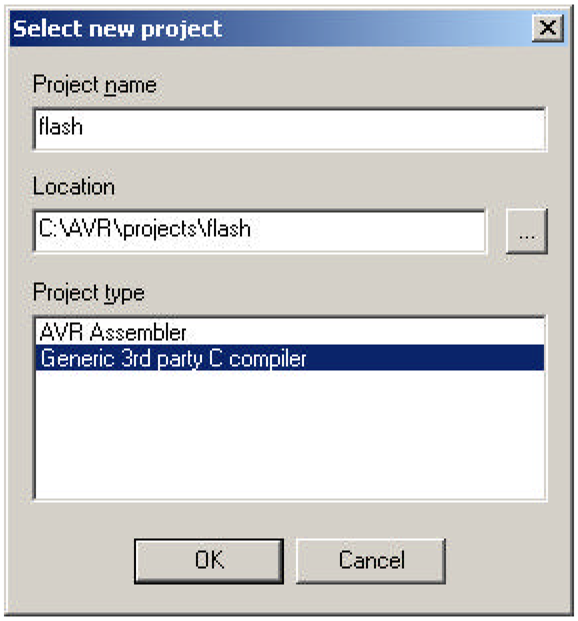
(This document is adapted from Avr-gcc/AVRstudio beginner's guide dec. 14,2001, http://www.avrfreaks.net/AVRGCC/)

Part1: Building a new project with AVRstudio and avr-gcc

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.
Go to *C:\AVR*. There should be a directory called *C:\AVR\projects*. If the directory isn't there, create it.
Go to *C:\AVR\projects*. The directory should be empty. If not, delete all contents. This is your working directory for development, and all files stored here should be saved to your share. **When you are finished for a session, save all files and folders to your share, and delete them here, so the next user doesn't benefit from your hard work.**
We have provided an example folder in the course file area:
\\ntdfs\cs\cse\courses\cse466\02au.
Find a folder called "flash" and copy the folder to *C:\AVR\projects*.

Make a new project with AVRstudio 3.53

On the *Project* menu of AVRstudio, click *New* to see the *Select new project* dialog. Enter the name *flash* for your new project, click the browse button to the right of the *Location* textbox and choose the directory of "C:\AVR\projects\flash" 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 3 rd party compile***r" as your Project type before clicking the "OK" button. AVRGCC will be your generic 3 rd party compiler.

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 3 rd party compiler support to work properly.
The GNU tool *make* that comes with the avr-gcc distribution will use the makefile's rules to compileand 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 makefile included here.

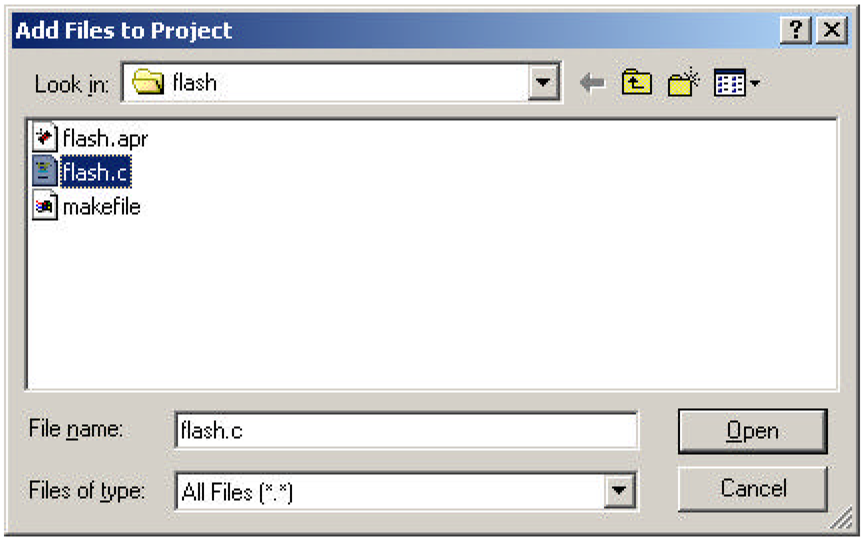
Adding files to the project

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 *"flash.c"* :

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



flash

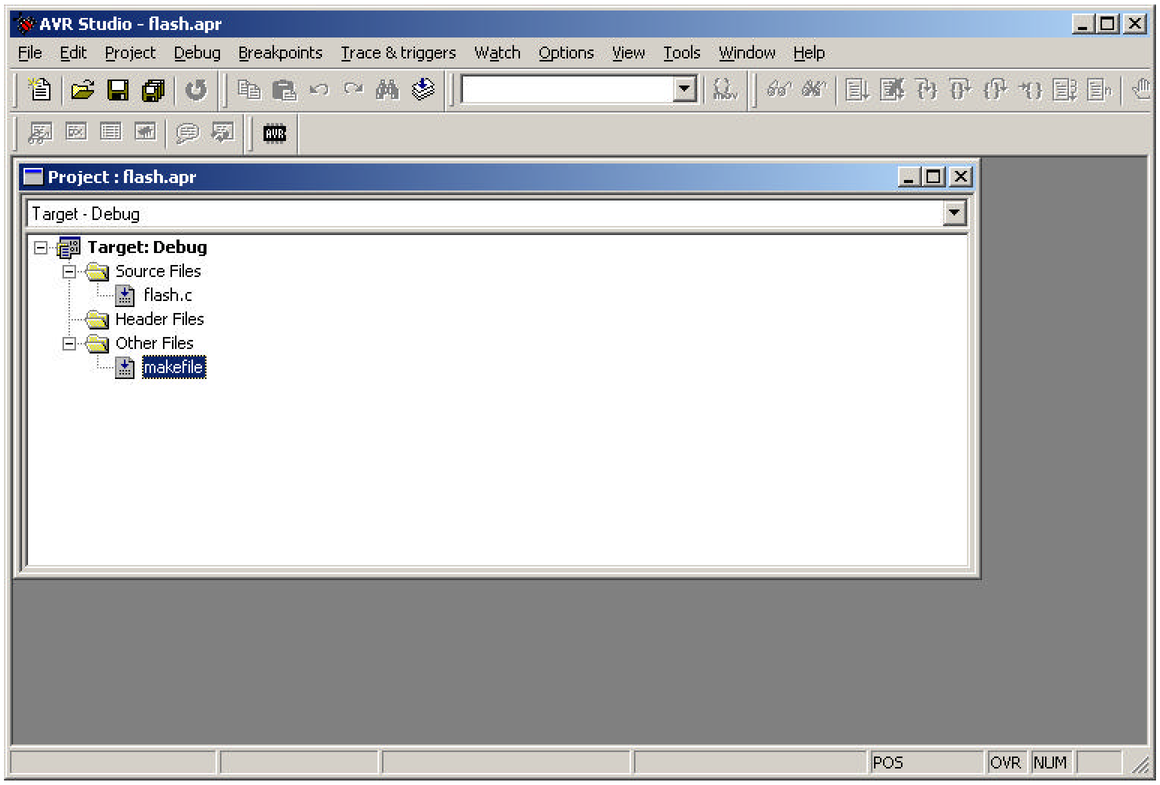
flash.apr
flash.c
makefile

File name: flash.c
Files of type: All Files (*.*)

Now add the makefile for this project:

- Right-click the "Other files" tab in the.
- Select "Add file", browse to the makefile. If you can't see it, you may have to select "All files" in the filetype drop-down.

Our project should now look like this:

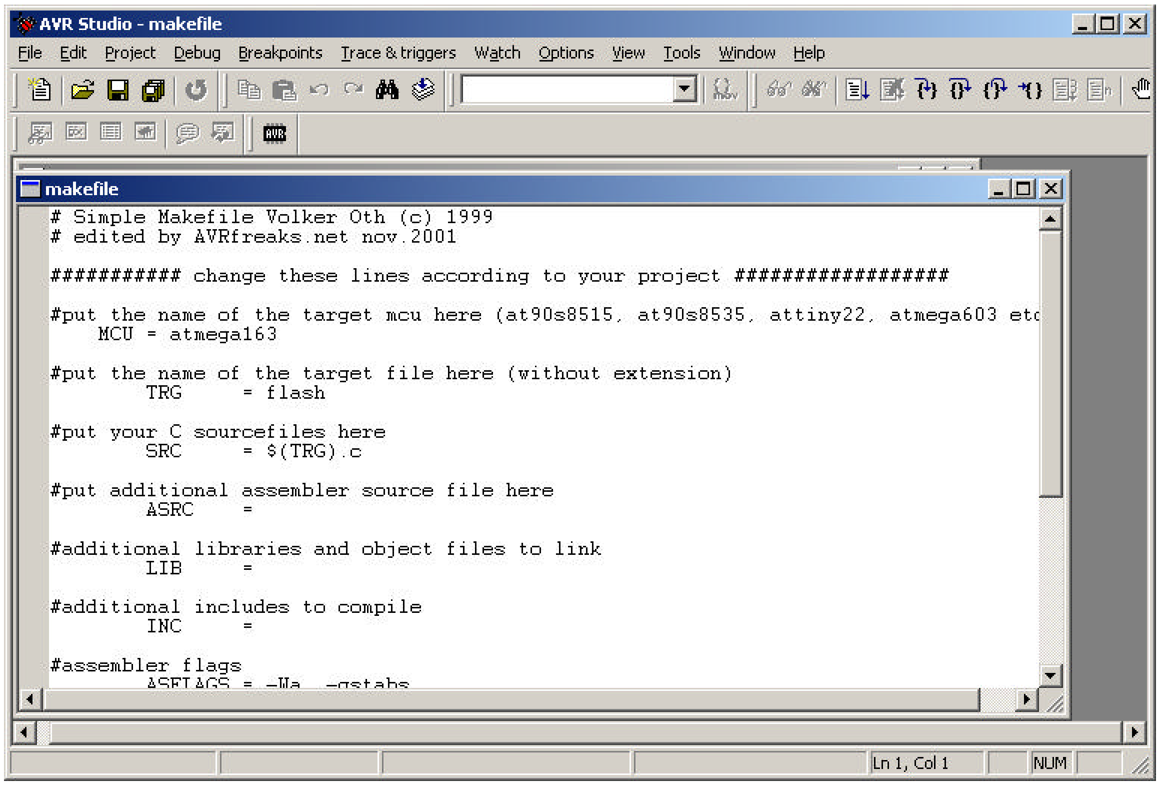


For every included file, you can double-click its representation in the folder tree of the project window to open and edit the file.

Open the makefile.

The makefile

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


AVR Studio - makefile
File Edit Project Debug Breakpoints Trace & triggers Watch Options View Tools Window Help
makefile
Ln 1, Col 1
NUM

```
# 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 = atmega163

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

#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     =

#assembler flags
        ASFLAGS = -Wa, -gstabs
```

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

- The line saying: "TRG = flash ". This is the name of your target.

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 the *Avr-gcc/AVRstudio beginner's guide dec. 14,2001, http://www.avrfreaks.net/AVRGCC/)*.

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 3 rd 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…

.
Using Win2000 or WinXP:
For Win2000, you need a "start"-file to kick off the compilation

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

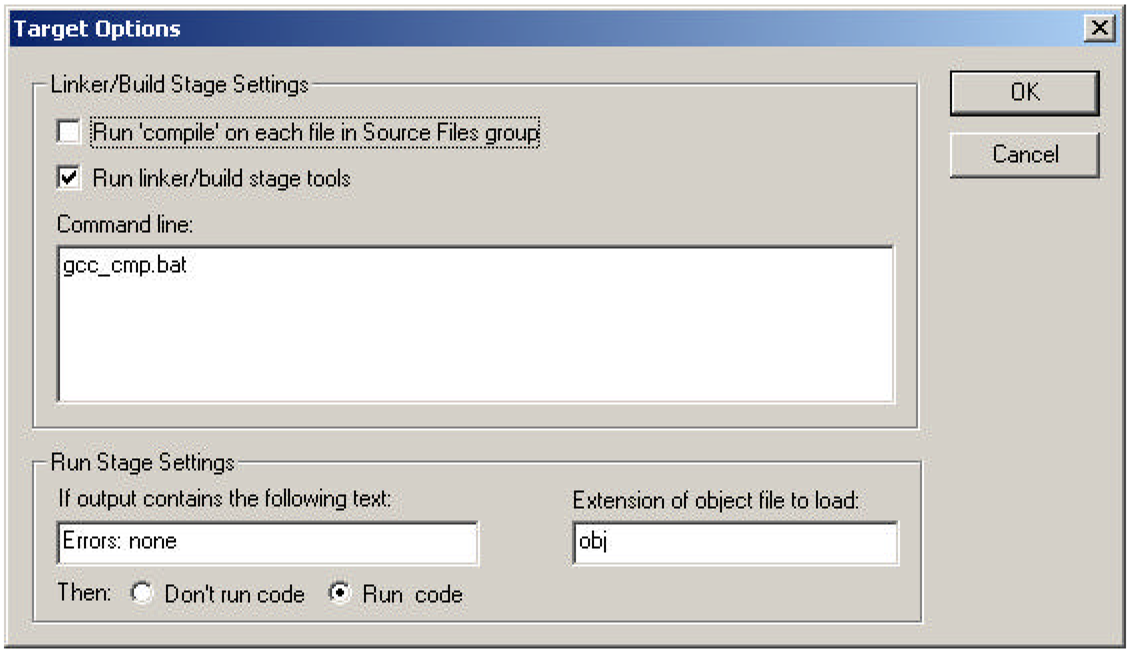
```
@set AVR=c:\avrgcc
@set CC=avr-gcc
@set PATH=c:\avrgcc\bin
make.exe %1 >c:\tmpout.txt 2>&1
```

These files have already been modified and saved somewhere in your path.

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:



Linker/Build Stage Settings

[ ] Run 'compile' on each file in Source Files group

[✓] Run linker/build stage tools

Command line:

```
gcc_cmp.bat
```

Run Stage Settings

If output contains the following text: Errors: none

Extension of object file to load: obj

Then: Don't run code [✓ Run code]

- Right-click "Targe t:debug" in the Project window, and select "settings".
- Uncheck "Run compiler…".
- Check "Run linker…"
- Enter the name of our 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…

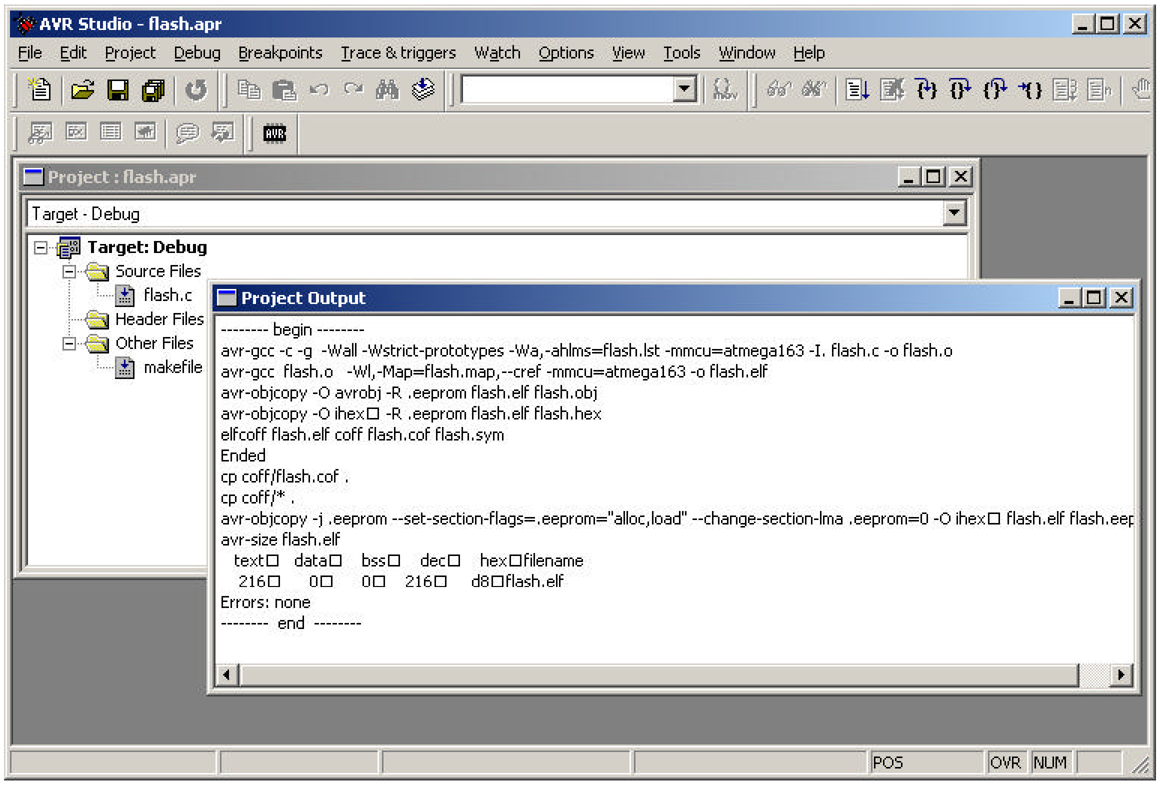
Building our first project with AVRstudio and avr-gcc

Now you should be all ready to run.

To build the project:

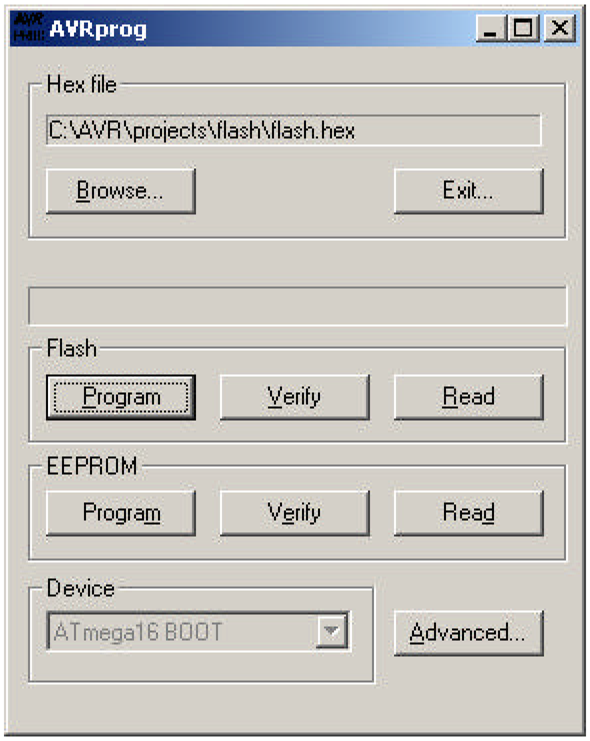
- Right-click "Target: debug" in the project window of AVRstudio, and select "build" from the bottom of the menu.

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 --------
avr-gcc -c -g  -Wall -Wstrict-prototypes -Wa,-ahlms=flash.lst -mmcu=atmega163 -I. flash.c -o flash.o
avr-gcc  flash.o   -Wl,-Map=flash.map,--cref -mmcu=atmega163 -o flash.elf
avr-objcopy -O avrobj -R .eeprom flash.elf flash.obj
avr-objcopy -O ihex□ -R .eeprom flash.elf flash.hex
elfcoff flash.elf coff flash.cof flash.sym
Ended
cp coff/flash.cof .
cp coff/* .
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex□ flash.elf flash.eep
avr-size flash.elf
   text□  data□   bss□   dec□   hex□filename
    216□     0□     0□   216□    d8□flash.elf
Errors: none
--------  end  --------
```

Programming the AVR with AVRprog



Now you're ready to test your code. The TA will bring the test board, and connect it to your serial port. Make sure it’s on, and one LED segment is blinking.

To load your newly built code onto the chip, select “AVR prog” from the “Tools” menu.

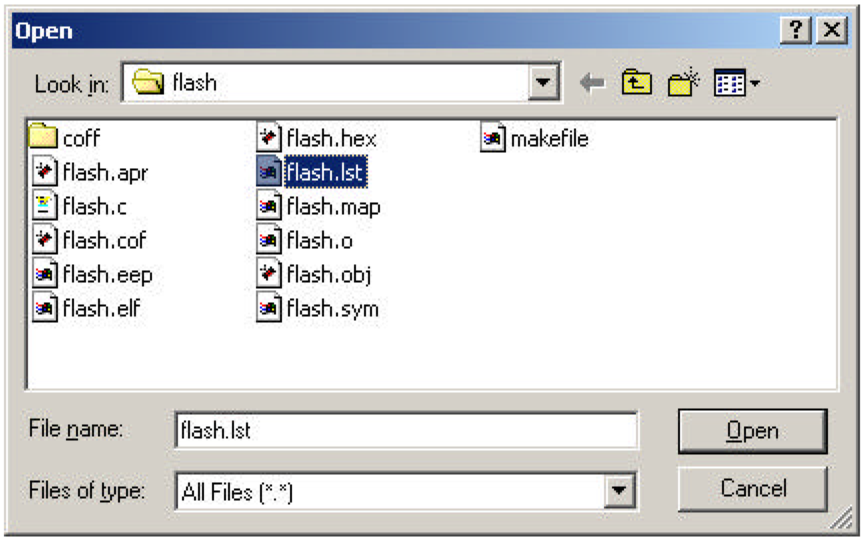
Select the “flash.hex” file, and click “Flash Program”.

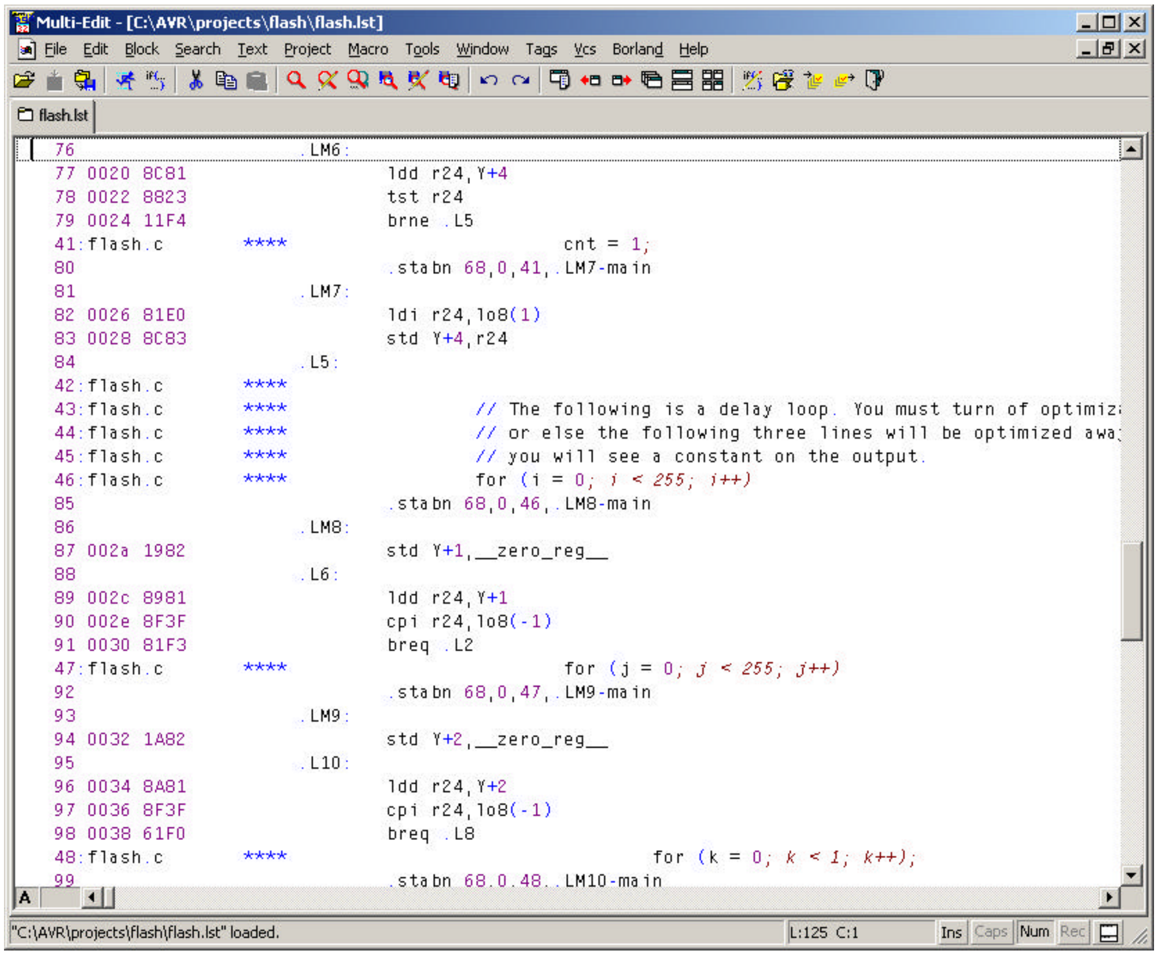
The blue progress bar should progress all the way to the right. Ignore the “ffff” error message.

Press the reset button on the test board; your code should run.

Save your work to your share!

You can open the “flash.lst” file to examine the output of the compiler:




Multi-Edit - [C:\AVR\projects\flash\flash.lst]
File Edit Block Search Text Project Macro Tools Window Tags Vcs Borland Help
flash.lst
"C:\AVR\projects\flash\flash.lst" loaded.
L:125 C:1
Ins Caps Num Rec

```
76                          .LM6:
77 0020 8C81                       ldd r24,Y+4
78 0022 8823                       tst r24
79 0024 11F4                       brne .L5
41:flash.c           ****                      cnt = 1;
80                                 .stabn 68,0,41,.LM7-main
81                          .LM7:
82 0026 81E0                       ldi r24,lo8(1)
83 0028 8C83                       std Y+4,r24
84                          .L5:
42:flash.c           ****
43:flash.c           ****               // The following is a delay loop. You must turn of optimiz
44:flash.c           ****               // or else the following three lines will be optimized awa
45:flash.c           ****               // you will see a constant on the output.
46:flash.c           ****               for (i = 0; i < 255; i++)
85                                 .stabn 68,0,46,.LM8-main
86                          .LM8:
87 002a 1982                       std Y+1,__zero_reg__
88                          .L6:
89 002c 8981                       ldd r24,Y+1
90 002e 8F3F                       cpi r24,lo8(-1)
91 0030 81F3                       breq .L2
47:flash.c           ****                      for (j = 0; j < 255; j++)
92                                 .stabn 68,0,47,.LM9-main
93                          .LM9:
94 0032 1A82                       std Y+2,__zero_reg__
95                          .L10:
96 0034 8A81                       ldd r24,Y+2
97 0036 8F3F                       cpi r24,lo8(-1)
98 0038 61F0                       breq .L8
48:flash.c           ****                             for (k = 0; k < 1; k++);
99                                 .stabn 68,0,48,.LM10-main
```

(You may need to use some other editor to examine this file.)

Part 2: Building a new project with AVRstudio 4 in Assembler

Now, create a version of this program in assembler. Use the AVRstudio 4 in the AVR folder.

Refer to the HELP menu for guidance, and check AVRfreaks and the ATMEL CD for help in using Studio 4. Also, look for examples. Any code that will run on the STK500 will work. Set up your assembly code for the Atmega16. Call your source file "flash[your_logon_name].asm".

We'll leave the test board in the lab for your use in debugging. Downloading is similar in Studio 4, using AVRprog.

ASSIGNMENT:

Each member of your lab team should complete this assignment individually. You may help each other, but collaboration should be indicated in source code comments.

Turn in your completed assembly source code. It's due at the start of Lab Session 2.

Good luck!

--END--