio executable installer, “astudio.exe” release 3.53 of nov.8, 2001[5.9M] or later. Downloadable from www.avrfreaks.net
- AVRGCC executable installer “avrgcc200112XXa_AVRfreaks.exe” [8.1M], AVRfreaks distribution of dec.07, 2001 or later.

This package also contains:

- Flavio Gobber’s Elf2Coff converter.
- The gcctest 1-9 files, by Volker Oth.

Downloadable from www.avrfreaks.net. It is important that you try to use our package, as it contains all the correct additional files (.bat files, makefiles) that you need to complete this guide).

Besides this software you of course need the STK-500 (or equivalent) development board, a chip (i.e. AT90s8515), and a Personal Computer of a certain standard running MS Windows9x/NT/2000. It is assumed that you have set up your STK500 to work with your computer’s COM port before starting out.

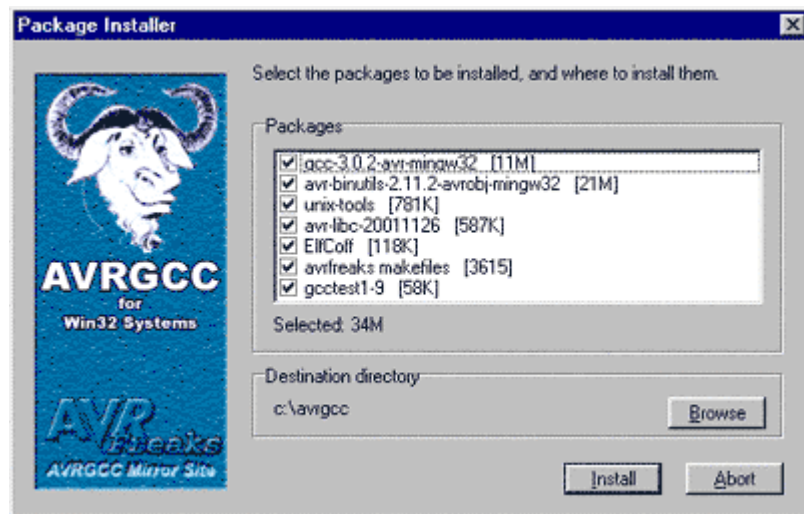
Installing AVRstudio

This should involve no problems. The self-extracting archive unpacks to the folder of your choice. Open this folder after decompression is finished, and run “install.exe”. Follow the instructions on-screen.

Installing AVRGCC

This is also a fairly straightforward process; just run the downloaded executable. For convenience, choose the default install location; “c:\avrgcc”.

Make sure to leave all seven boxes checked. You may not think you need the “Unix Tools”, but these include also the *make.exe* and *rm.exe* utilities; and you *will* need them.



Also, it is important to make a few checks to ensure that the installation, including the compilation of all libraries are completed:

- During installation, a DOS-window should appear, dumping lines of compiler output to the screen for about a minute or so.
- When compilation is done, another DOS window appears, declaring that “your pop-up program is ready to run”. Close this window by typing ctrl-c. **Note:** this does *not* happen on Windows2000.

If this does NOT occur, there could be a problem. The object libraries may not have been compiled properly. In this case, you need to run the file “run.bat” manually. It can be found in the directory where you installed avr-gcc.

- Provided you chose the default installation target directory to be “c:\avrgcc”, the directory “C:\avrgcc\avr\lib” should be crowded with .o files dated around the time you ran the installation. If not, execute *run.bat* manually.

Building a new project with AVRstudio and avr-gcc

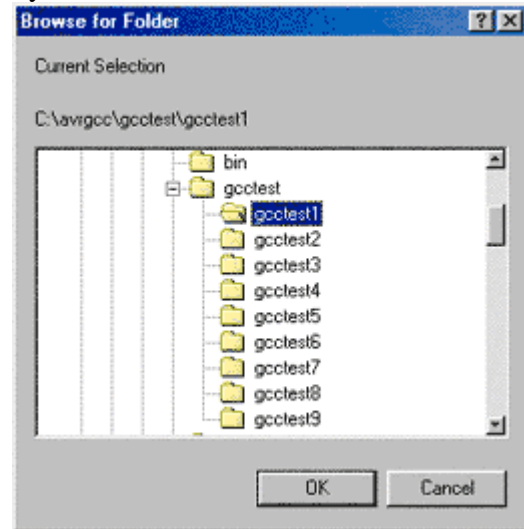
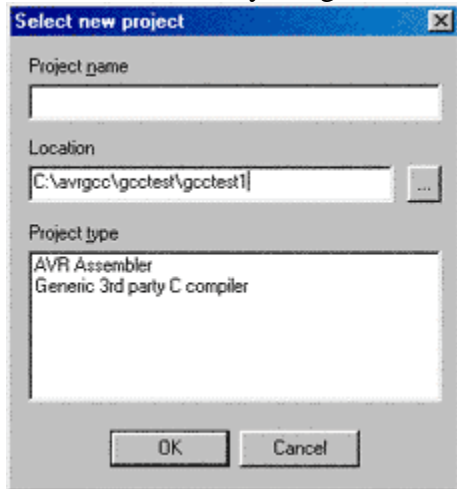
Now it's time to get down to it. We will open a new project in AVRstudio, add files to the project, and provide the necessary mechanics to make GNU make.exe take care of the building of our project.

The gcctest files by Volker Oth, included in our avrfreaks release of avr-gcc, will serve as examples. They should be located in the */gcctest* subdirectory of your avr-gcc installation. Keep them there or copy them wherever you want.

Make a new project with AVRstudio 3.53

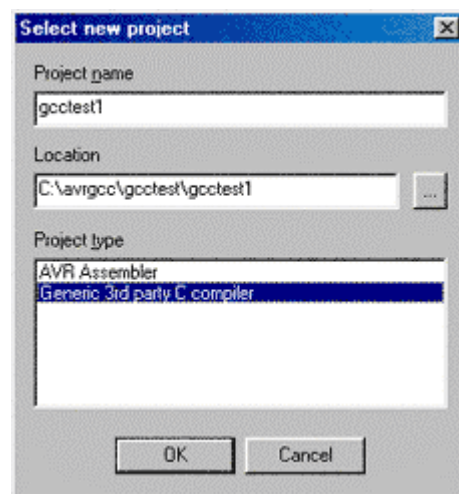
On the *Project* menu of AVRstudio, click *New* to see the *Select new project* dialog.

Enter a name for your new project, click the browse button to the right of the *Location* textbox and choose the directory of “gcctest1” as your project directory:



The selected project folder/directory becomes the location where all files generated by avr-gcc and AVRstudio will be output. This is also where you should keep *all* your project files.

When done, make sure to select “*Generic 3rd party compiler*” as your Project type before clicking the “OK” button. AVRGCC will be your generic 3rd party compiler. As stated in the AVRstudio manual, this feature is only recommended for advanced users. We’ll do it anyway.



Save the project.

The project concept, and using the GNU *make* utility

We will go for a concept where you stick to using a new makefile for each project, and hence; stick to using AVRstudio “projects” for each project. All the project files should be kept in the same directory/folder. This would be the folder you selected in the steps above.

This is necessary for AVRstudio’s Generic 3rd party compiler support to work properly.

The GNU tool *make* that comes with the avr-gcc distribution will use the makefile’s rules to compile and link our project into a complete .hex file to load into the mcu.

This is a neat, clean and comprehensible way of organizing your work.

As a template for your makefiles, you should use the makefiles included in the *gcctest* set. There is a makefile in each of the *gcctest* folders. They are mostly the same, but some of them have slight extensions. So, do not copy the one from *gcctest1* over to *gcctest2* and so on.

Note: If you did not download an AVRfreaks distribution including the *gcctest1-9* set, you have to compare the makefiles yourself and edit them according to this guide.

Adding files to the project

Now, we need to add the files that compose our little project. In the case of this little test project, we need:

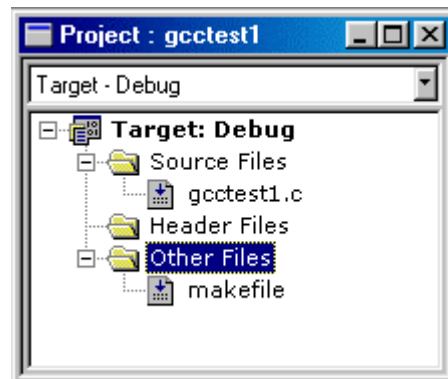
- Some source code.
- A makefile.

Let's start by including the C-sourcefile "*gcctest1.c*" :

- Right-click the "Source files" tab in the Project window of studio.
- Select "Add file", browse to the file "*gcctest1.c*" in your project folder.
- Double-click the file, or click "open".

Now add the makefile for this project:

- Right-click the "Other files" tab in the.
- Select "Add file", browse to the makefile.
If you cant see it, you may have to select "All files" in the filetype drop-down.
- Open it.



For every included file, you can double-click its representation in the folder tree of the project window to open and edit the file.

The makefile

The template makefile by Volker Oth should look like this in our version (included in the */avrfreaks* subdir of your avr-gcc installation):

```
# Simple Makefile Volker Oth (c) 1999
# edited by AVRfreaks.net nov.2001

##### change these lines according to your project #####

# put the name of the target mcu here (at90s8515, at90s8535, attiny22, atmega603 etc.)
MCU = at90s8515

# put the name of the target file here (without extension)
TRG    = gcctest1

# put your C sourcefiles here
SRC    = $(TRG).c

#put additional assembler source file here
ASRC   =

#additional libraries and object files to link
LIB     =

#additional includes to compile
INC     =

#compiler flags
CPFLAGS = -g -Os -Wall -Wstrict-prototypes -Wa,-ahlns=$(<:.c=.lst)

#linker flags
LDFLAGS = -Wl,-Map=$(TRG).map,--cref

##### you should not need to change the following line #####
include $(AVR)/avrfreaks/avr_make

##### dependencies, add any dependencies you need here #####
$(TRG).o : $(TRG).c
```

The only lines you should perhaps have to edit, are these:

- The line saying “MCU = at90s8515”.
This should be edited to reflect which AVR mcu you are using.
- The line saying: “TRG = gcctest1”. This is the name of your target. You can alter it to whatever you like, but gcctest1 is a good name. In this *gcctest* set, the makefiles are written so that the source file needs to have the same name as the core target name. If it doesn’t, *make* simply won’t find your sourcefile.
You can see this from the next line: SRC = \$(TRG).c
NB! Do not use any extension/suffix for the target name!

Note the line saying “include \$(AVR)/avrfreaks/avr_make” in the makefile. This line takes care of including more dependencies and commands for the make process, from the file “*avr_make*”. This file is

included in the avr-gcc distribution from AVRfreaks. For a quick introduction to what it contains; consult *appendix A* of this document.

Tying it all together

Okay, so now we have an open project, a source file and a makefile all ready to go. But still a little work remains to make all these things work together. Fear not; when done with these steps you will have a method to stick with for later that should be relatively simple to maintain.

A few things we have to realize at this point:

- AVRstudio knew nothing about avr-gcc; what it is or where it is, when we started out.
- Selecting “Generic 3rd party compiler” as project type did not initiate any magic event in your computer, so that AVRstudio suddenly knows which compiler and how to use it.
- Avr-gcc is run from the *command-line*, as well as the other GNU avr tools like the assembler, linker and the unix tools. They don’t have nice user interfaces. All these are most effectively controlled by the program *make*; run from the command line.

So the question is: *how do we make AVRstudio use avr-gcc?*

The trick is:

- To make AVRstudio aware of the path to the GNU tools via environment parameters.
- To run *make* and feed it with the right makefile.

Hence,

- Writing a *.bat* file that points out the right directories and runs *make*.
- Letting AVRstudio know that it should take a look at that file...

The procedure will differ slightly for Win9x and Win2000 users:

Using Win9x:

This file is also included in the */avrfreaks* subdir of your avr-gcc installation.

Let's do it; open a text editor and create a file consisting of these lines (provided that you installed avr-gcc to "c:\AVRGCC"):

```
@echo ----- begin -----  
@set AVR=c:\AVRGCC  
@set CC=avr-gcc  
@set PATH=c:\AVRGCC\bin  
make %1  
@echo ----- end -----
```

Save this file as "gcc_cmp.bat" in a directory on your computer that you *know* is included in the Windows *path*, i.e. "c:\windows" .

This file will set some important environment parameters for the *make* utility, then *make* will be run. The "%1" argument is the project folder path, provided by AVRstudio. The *make* utility by default looks for the file called "makefile" in this folder.

Using Win2000 or WinXP:

For Win2000, you need a "start"-file to kick off the compilation. Both of these files are available in the *\avrfreaks\win2000* subdir of your avrgcc installation:

```
@echo ----- begin -----  
@start /MIN /wait cmd /c gcc_cmp2.bat %1  
@type c:\tmpout.txt  
@del c:\tmpout.txt  
@echo ----- end -----
```

Save as "gcc_cmp.bat" in "c:\winnt", or some other folder you know is included in your path. Now, create a file consisting of these lines (again, provided that you installed avr-gcc to the same path as we did):

```
@set AVR=c:\avrgcc  
@set CC=avr-gcc  
@set PATH=c:\avrgcc\bin  
make.exe %1 >c:\tmpout.txt 2>&1
```

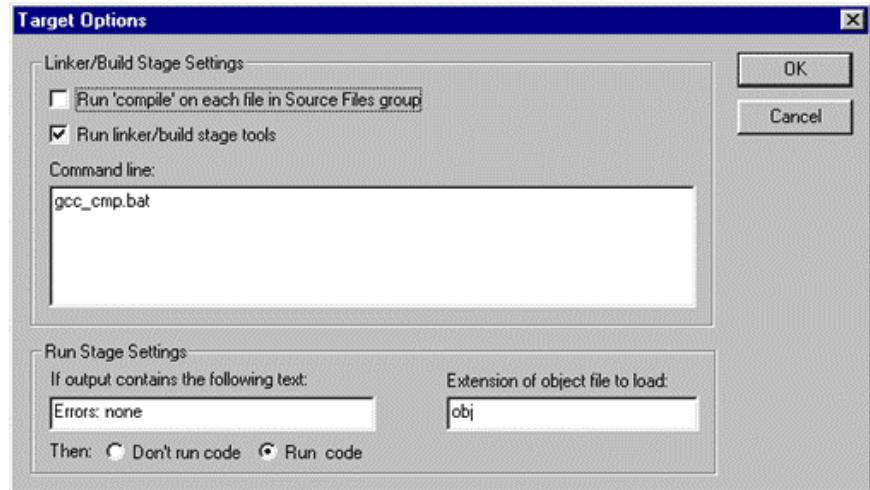
Save this file as "gcc_cmp2.bat" in the same folder as above.

This setup will start make.exe, which is instructed to direct its screen dump to a file that is displayed as the compiler output inside AVRstudio, and then deleted.

This is an important step!

Now, set the "target options" in AVRstudio:

- Right-click “Target: debug” in the Project window, and select “settings”.
- Uncheck “Run compiler...”.
- Check “Run linker...”
- Enter the name of your clever little `gcc_cmp.bat` file in the command line window.
- Under “Run stage settings”, opt for “Run code”. In the first text box type “Errors: none”, in the second type “obj” for object file extension.



And here we go...

Building our first project with AVRstudio and avr-gcc

Now you should be all ready to run. Just double-click the makefile icon in the Project window of AVRstudio, and make sure that this file is entered as shown above. Especially note that the line setting your target name is correct. As you can see from the line below it, this should be the same name as your C source file.

To build the project:

- Right-click “Target: debug” in the project window of AVRstudio, and select “build” from the bottom of the menu.

Provided you completed all we went through so far as you should, this project should build just fine. Watch the the *make* output appearing in a window inside AVRstudio. As long as it doesn’t report any errors; all is OK:


```

----- begin -----
C:\avrgcc\gcctest\gcctest1>make
avr-gcc -c -g -Os -Wall -Wstrict-prototypes -Wa,-ahlns=gcctest1.lst -mmcu=at90s8515 -l. gcctest1.c -o gcctest1.o
avr-gcc gcctest1.o -Wl,-Map=gcctest1.map,--cref -mmcu=at90s8515 -o gcctest1.elf
avr-objcopy -O avrobject .eeprom gcctest1.elf gcctest1.obj
avr-objcopy -O ihex -R .eeprom gcctest1.elf gcctest1.hex
elf2hex gcctest1.elf -R .eeprom gcctest1.coff gcctest1.sym
Ended
cp coff\gcctest1.coff .
cp coff\*.sym .
cp coff\*.S .
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex gcctest1.elf gcctest1.eep
Errors: none
----- end -----

```

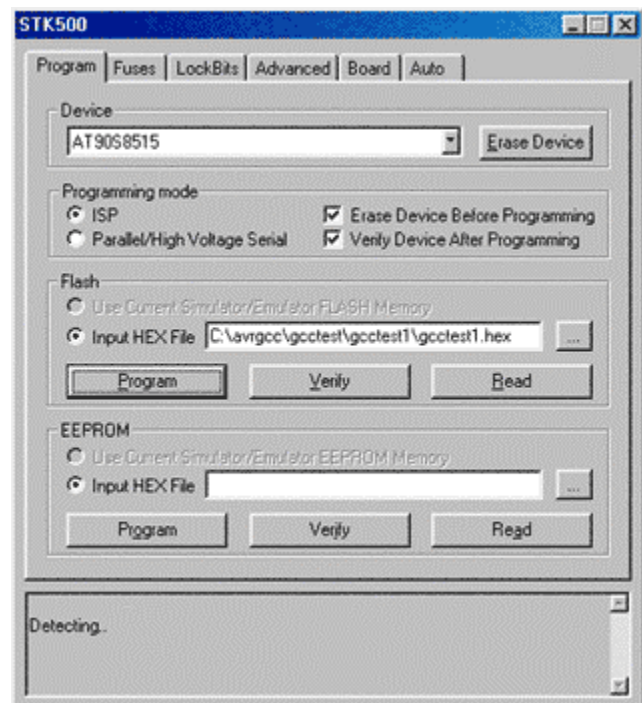
Programming the AVR in the STK500

Helping you unpack, install and troubleshoot your STK500 development kit is out of scope of this document. If you are having trouble with this; consult article written by Sean Ellis on AVRfreaks.net, dec. 2001. But you should always:

- Make sure the STK500 is connected, and switched on.
- For this project, connect PortB to the LEDs connector with the 10-pin jumper cable.

To load your newly built code onto the chip, click the nice little AVR chip icon on the toolbar of AVRstudio, or open the *STK500 tool menu* by selecting “STK500/...” from the “Tools” menu:

- You can probably leave most of the settings to their default values.
- Make sure you load the right binary to the chip. In the “Flash” section of the dialog box; browse to the file “gcctest1.hex” in your project directory. You may have to select “other files” in the filetype drop-down box to see the – rom file.
- Click the “Program” button.
- Make sure the output textbox at the bottom of the dialog says everything went OK.



The LEDs on your STK500 board should light up, and it's Christmas time again. Deck the hall with bowls of holly and enjoy. Not. Because what happens in my case, at least, is that the LEDs don't

flash. This is not a running light. It should be, and the reason it's not running is the optimization level of the compiler.

Tampering the makefile

Have a look at the compiler flags line in your makefile:

```
#compiler flags
CPFLAGS = -g -Os -Wall -Wstrict-prototypes -Wa,-ahlns=$(<:.c=.lst)
```

The `-O` flag indicates the optimization level for the compiler. This is set pretty tight. If you also consult your C source file, you will see that the delays between each output is implemented with counting loops. No timer is involved to ensure an appropriate delay. What happens here is that the delays are optimized away. They are actually gone; since the compiler realized they did nothing useful, it left them out.

To work around this, edit the makefile by clicking the makefile icon in the Project window of AVRstudio. Change the following line:

```
#compiler flags
CPFLAGS = -g -Os -Wall -Wstrict-prototypes -Wa,-ahlns=$(<:.c=.lst)
```

To:

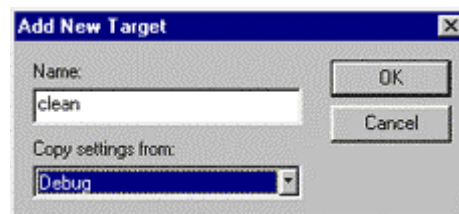
```
#compiler flags
CPFLAGS = -g -O1 -Wall -Wstrict-prototypes -Wa,-ahlns=$(<:.c=.lst)
```

The optimization level has been set to 1. The previous “s” (size) optimization key provides optimizations for code size. Consult `avr-gcc` man pages for more information.

Make clean

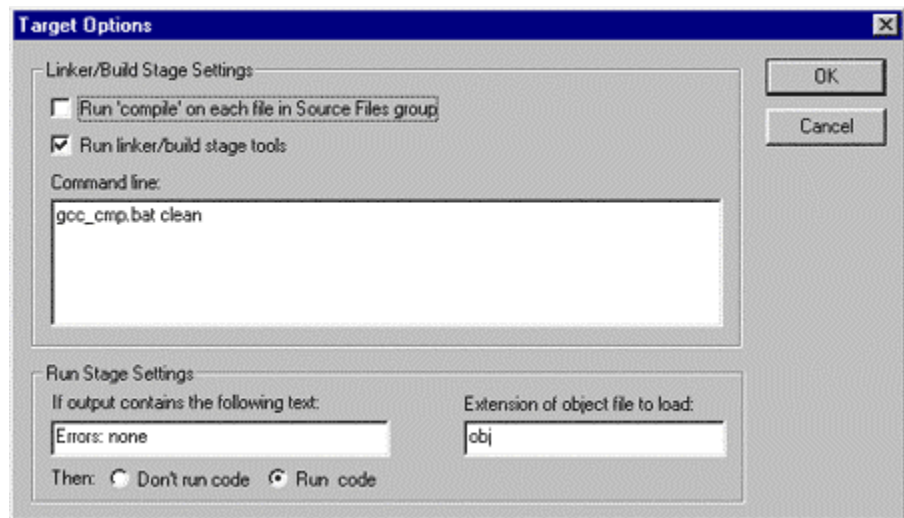
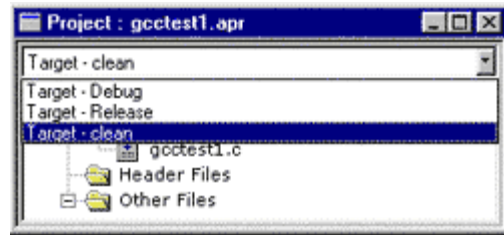
Important note: You *must* clean out the products from the previous build to make this new build. Or else, *make* will decide that a rebuild is not necessary, since it is not set up to look for changes in the makefile. We will not clean out these files manually, instead we will add a new *Target* in AVRstudio that performs the cleaning. Oh yes.

- Right click *Target: debug* in the Project dialog of AVRstudio; select *Targets -> add....* The *Add new target* dialog appears. Enter the name “clean” for your new target, and select *Debug* from the drop-down



below (“Copy settings from”).

- The target “clean” will now be available from the Targets drop-down in the Project dialog.
- Select this target, and right-click to access its *settings...*
- You can see that it inherited the settings from the *Debug* target. Just add the keyword “clean” behind the call to the .bat file you made.
- Build the *clean* target.



This will clean out products from the make process. Why this works? “Clean” is a defined target in the *avr_make* makefile included in your *avr-gcc* package. Consult Appendix A for a quick peek at this file.

- Go back to the *Debug* target.
- Now rebuild all, reprogram the AVR and see what happens.

If you try to rebuild this project once more without making any changes, *make* will simply abort the build. This is because *make* is, through the makefile, set up to only rebuild the parts of the project that *needs* to be rebuilt at any given time. So, when *make* discovers there are no recent output files, it decides to rebuild the whole shebang. But the makefile itself is not included as one of the dependencies. So a change in this file will not *automatically* initiate a rebuild. But, reasonably enough, a change in a source file will.

This is of no significance to our little single-source-file project, but it will be to larger projects.

Now, if all went well you should:

- Take a deep breath.
- Save your project.
- Sit back for a few minutes, marvelling at the wonders of science.

You could also check out the quick overlook of the *avr_make* file included in the avr-gcc distribution, in *appendix A* of this document.

Debugging with .coff files in AVRstudio

Now, there's something missing with our setup still:

The *.obj file that avr-gcc generates does not contain the necessary information to let AVRstudio watch variables during debugging. The standard gcc file *.elf actually includes all this information; but AVR Studio does not support the ELF format. It does, however, support the COFF format with variable watch.

Flavio Gobber, an AVRfreaks.net registered user, has made an ELF to COFF converter. It is included and integrated in AVRfreaks distributions of avr-gcc later than 20011203a, for your convenience. These distributions are set up so that .cof files should be output in your working directory if you followed the instructions so far. Then all you have to do is simply open the .cof file in AVRstudio, add breakpoints and watches, and off we go.

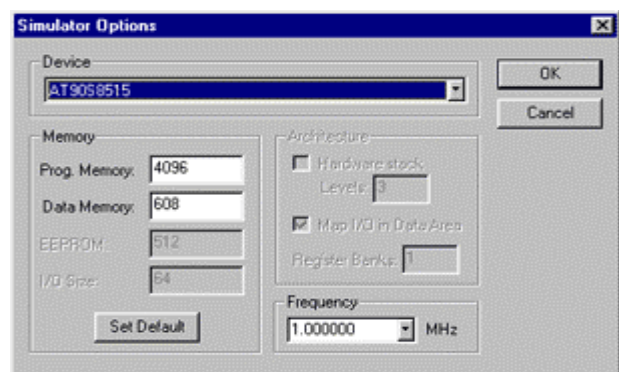
NB! “Elf2coff” is also available from AVRfreaks.net:

<http://www.avrfreaks.net/AVRGCC/download.php> in case you are not using our preferred release of avr-gcc. To set it up on your own, refer to the Appendix B of this document.

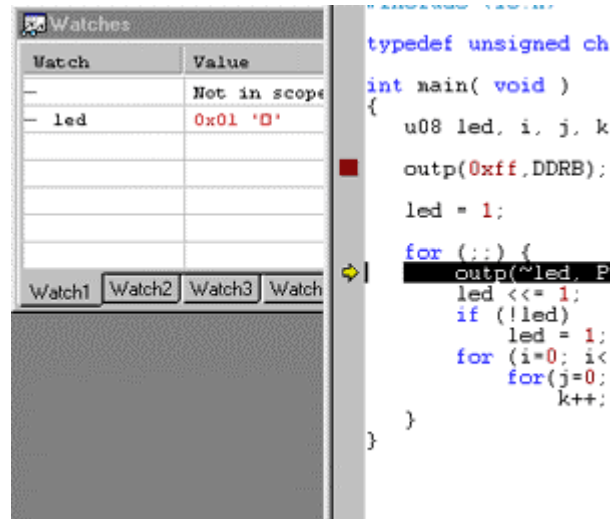
An example:

After successfully building your *gcctest1* project, click to open the .cof file that should have been output in your project directory:

- A new window should appear, showing you the .coff dump with lots of directives and chunks of assembly code, while your project window disappears.
- The “*Simulator options*” dialog appears. Select your device and the clock frequency.
- Reopen your project, and click to see the C source file.
- Scroll to this line in the *main* function:
 outp(0xff,DDRB);
- Press F9 to set a breakpoint here.
- Open the *Add watch* window from the *Watch* menu; and add a watch for “led” (right-click).
- Press Shift-F5 to reset the debug; the focus should change to the coff file window. Run with F5. Focus changes to the source code window with a break at the line you marked...
- Click F10 to single-step, and watch the variable “led” change to “0x01”...
- Hark! The herald angels sing.



- That's all there is to it.



Now, you should be on your way developing for the AVR family with avr-gcc and AVRstudio. The setup provided through the steps in this document should be sufficient for some time. Start by going through the *gcctest1-9* example sets. Also, download a copy of Rich Neswold's "GNU development Environment for the AVR microcontroller" from www.avrfreaks.net/AVRGCC/, as a nice introduction and reference.

You may need to extend or alter this setup as you become more experienced and start working on more complex projects. By then you will also probably know more about how to make the necessary changes and fixes.

Finally:

- If you haven't already downloaded the document "GNU development environment for the AVR Microcontroller", you should do that now.
- Frequent the AVR GCC forum at AVRfreaks.net
- Good luck.

Appendix A – the “avr_make” file

Listed below are the lines that constitute the file “avr_make”. The file is composed from the *make1-2* files by Volker Oth, jan.2000. It can be found in the subdirectory *c:\avrgcc\avrfreaks*, provided *c:\avrgcc* is the directory where you installed the avr-gcc distribution. “Block” markers; “BLOCK1, ...” etc are inserted in this file for easy reference. Comments for each block are included below. Reading through them should be an easy way of getting some idea of how this works.

```
#----- START OF FILE -----
# GCC-AVR standard Makefile part 3
# Based on Volker Oth's makefiles of jan.2000
# Modified and merged by AVRfreaks.net for smoother integration with AVR Studio,
# and easier comprehension for the average user (nov.2001). Minor errors corrected.
# -----

##### BLOCK 1) define some variables based on the AVR base path in $(AVR) #####

CC      = avr-gcc
AS      = avr-gcc -x assembler-with-cpp
RM      = rm -f
RN      = mv
OUT     = coff
ELFCOF  = elfcoff
CP      = cp
BIN     = avr-objcopy
SIZE    = avr-size
INCDIR  = .
LIBDIR  = $(AVR)/avr/lib
SHELL   = $(AVR)/bin/sh.exe

##### BLOCK 2) output format can be srec, ihex (avrobj is always created) #####

FORMAT = ihex

##### BLOCK 3) define all project specific object files #####

OBJ      = $(ASRC:.s=.o) $(SRC:.c=.o)
CPFLAGS += -mmcu=$(MCU)
ASFLAGS += -mmcu=$(MCU)
LDFLAGS += -mmcu=$(MCU)

##### BLOCK 4) this defines the aims of the make process #####

all:      $(TRG).obj $(TRG).elf $(TRG).hex $(TRG).cof $(TRG).eep $(TRG).ok

##### BLOCK 5) compile: instructions to create assembler and/or object files from C source #####

%.o : %.c
    $(CC) -c $(CPFLAGS) -I$(INCDIR) $< -o $@

%.s : %.c
    $(CC) -S $(CPFLAGS) -I$(INCDIR) $< -o $@

##### BLOCK 6) assemble: instructions to create object file from assembler files #####
```

```

%.o : %.s
    $(AS) -c $(ASFLAGS) -I$(INCDIR) $< -o $@

##### BLOCK 7) link: instructions to create elf output file from object files #####
%.elf: $(OBJ)
    $(CC) $(OBJ) $(LIB) $(LDFLAGS) -o $@

##### BLOCK 8) create avrobj file from elf output file #####

%.obj: %.elf
    $(BIN) -O avrobj -R .eeprom $< $@

##### BLOCK 9) create bin (.hex and .eep) files from elf output file #####

%.hex: %.elf
    $(BIN) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
    $(BIN) -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O $(FORMAT) $< $@

%.cof: %.elf
    $(ELFCOF) $< $(OUT) $@ $*.sym
    $(CP) $(OUT)\$@ .
    $(CP) $(OUT)\$*.sym .
    $(CP) $(OUT)\$*.S .

##### BLOCK 10) If all other steps compile ok then echo "Errors: none" #####

%ok:
    $(SIZE) $(TRG).elf
    @echo "Errors: none"

##### BLOCK 11) make instruction to delete created files #####

clean:
    $(RM) $(OBJ)
    $(RM) $(SRC:.c=.s)
    $(RM) $(SRC:.c=.lst)
    $(RM) $(TRG).map
    $(RM) $(TRG).elf
    $(RM) $(TRG).cof
    $(RM) $(TRG).obj
    $(RM) $(TRG).a90
    $(RM) $(TRG).hex
    $(RM) *.bak
    $(RM) *.log

size:
    $(SIZE) $(TRG).elf
#----- END OF FILE -----

```

Note: Make sure not to delete any tabs preceding a command in this file, as they are crucial for the *make* utility's ability to recognize commands from dependency rules. On the other hand, make sure to delete all *accidental* spaces etc. in variable declarations. These may for instance cause your shell not to recognize an executable or an executable to not recognize a parameter...

Comments

Block 1 : this block provides some additional pointers to folders and utilities needed in the *make* process. Recall that the variable called AVR in our case is “c:\avrgcc”. So, the string

```
LIBDIR = $(AVR/avr/lib)
```

simply sets the variable LIBDIR to “c:\avrgcc\avr\lib”. This block provides a single entry point for fundamental changes in the setup.

Block 2: sets optional binary output format to Intel hex.

Block 3: the first line defines the names of all object files to be created, from the names of all included source files (both assembler and C source files). These are to be entered in the *makefile*. Note also the MCU variable, referred as \$(MCU), which is set to the type of AVR microcontroller to use (i.e. AT90s8515), in your makefile.

Block 4: the aims are to create the target files listed; .obj, .elf, .hex... The order of the files are indifferent, but the order of the following blocks in the makefile, is not.

Block 5: This is the first *target : dependency* line in this file. Such lines are called *rules*, and the following lines are the *commands* that make the rule. All commands are preceded by a tab; this is imperative to make it work. If the tab isn't there, the command is skipped.

This rule says that all .o files depend on the corresponding .c file. The “%” is a wildcard. Hence, the following line compiles any .c file into a .o file:

```
%o : %.c
    $(CC) -c $(CPFLAGS) -I$(INCDIR) $< -o $@
```

The \$< and \$@ are *automatic* variables for make, meaning “the filename of the first dependency” and “the filename of this rule's target”, respectively. The target of this rule is any object file, .o. Hence, in this line when the source “gcctest1.c” comes wandering along; it is compiled into the file “gcctest1.o”.

Block 6: Similar to the above; this time for assembly source files. Note that other flags are included, and that avr-gcc is called with other directives than when compiling C source files (the \$(AS) variable).

The next few blocks work in a similar manner.

Block 10: Remember that all rules are performed sequentially in order of appearance in the file? Well, if we got this far without make aborting the process, all is OK. Hence, we report to AVRstudio “Errors: none”. This step is actually necessary for AVRstudio to see that all the previous steps went OK. Also, a call to *avr-size.exe* is performed. This outputs the size of the elf file. Hence, you can see how code changes affect the size of the resulting code.

Block 11: These few targets do not conform to the *target : dependencies* form of rules in a regular fashion, since they have no actual target files. No actual files are to be built as the result of these rules. Such rules are called *phony rules*. They are useful for things like cleaning up, like this first

one, which deletes all files according to its rules when called. This step will never be performed automatically, but *make clean* can be called explicitly to perform it.

Take a look again at the rule in Block 4. This is the default rule, which defines the goals of the entire make process, and is always performed when nothing specific is called for. This is also a phony target. We *could* include *clean* as one of the dependencies, and have it clean up intermediate files etc.

Note: when making a change to, for instance, the compiler flags in the makefile; a complete remake is necessary. But the makefile is not commonly considered a dependency of any project. Hence, it is not included in any rule.

If it was for instance included as a dependency for the assembler and object files in block 5 and 6 above, the entire project would have to be rebuilt every time the *makefile* had changed (i.e. every time *make* could tell that the *makefile* was newer than the source files and object files. So it usually isn't.

The safest/simplest thing when you have changed some entries in the makefile, is to run “*make clean*” to clean out the directory and have *make* build the whole project all over. If working in a windows environment, you can also delete all the files produced by *make* manually (all the extensions listed in block 11).

Appendix B – setting up “elf2coff” with AVRstudio and make

The standard gcc file *.elf actually includes all information for watching variables while debugging; but AVR Studio does not support the ELF format. It does, however, support the COFF format with variable watch. We will utilize “Elf2Coff”, a utility made by AVRfreaks user Flavio Gobber, that converts .elf to .cof files. These files can be directly loaded into studio for happy debugging.

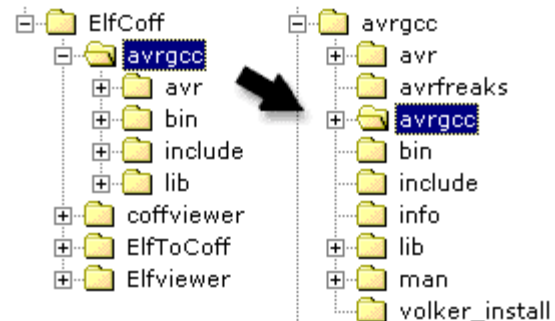
Setting up Elf2Coff to run with avr-gcc and AVRstudio is not too hard, when you know what to do. Since information on how to accomplish this is hard to come by (we know only a single source for this – the AVR-GCC forum at AVRfreaks.net...), it is compiled and provided here for your pleasure and convenience.

Note: If you already downloaded an avr-gcc distribution from AVRfreaks.net dated dec.03 or later; Elf2Coff is already integrated into the package, and you don’t need to proceed with this. Coff files should be automatically output to your project directory. That is; provided you follow the directions in the main part of this document.

Just do it

Now, this shouldn’t take long. Lets’ get started:

- Download *Elf2Coff* from www.AVRfreaks.net/AVRGCC
- Unzip into any directory.
- Inspect the unzipped structure (seen to the right). The interesting part is the subdirectory “avrgcc”.
- Copy this whole subdir structure into the directory where you installed avr-gcc (i.e. “c:\avrgcc\”). You get a new node called c:\avrgcc\avrgcc\... with four subdirs in it.
- Make sure you copy the file “elfcoff.exe” from the \elfcoff\avrgcc\bin directory, into the \bin directory of your avr-gcc installation (should be c:\avrgcc\bin).



You also need to make some changes in the *avr_make* file, see Appendix A:

- In Block 1, add these line (use tabs to align with the other entries in this block):

```
ELFCOF = elfcoff
OUT = Coff
```

CP = cp

- In Block 4, add this entry for the .cof target file: \$(TRG).cof
Now, this line reads:

```
all: $(TRG).obj $(TRG).elf $(TRG).cof $(TRG).hex $(TRG).eep $(TRG).ok
```

- In Block 9, add this new rule (watch the tabs!):

```
%.cof: %.elf

$(ELFCOF) $< $(OUT) @$ $*sym
$(CP) $(OUT)\$@ .
$(CP) $(OUT)\$*sym .
$(CP) $(OUT)\$*S .
```

- ...and finally add this to the clean routine in Block 11:

```
$(RM) $(TRG).cof
```

Note: make sure to trim all unnecessary spaces when adding keywords to a makefile. An accidental tab or space may cause a step in the *make* process to fail. The output from the compilation should look like this:

```
----- begin -----
C:\WINDOWS\Desktop\AVRprojects\testgoc\gcctest\gcctest1>make
avr-gcc -c -g -Os -Wall -Wstrict-prototypes -Wa,-ahlns=gcctest1.lst -mmcu=at90s8515 -I. gcctest1.c -o gcctest1.o
avr-gcc gcctest1.o -Wl,-Map=gcctest1.map,--cref -mmcu=at90s8515 -o gcctest1.elf
avr-objcopy -O avrobj -R .eeprom gcctest1.elf gcctest1.obj
elfcoff gcctest1.elf Coff gcctest1.cof gcctest1.sym
Ended
cp Coff\gcctest1.cof .
cp Coff\*sym .
cp Coff\*S .
avr-objcopy -O ihex -R .eeprom gcctest1.elf gcctest1.rom
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex gcctest1.elf gccte...
Errors: none
----- end -----
```

Good luck.