How to use EGNU/GCC and uBUG12

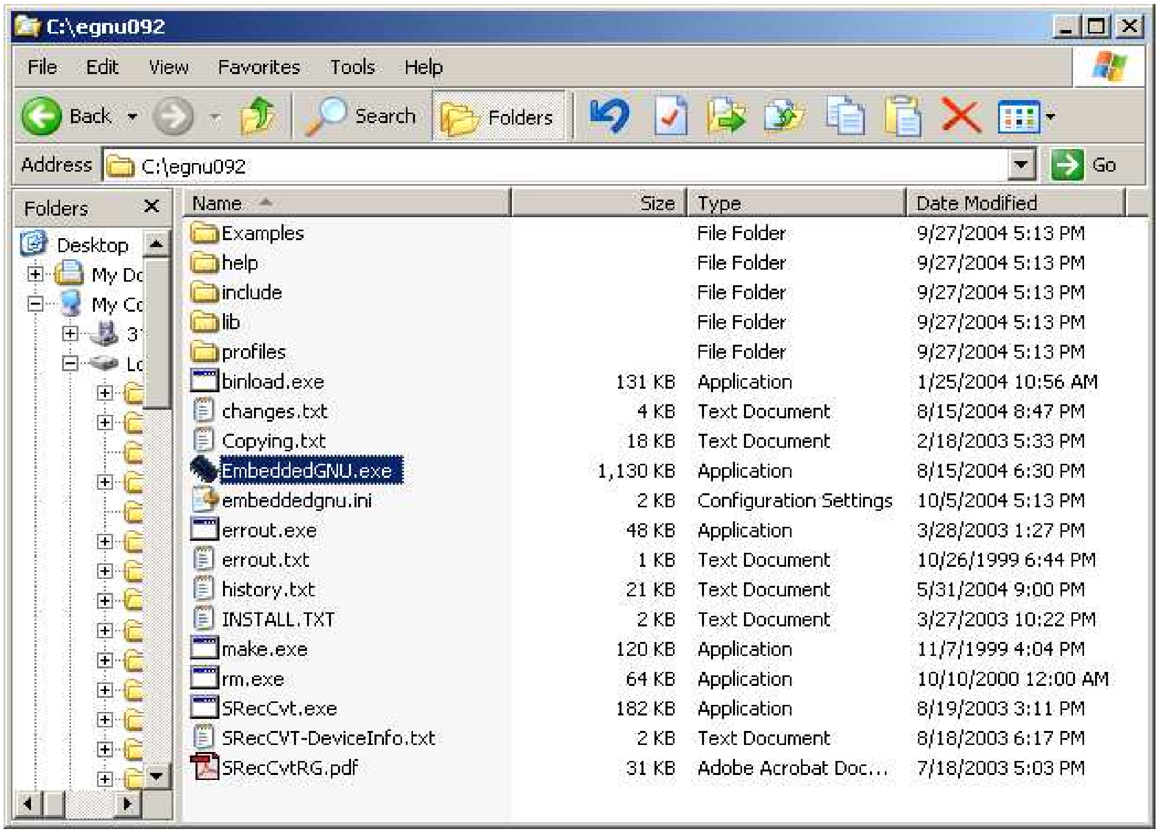
Download and install both EGNU and GCC. Links are:

http://www.geocities.com/englere_geo/
http://www.ericengler.com/EmbeddedGNU.aspx
http://www.gnu-m68hc11.org/

EGNU IDE download link
http://www.ericengler.com/downloads/egnu092.zip

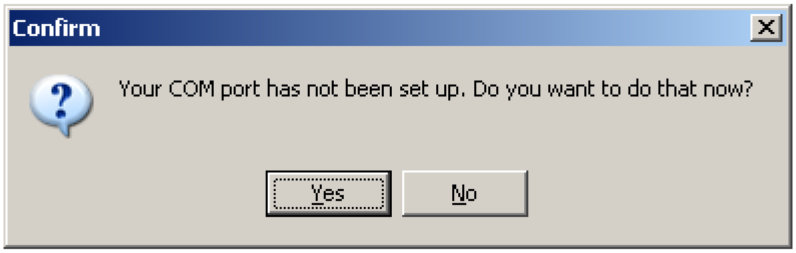
GCC download for Win98, XP and 2K
http://stephane.carrez.free.fr/m68hc11_pkg_zip.php
http://stephane.carrez.free.fr/EXE/gnu-68hc1x-2.92.exe

Locate the ***EmbeddedGNU.exe*** where it is installed too and double click on the icon.



Name	Size	Type	Date Modified
Examples		File Folder	9/27/2004 5:13 PM
help		File Folder	9/27/2004 5:13 PM
include		File Folder	9/27/2004 5:13 PM
lib		File Folder	9/27/2004 5:13 PM
profiles		File Folder	9/27/2004 5:13 PM
binload.exe	131 KB	Application	1/25/2004 10:56 AM
changes.txt	4 KB	Text Document	8/15/2004 8:47 PM
Copying.txt	18 KB	Text Document	2/18/2003 5:33 PM
EmbeddedGNU.exe	1,130 KB	Application	8/15/2004 6:30 PM
embeddedgnu.ini	2 KB	Configuration Settings	10/5/2004 5:13 PM
errout.exe	48 KB	Application	3/28/2003 1:27 PM
errout.txt	1 KB	Text Document	10/26/1999 6:44 PM
history.txt	21 KB	Text Document	5/31/2004 9:00 PM
INSTALL.TXT	2 KB	Text Document	3/27/2003 10:22 PM
make.exe	120 KB	Application	11/7/1999 4:04 PM
rm.exe	64 KB	Application	10/10/2000 12:00 AM
SRecCvt.exe	182 KB	Application	8/19/2003 3:11 PM
SRecCVT-DeviceInfo.txt	2 KB	Text Document	8/18/2003 6:17 PM
SRecCvtRG.pdf	31 KB	Adobe Acrobat Doc...	7/18/2003 5:03 PM

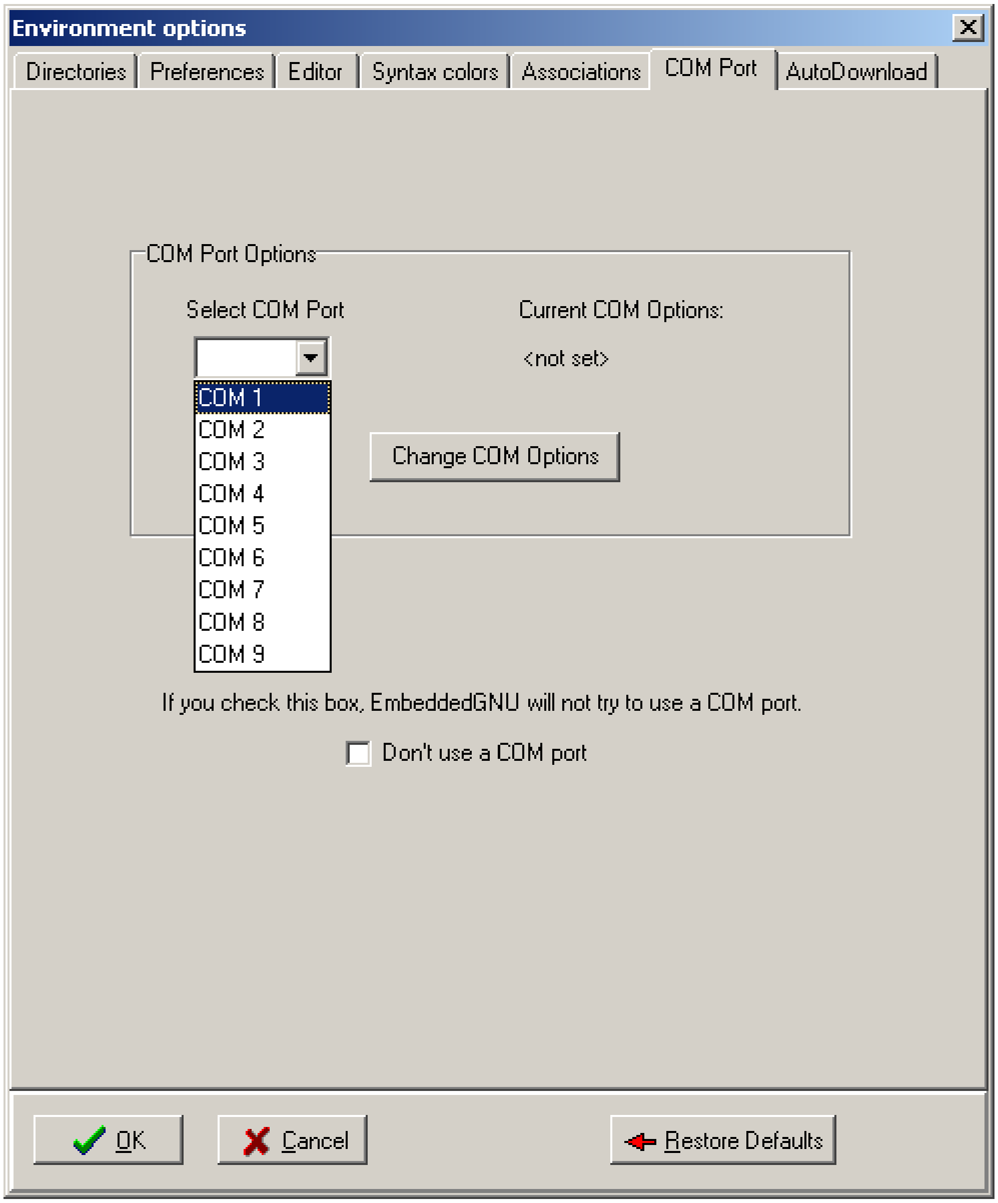
It will prompt for setting up the COM. Select the Yes or No button. It is NOT necessary to setup the COM but if one wants too then the following sequence is how to. Please note that if COM is enabled it will interfere with uBUG12 if there is only one COM port in the PC. If there are 2 serial ports in the PC then this problem will not exist.



Confirm

Your COM port has not been set up. Do you want to do that now?

If yes is selected then choose a COM port by clicking on the pull down arrow. The environment setup can be change by the Change COM options button.



Environment options

COM Port Options

Select COM Port

Current COM Options:

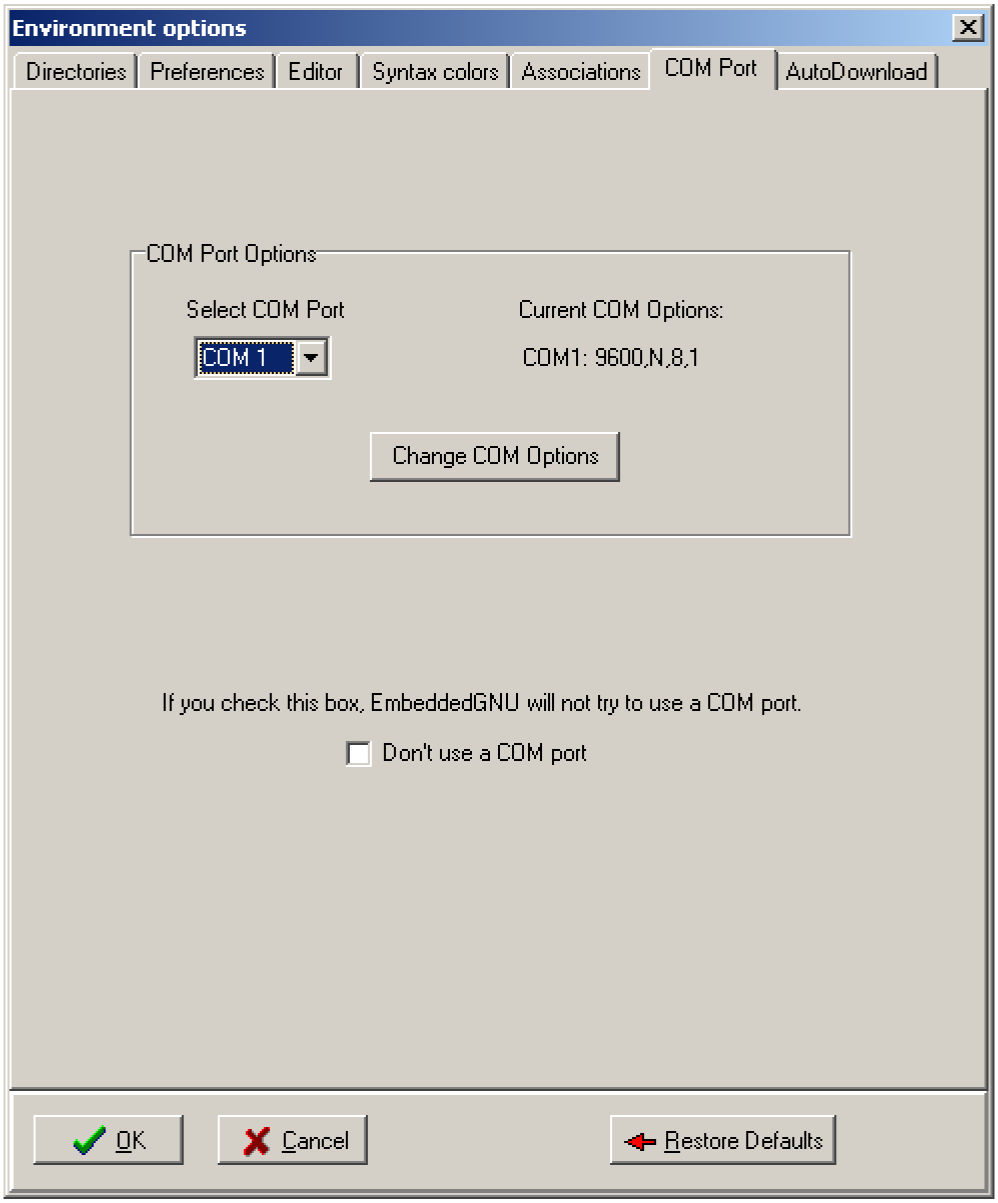
<not set>

COM 1
COM 2
COM 3
COM 4
COM 5
COM 6
COM 7
COM 8
COM 9

If you check this box, EmbeddedGNU will not try to use a COM port.

□ Don't use a COM port

Here COM 1 is selected then press OK button.



COM Port Options

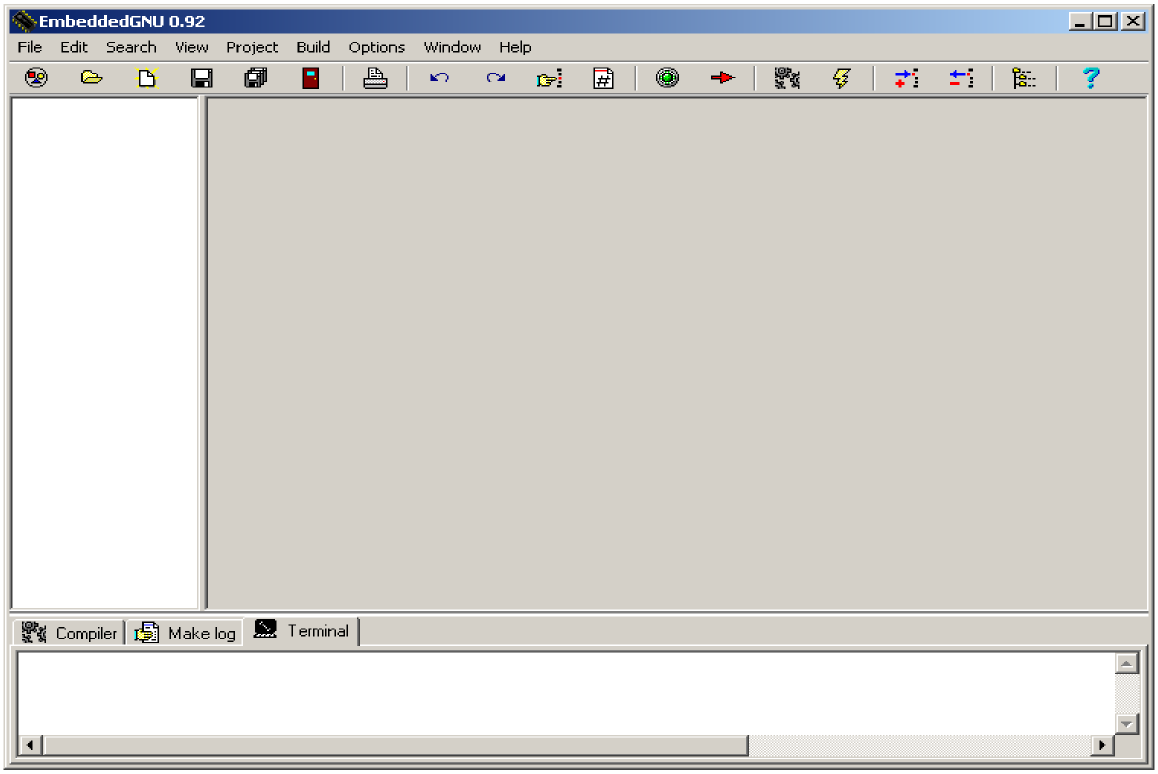
Select COM Port: COM 1

Current COM Options:
COM1: 9600,N,8,1

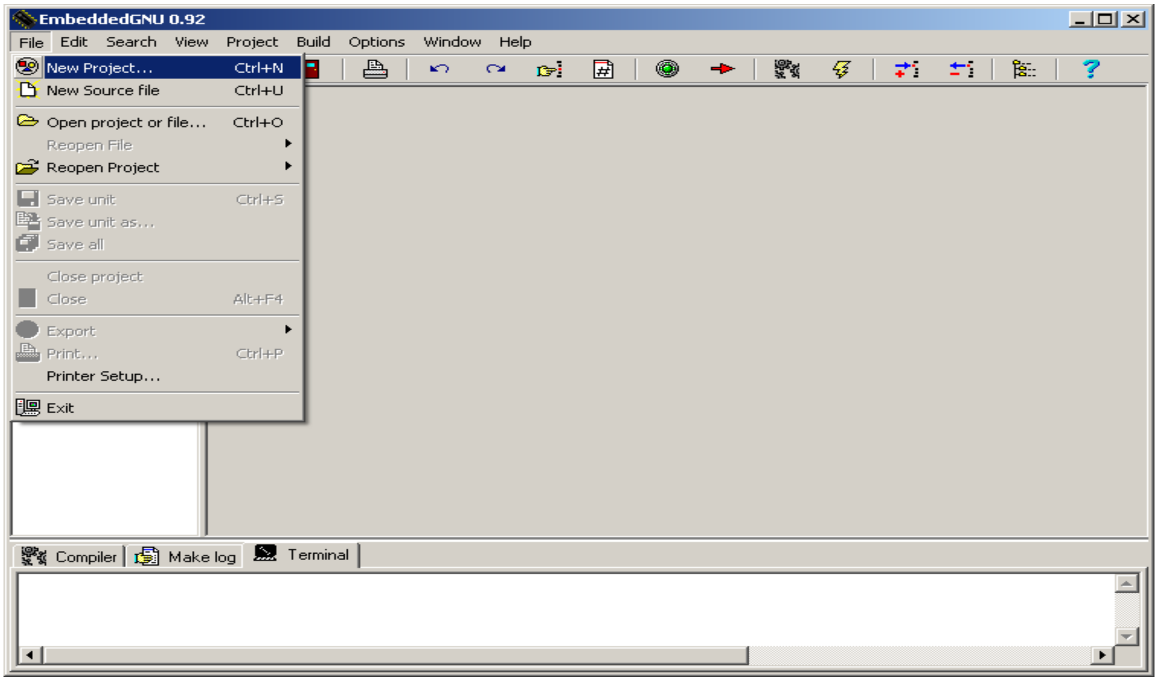
If you check this box, EmbeddedGNU will not try to use a COM port.

□ Don't use a COM port

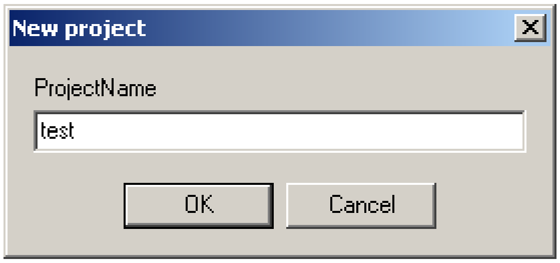
After the setup note the right window pane is greyed out and left window is blank. As usual with IDEs one is left with a BIG question what to do next. Let us then create a Project.



To create a new Project select File - New Project.

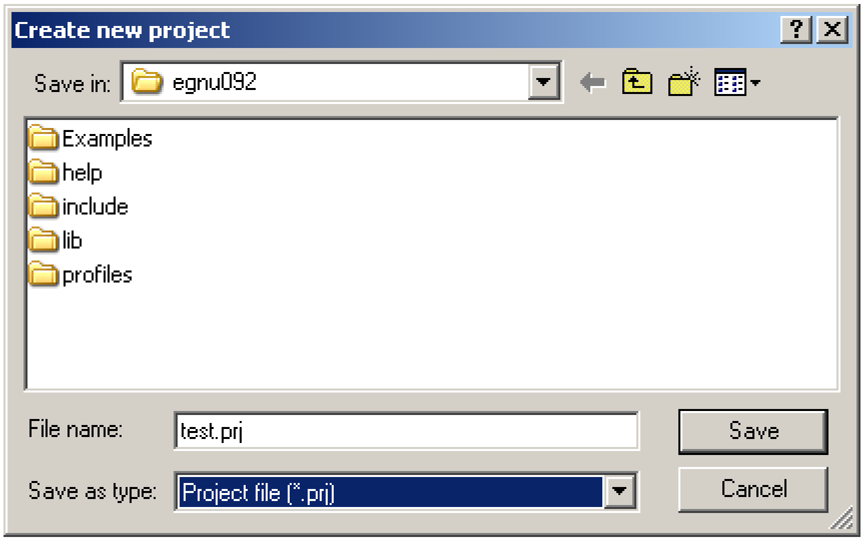


Give a name that makes sense. Here it is called test then press OK



ProjectName: test

Showing where to save the new project. User should decide whether to create a new folder or save it to an existing folder. Press Save button to Save project.



Save in: egnu092

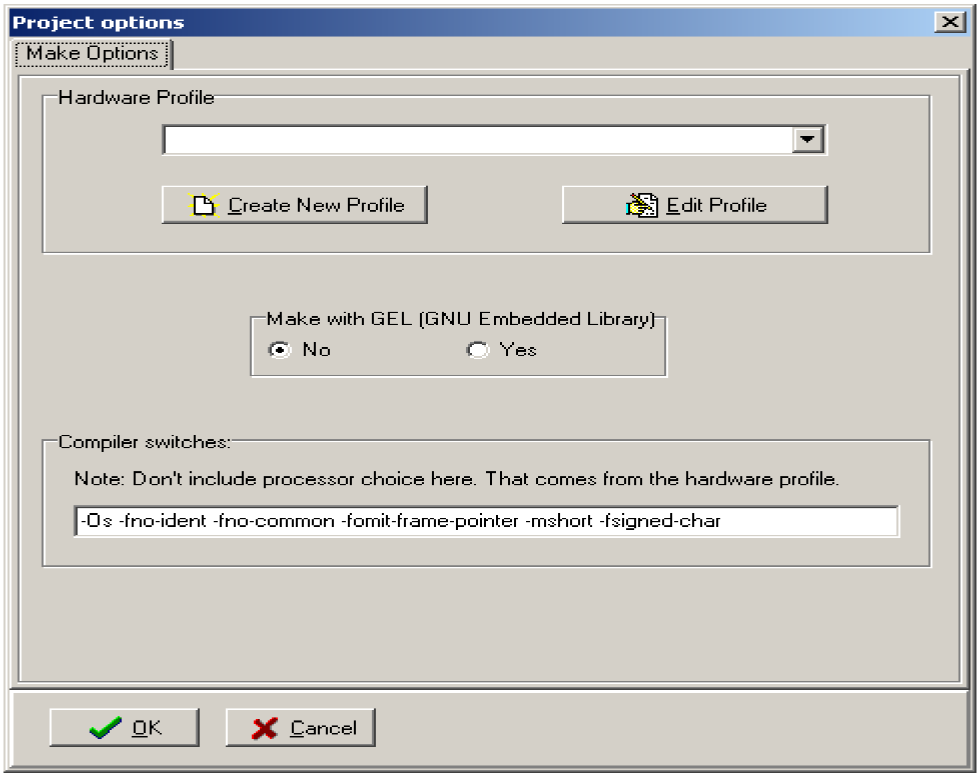
File name: test.prj

Save as type: Project file (*.prj)

The setup is the Project options. Here one can create a new profile or edit an existing profile.

What is profile? Profile is the type of board and MCU resources that one is working with. If a profile does not exist for one particular board/MCU model then one can create one. One can also edit an existing profile.

The Project Options is VERY important to the setup.



Hardware Profile:

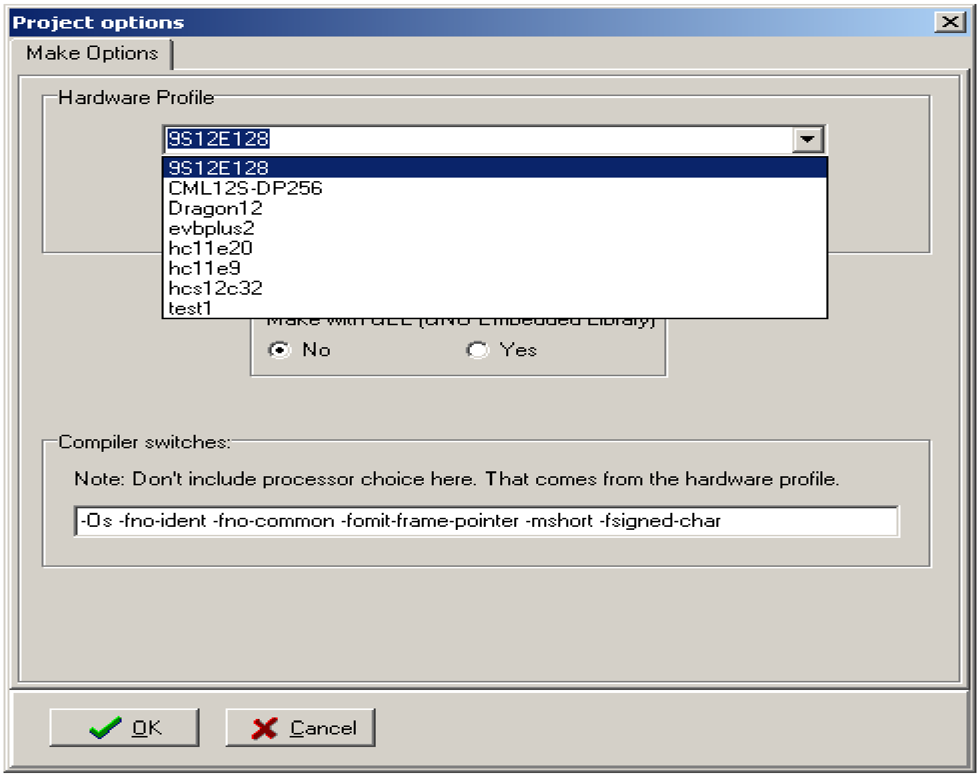
Make with GEL (GNU Embedded Library): ● No ○ Yes

Compiler switches:

Note: Don't include processor choice here. That comes from the hardware profile.

```
-Os -fno-ident -fno-common -fomit-frame-pointer -mshort -fsigned-char
```

Choosing an existing profile.



Hardware Profile: 9S12E128

- 9S12E128
- CML12S-DP256
- Dragon12
- evbplus2
- hc11e20
- hc11e9
- hcs12c32
- test1

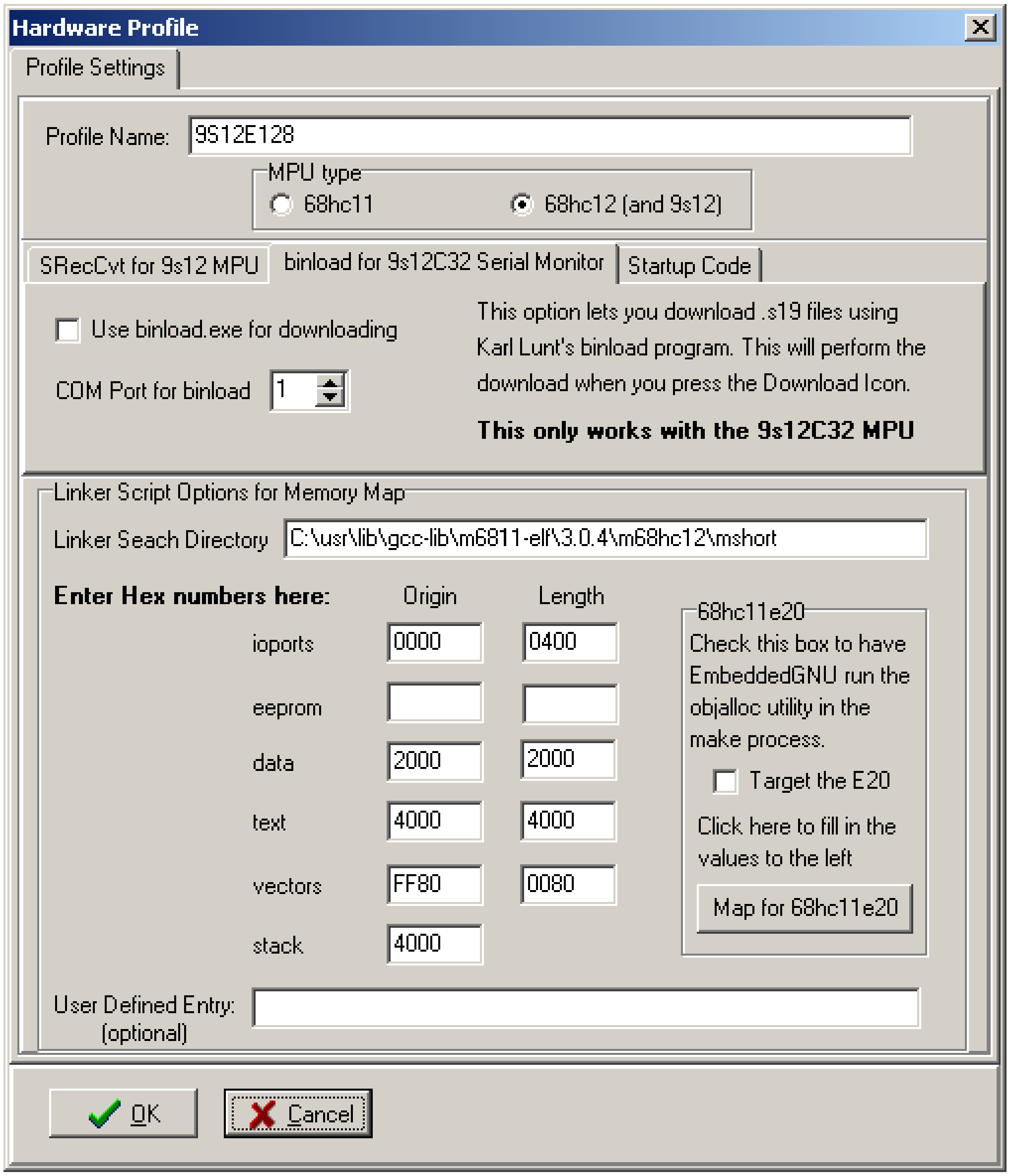
Make with GEL (GNU Embedded Library): ● No ○ Yes

Compiler switches:

Note: Don't include processor choice here. That comes from the hardware profile.

```
-Os -fno-ident -fno-common -fomit-frame-pointer -mshort -fsigned-char
```

In this example we will be using the 9S12E128 with Technological Arts Adapt9S12E128. By selecting the Edit profile button one can see the hardware profile as shown.



Hardware Profile

Profile Settings

Profile Name: 9S12E128

MPU type: ○ 68hc11 ● 68hc12 (and 9s12)

□ Use binload.exe for downloading

COM Port for binload: 1

This option lets you download .s19 files using Karl Lunt's binload program. This will perform the download when you press the Download Icon.

This only works with the 9s12C32 MPU

Linker Script Options for Memory Map

Linker Seach Directory: C:\usr\lib\gcc-lib\m6811-elf\3.0.4\m68hc12\mshort

Enter Hex numbers here:

	Origin	Length
ioports	0000	0400
eeprom		
data	2000	2000
text	4000	4000
vectors	FF80	0080
stack	4000	

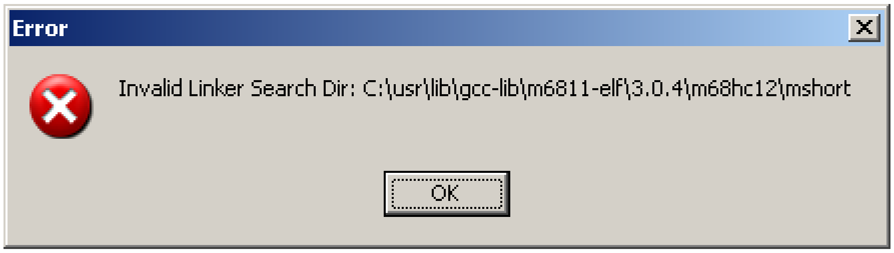
68hc11e20: Check this box to have EmbeddedGNU run the objalloc utility in the make process.

□ Target the E20

Click here to fill in the values to the left

User Defined Entry: (optional)

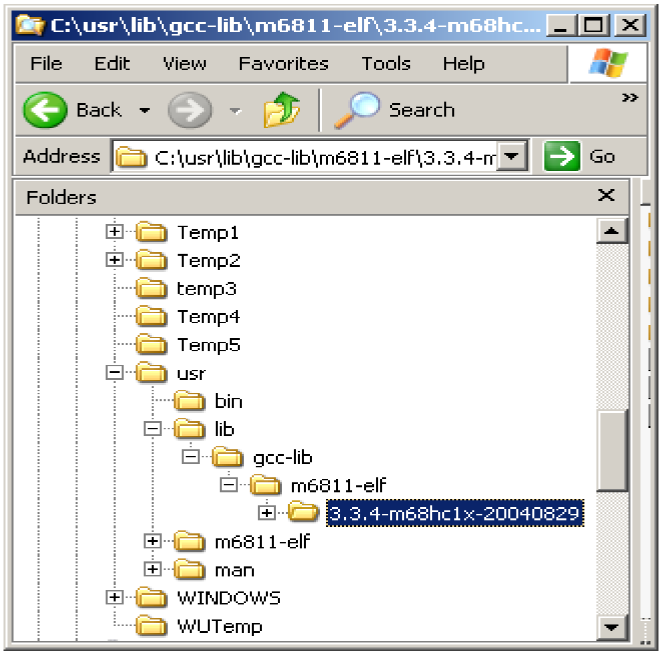
If the linker is not setup properly there is an error similar to this.



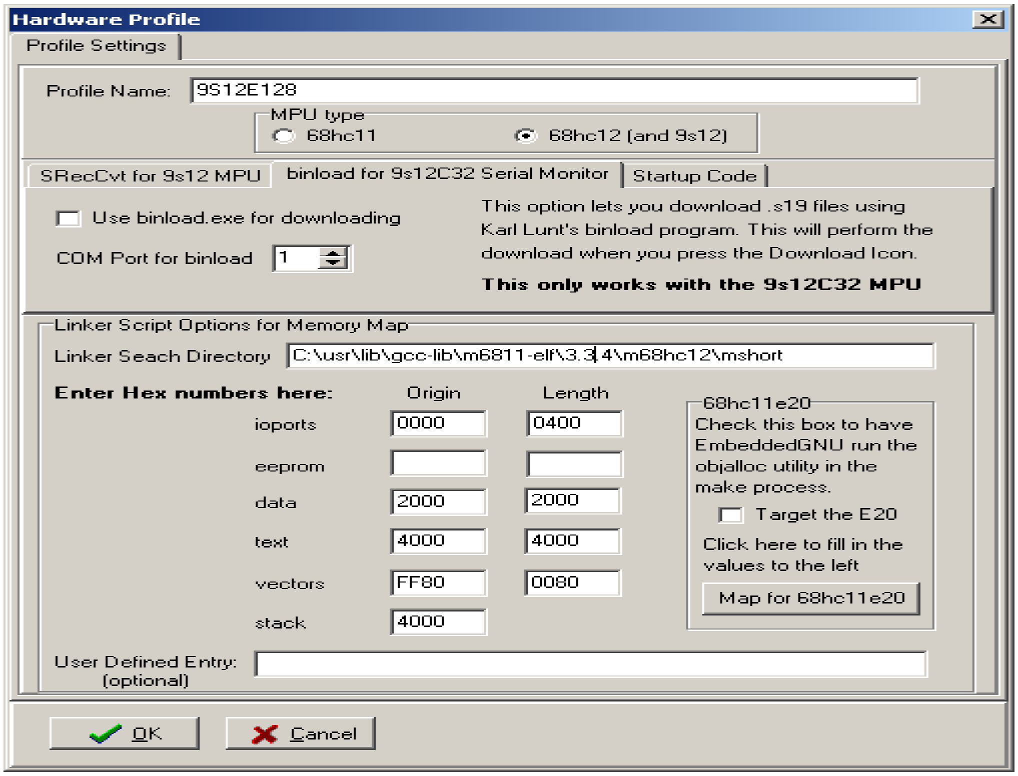
Error

Invalid Linker Search Dir: C:\usr\lib\gcc-lib\m6811-elf\3.0.4\m68hc12\mshort

Locate where the GCC is installed and note the version number.



Edit Linker Search Directory to C:\usr\lib\gcc-lib\m6811-elf\3.3.4-m68hc1x-20040829\mshort then press OK and the error will go away.



Hardware Profile

Profile Settings

Profile Name: 9S12E128

MPU type: ○ 68hc11 ● 68hc12 (and 9s12)

SRecCvt for 9s12 MPU | binload for 9s12C32 Serial Monitor | Startup Code

□ Use binload.exe for downloading

COM Port for binload: 1

This option lets you download .s19 files using Karl Lunt's binload program. This will perform the download when you press the Download Icon.

This only works with the 9s12C32 MPU

Linker Script Options for Memory Map

Linker Seach Directory: C:\usr\lib\gcc-lib\m6811-elf\3.3.4\m68hc12\mshort

Enter Hex numbers here:	Origin	Length
ioports	0000	0400
eeprom		
data	2000	2000
text	4000	4000
vectors	FF80	0080
stack	4000	

68hc11e20

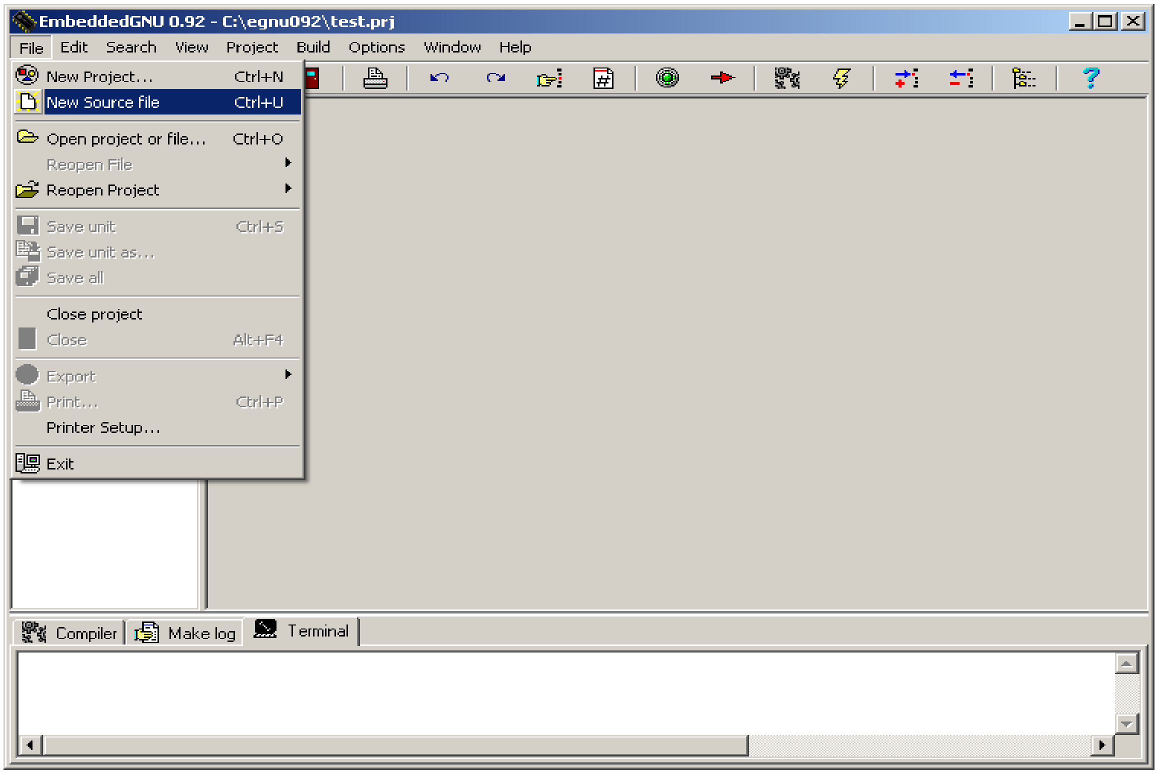
Check this box to have EmbeddedGNU run the objalloc utility in the make process.

□ Target the E20

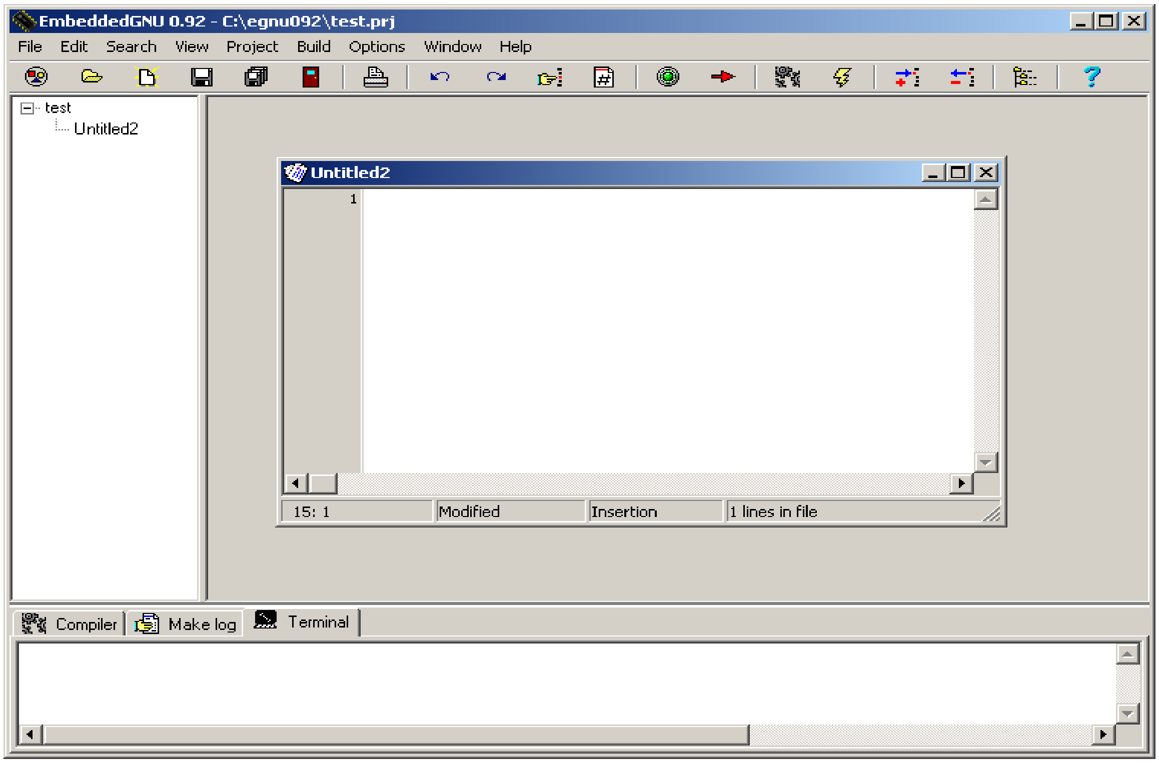
Click here to fill in the values to the left

User Defined Entry: (optional)

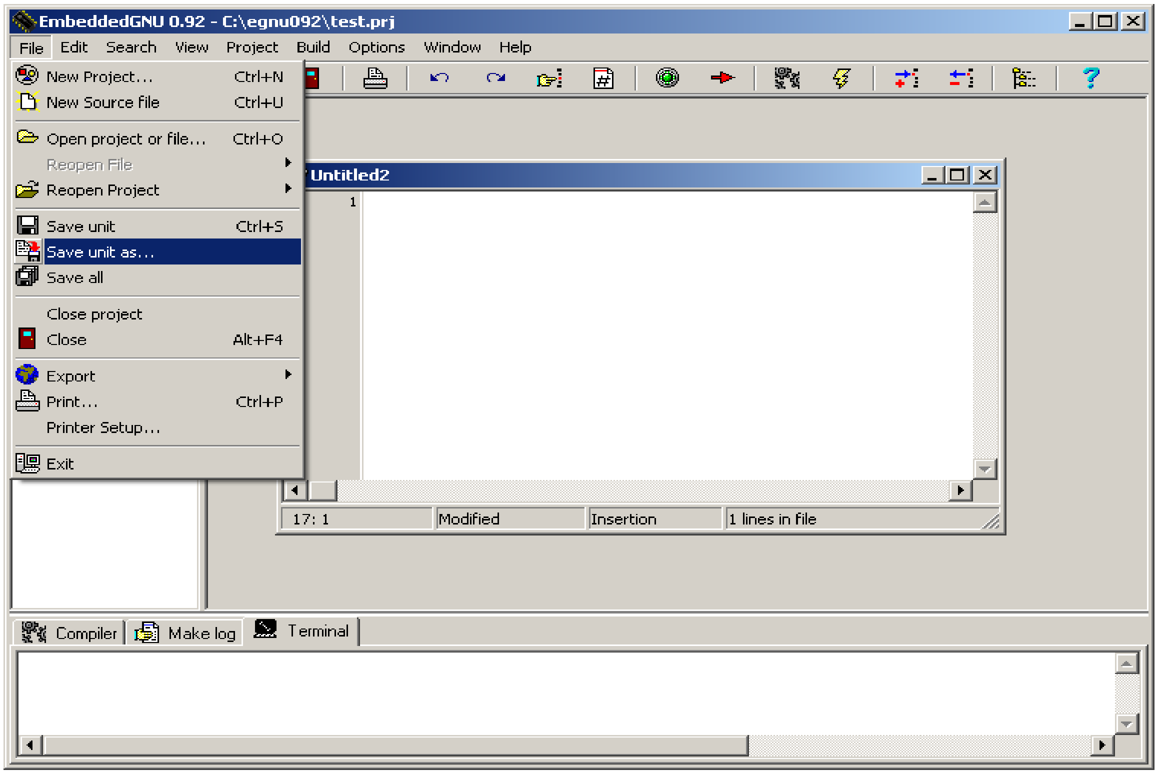
Add a new file. Select File – New Source file



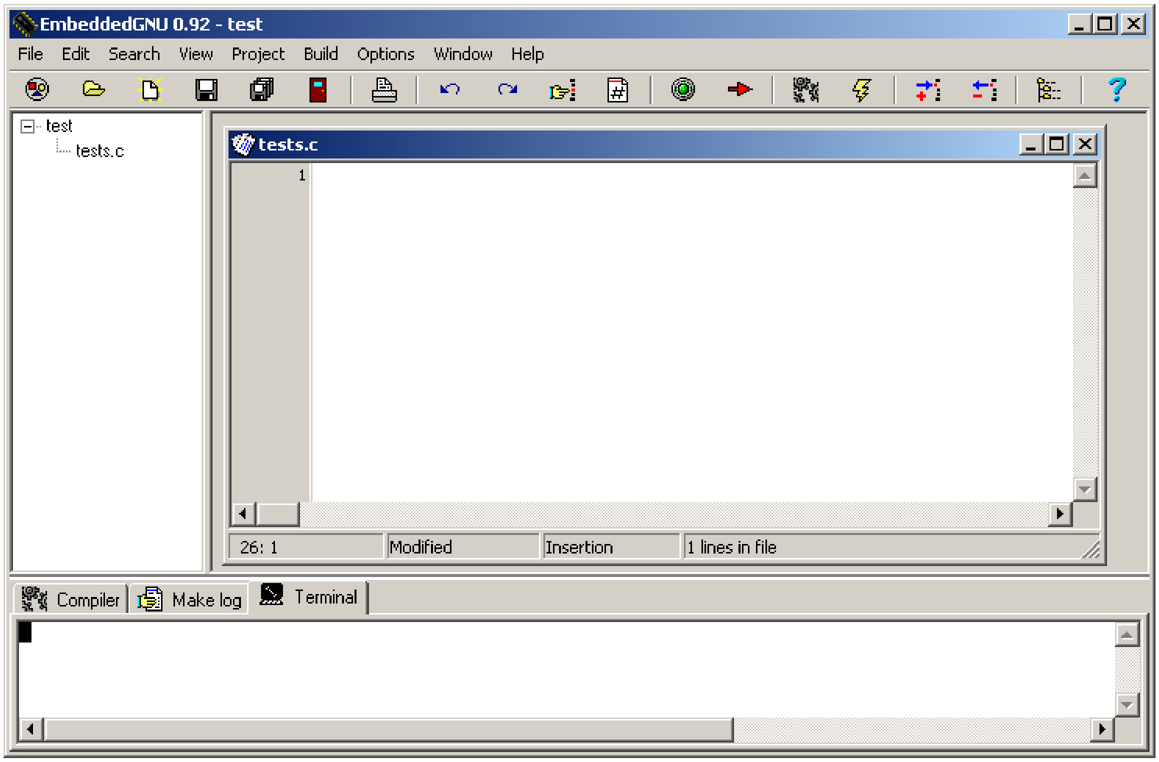
A new file is created.


EmbeddedGNU 0.92 - C:\egnu092\test.prj
File Edit Search View Project Build Options Window Help
test
Untitled2
Untitled2
1
15: 1
Modified
Insertion
1 lines in file
Compiler
Make log
Terminal

The file can be saved as ***tests.c.***

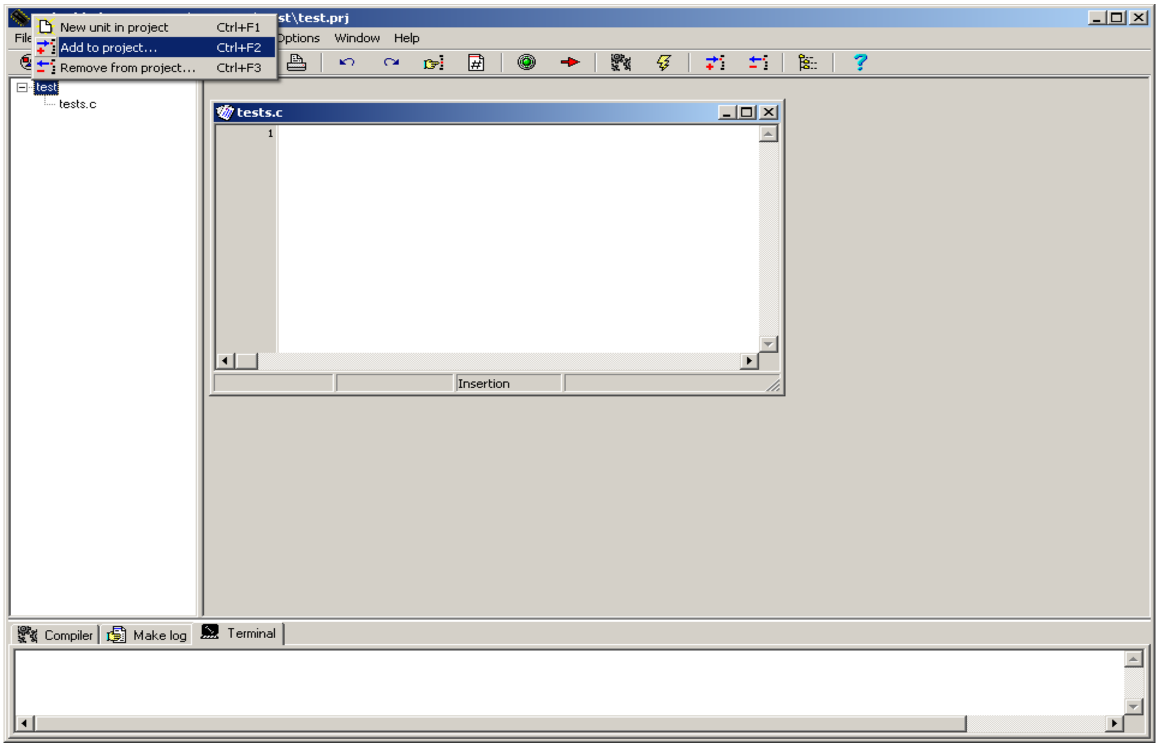


Note the file is renamed as shown.

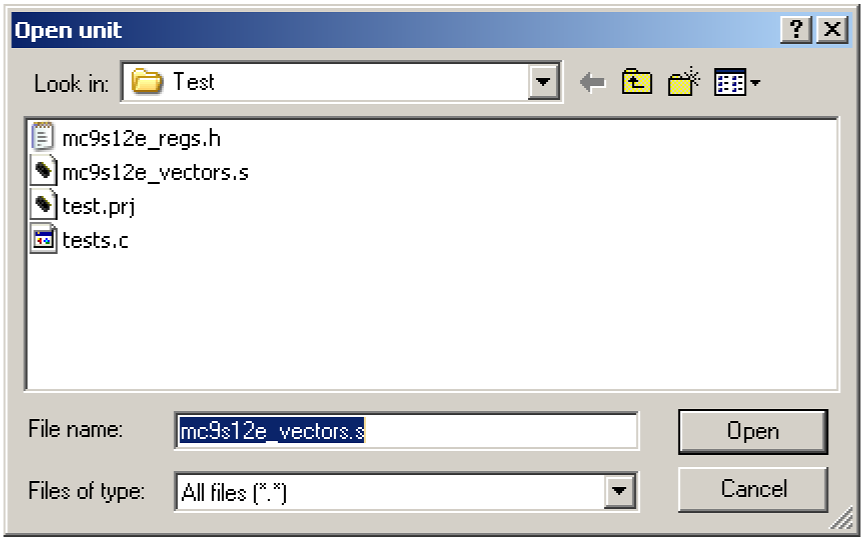


Because GCC and EGNU are freeware, the Ports register definition needs to be created if it does not exist. This document assume that the Port definition does exist but if it does not, one should find out from the GCC/EGNU forum where one could find a particular MCU port definition.

Let us add the 9S12E128 vector.s file. In the left window pane Right click on test project and click on Add to project the file mc9s12e_vectors.s.



A window explorer will open to help and locate the file. In this example it is in Test subdirectory.

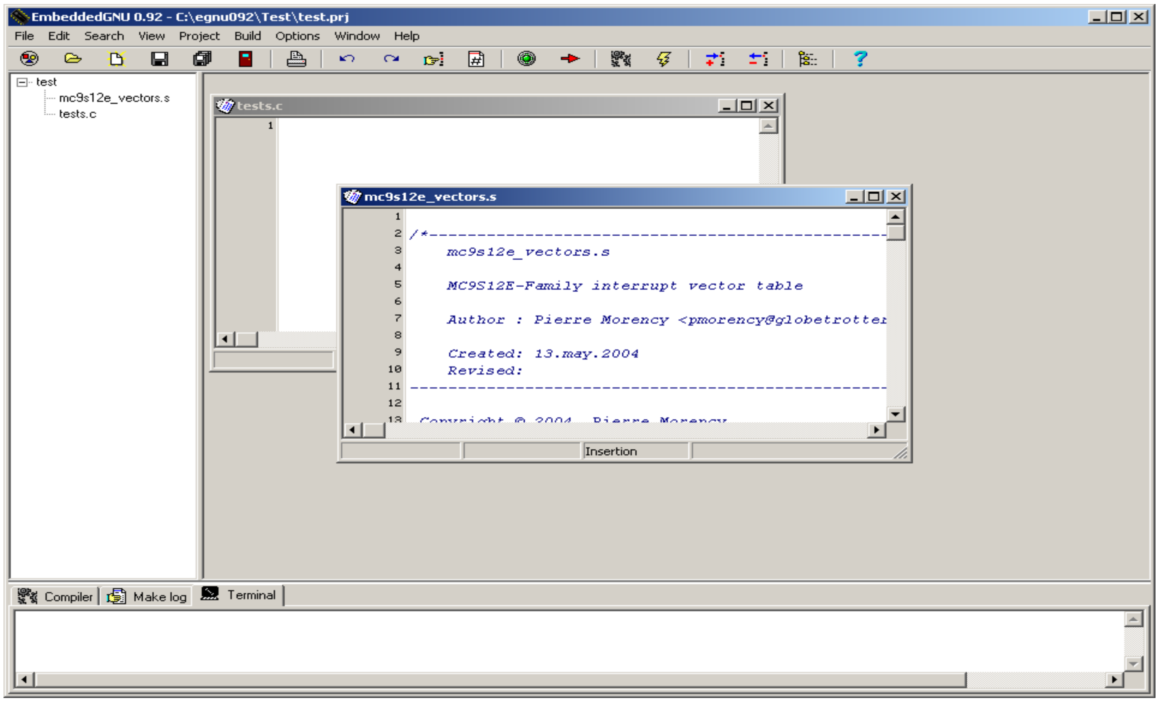


Look in: Test

File name: mc9s12e_vectors.s

Files of type: All files (*.*)

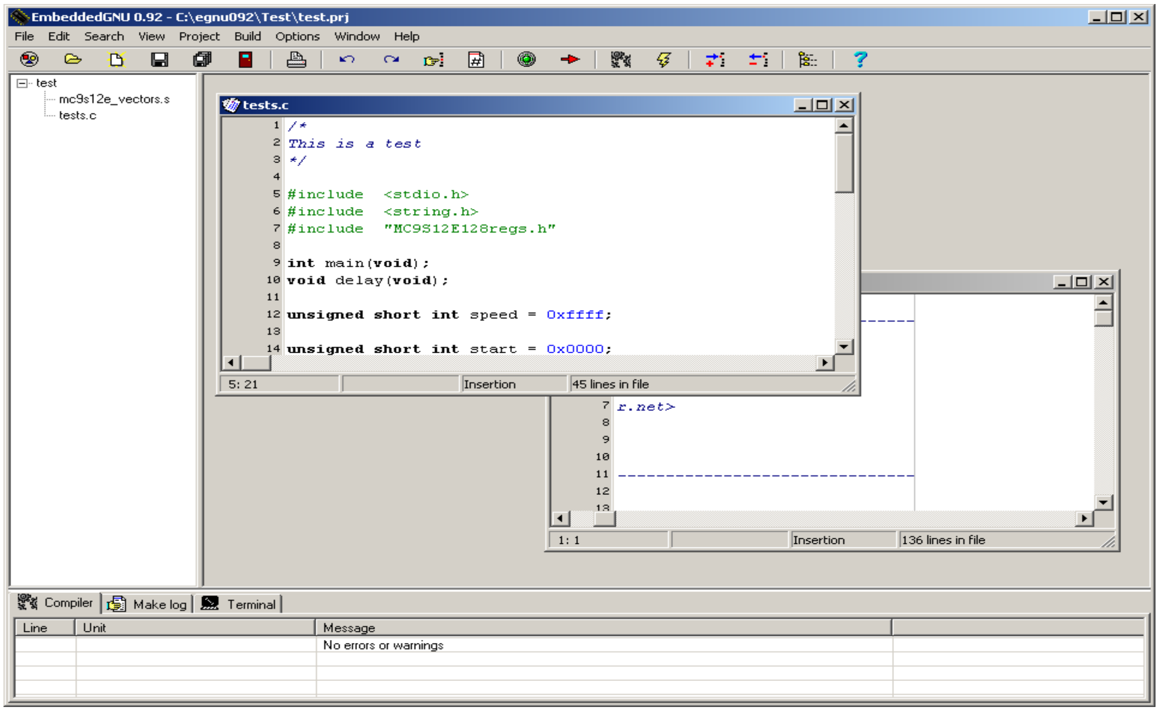
Here the GUI changed to show the file is added to project.


EmbeddedGNU 0.92 - C:\egnu092\Test\test.prj
File Edit Search View Project Build Options Window Help
test
mc9s12e_vectors.s
tests.c
tests.c
mc9s12e_vectors.s
Insertion
Compiler
Make log
Terminal

```
/*-------------------------------------------------
    mc9s12e_vectors.s

    MC9S12E-Family interrupt vector table

    Author : Pierre Morency <pmorency@globetrotter

    Created: 13.may.2004
    Revised:
--------------------------------------------------

Copyright © 2004  Pierre Morency
```

Add the headers to the test.c file.

#include <stdio.h>
#include <string.h>
#include "MC9S12E128regs.h"



```
/*
This is a test
*/

#include  <stdio.h>
#include  <string.h>
#include  "MC9S12E128regs.h"

int main(void);
void delay(void);

unsigned short int speed = 0xffff;

unsigned short int start = 0x0000;
```

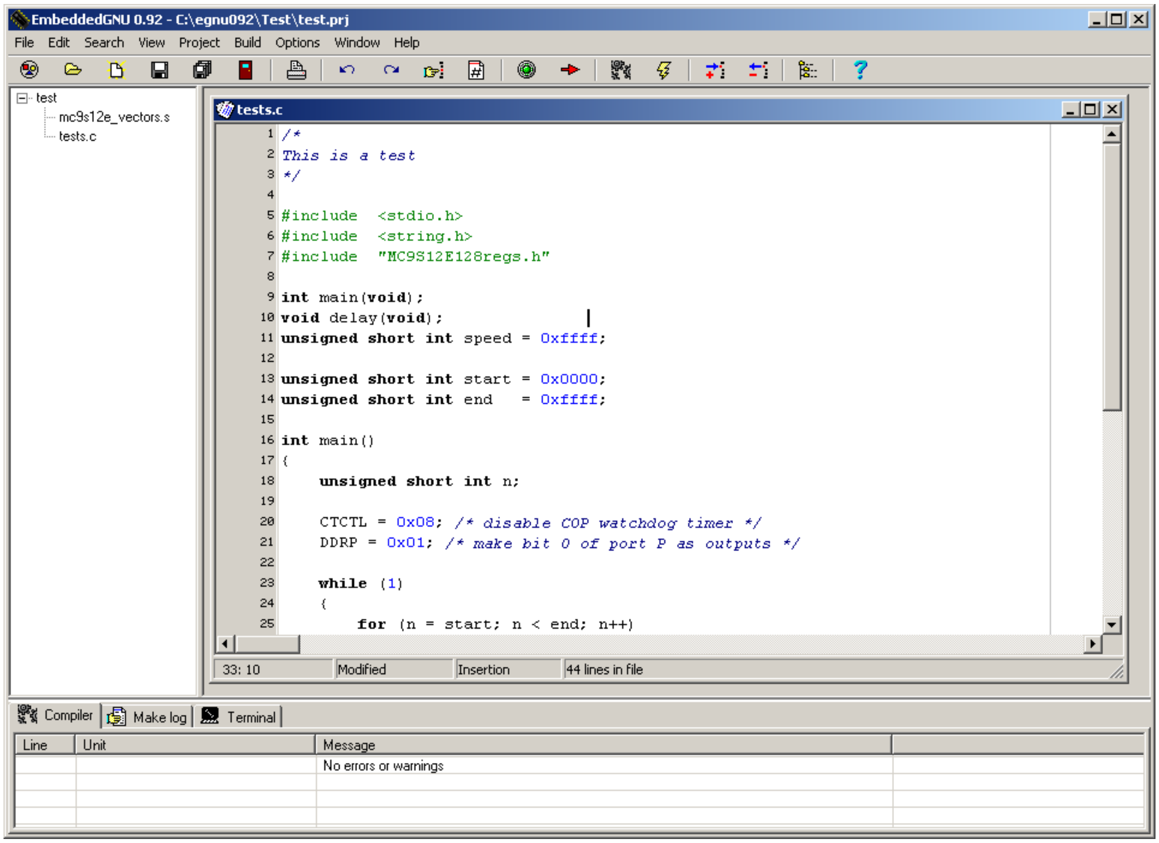
```
r.net>

------------------------------
```

Line	Unit	Message	
		No errors or warnings	

Add the rest of the code to test.c file.

```
int main(void);
void delay(void);

unsigned short int speed = 0xffff;

unsigned short int start = 0x0000;
unsigned short int end   = 0xffff;

int main()
{
    unsigned short int n;

    CTCTL = 0x08; /* disable COP watchdog timer */
    DDRP = 0x01; /* make bit 0 of port P as outputs */

    while (1)
    {
        for (n = start; n < end; n++)
        {
          PORTP = n & 0x01;  /* send the 1 low order bits to port P */
          delay();
          delay();
        }
    }
    return  0; /* not used */
}

void delay()
{
  unsigned short int i;

  for (i = 0; i < speed; i++)
  {
     /* nothing */
  }
}
```

After all the codes are typed we will now compile.

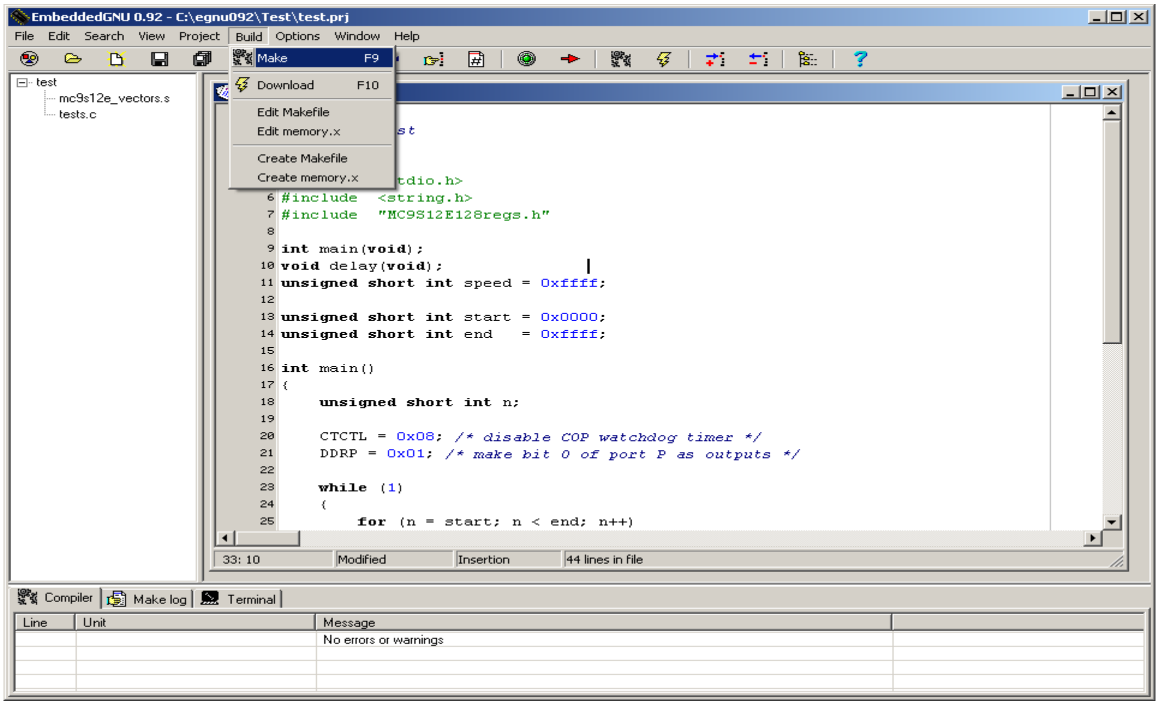


```
/*
This is a test
*/

#include  <stdio.h>
#include  <string.h>
#include  "MC9S12E128regs.h"

int main(void);
void delay(void);
unsigned short int speed = 0xffff;

unsigned short int start = 0x0000;
unsigned short int end   = 0xffff;

int main()
{
    unsigned short int n;

    CTCTL = 0x08; /* disable COP watchdog timer */
    DDRP = 0x01; /* make bit 0 of port P as outputs */

    while (1)
    {
        for (n = start; n < end; n++)
```

33: 10 | Modified | Insertion | 44 lines in file

Line	Unit	Message	
		No errors or warnings	

Click Build – Make

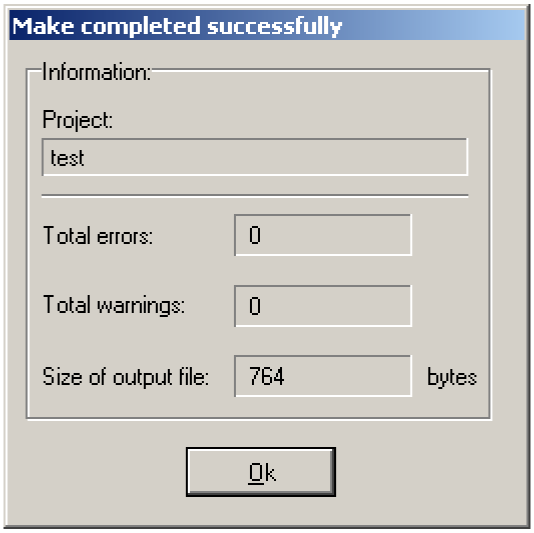


```
st

tdio.h>
#include  <string.h>
#include  "MC9S12E128regs.h"

int main(void);
void delay(void);
unsigned short int speed = 0xffff;

unsigned short int start = 0x0000;
unsigned short int end   = 0xffff;

int main()
{
    unsigned short int n;

    CTCTL = 0x08; /* disable COP watchdog timer */
    DDRP = 0x01; /* make bit 0 of port P as outputs */

    while (1)
    {
        for (n = start; n < end; n++)
```

33: 10 | Modified | Insertion | 44 lines in file

Line	Unit	Message	
		No errors or warnings	

It will try to build the files. It there are no errors then it will show the build was successful.



Make completed successfully

Information:

Project: test

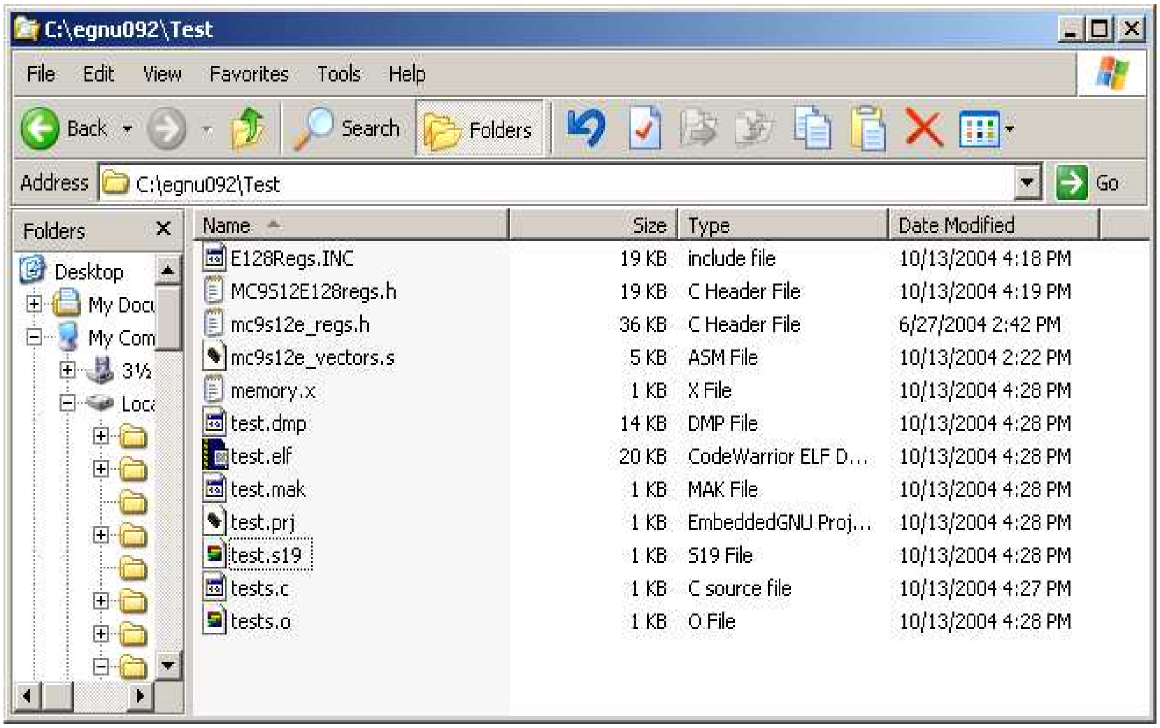
Total errors: 0

Total warnings: 0

Size of output file: 764 bytes

It is always a good idea to check the S-record. One should be aware what is an S-record and what it represent. See Appendix A for further explanation.

Here we can see that test.s19 was generated along with other files related after the build.



C:\egnu092\Test

Address C:\egnu092\Test

Name	Size	Type	Date Modified
E128Regs.INC	19 KB	include file	10/13/2004 4:18 PM
MC9S12E128regs.h	19 KB	C Header File	10/13/2004 4:19 PM
mc9s12e_regs.h	36 KB	C Header File	6/27/2004 2:42 PM
mc9s12e_vectors.s	5 KB	ASM File	10/13/2004 2:22 PM
memory.x	1 KB	X File	10/13/2004 4:28 PM
test.dmp	14 KB	DMP File	10/13/2004 4:28 PM
test.elf	20 KB	CodeWarrior ELF D...	10/13/2004 4:28 PM
test.mak	1 KB	MAK File	10/13/2004 4:28 PM
test.prj	1 KB	EmbeddedGNU Proj...	10/13/2004 4:28 PM
test.s19	1 KB	S19 File	10/13/2004 4:28 PM
tests.c	1 KB	C source file	10/13/2004 4:27 PM
tests.o	1 KB	O File	10/13/2004 4:28 PM

WordPad is used to open the Test.s19. Below is the content of the S-record. See appendix A for explanation of the S-record lines.

```
S00B0000746573742E73313929
S1134000CF4000164074CE407ECD2000CC00062761
S113401007180A30700434F9CC00002708CE2006B3
S113402069300434FB16403116407820FB400640CA
S11340301834180B08003E180B01025A180180208E
S113404002EC80BC200424F4E681C4017B025807FE
S113405010070EEC80C300016C80BC200425E9200D
S1134060DBFE2000CC0000044508C3000134ACB1E1
S111407025F83D0B87B7023D10EF3E20FBA75D
S109407EFFFF0000FFFF3C
S113FF8040734073407340734073407340734073D5
S113FF9040734073407340734073407340734073C5
S113FFA040734073407340734073407340734073B5
S113FFB040734073407340734073407340734073A5
S113FFC04073407340734073407340734073407395
S113FFD04073407340734073407340734073407385
S113FFE04073407340734073407340734073407375
S113FFF040734073407340734073407340734000D8
S9034000BC
```

A few things to point out in the record below taken out of context.

S1 13 ***4000*** CF4000164074CE407ECD2000CC000627 61

S1 13 ***FFF0*** 4073407340734073407340734073 ***4000*** D8

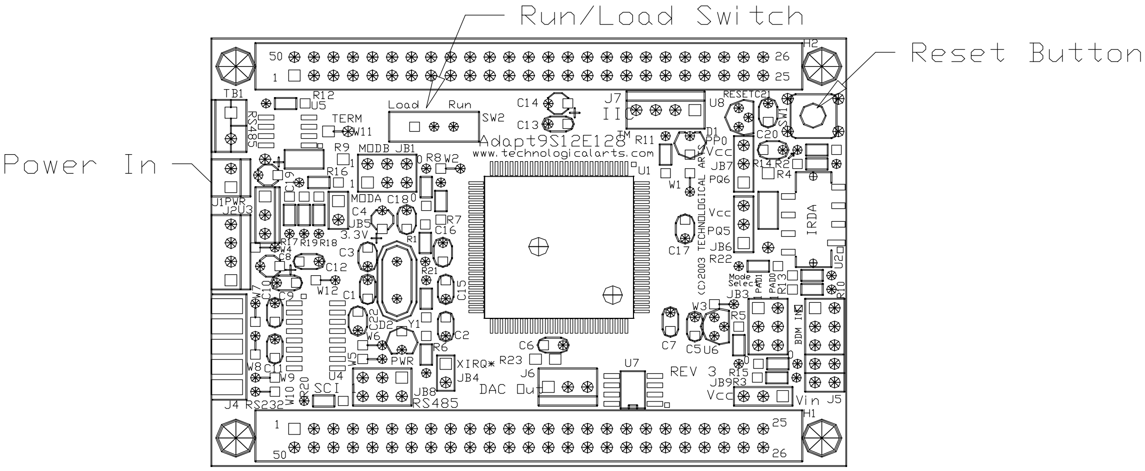
Note that the address at $FFFE contains $4000 and further note that the start of code is at $4000. The point being made is to verify and make sure that the code will start at where the vector address is pointing too.

Also note that the Serial Monitor resides at $F800 - $FFFF. Therefore uBUG12 will automatically re-locate the vector addresses at below $F800.

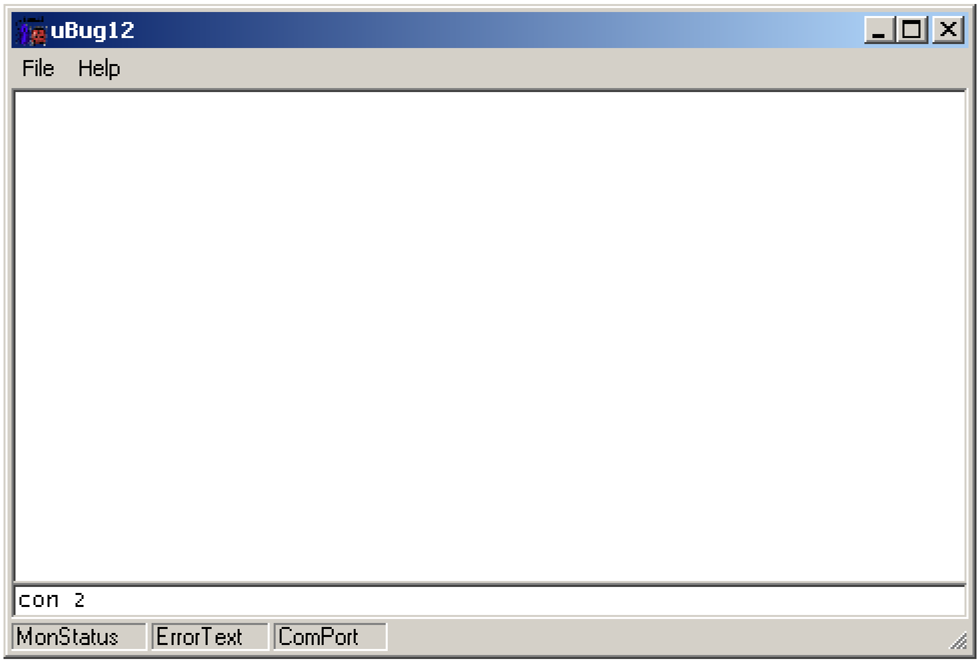
It is time now to program the S-record into the Adapt9S12E128 using uBUG12. Other method can be used to erase and program the MCU but in this case we will use uBUG12. This document assumes that the Serial Monitor is not erased. It further assumes that COM 1 is enabled and is used by EGNU leaving COM 2 for uBUG12.

If no COM is to be used then it is better to start a new Project with the COM disabled.

Switch the Run/Load switch to Load position and apply power to the board.



Double click on the uBUG12 icon to initiate GUI. Type ***con 2*** for COM 2 serial port.

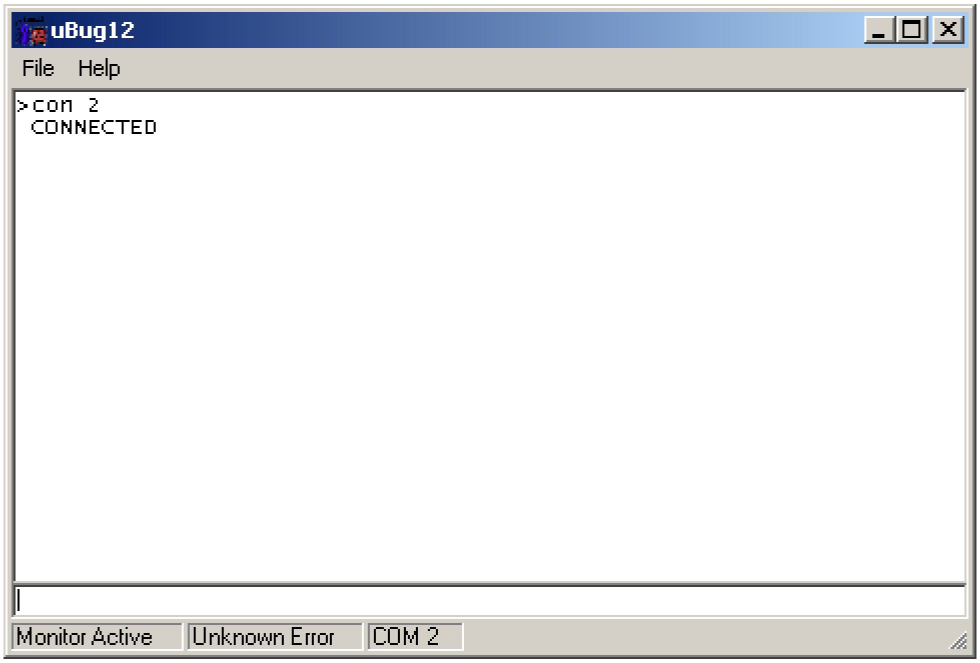


```
con 2
```

Connections establish between PC and Adapt9S12E128

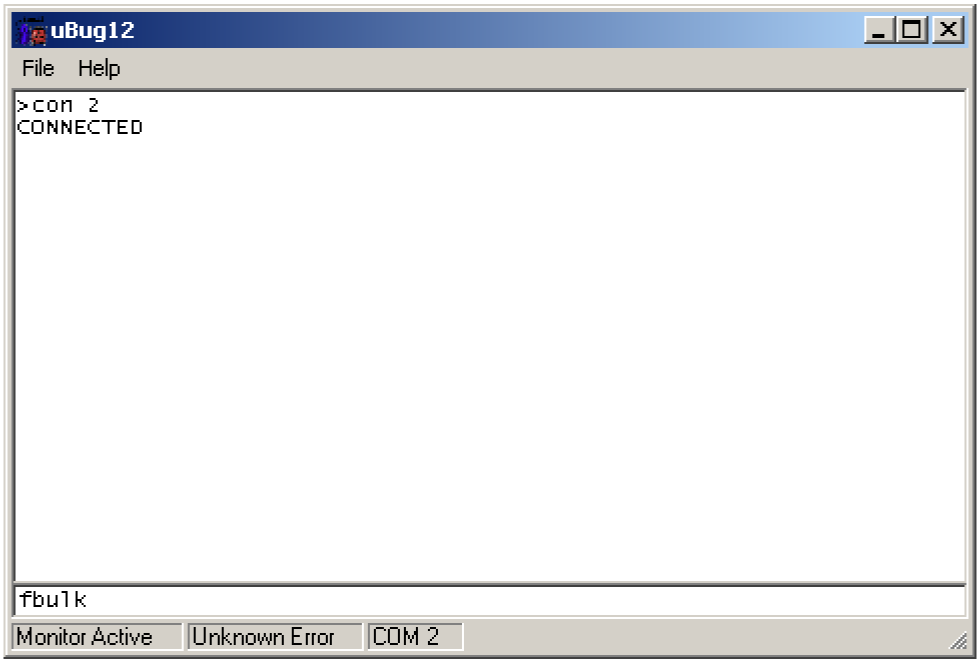
2 possible errors can occur:
Connection Error: Unable to open COM2 <- Another application is using the COM port

Connection Error: Read Error: Timeout error <- The MCU not currently in LOAD mode or the cable is disconnected from either PC or Docking Module. Lastly, the cable could be connected at the wrong COM port.



```
>con 2
 CONNECTED
```

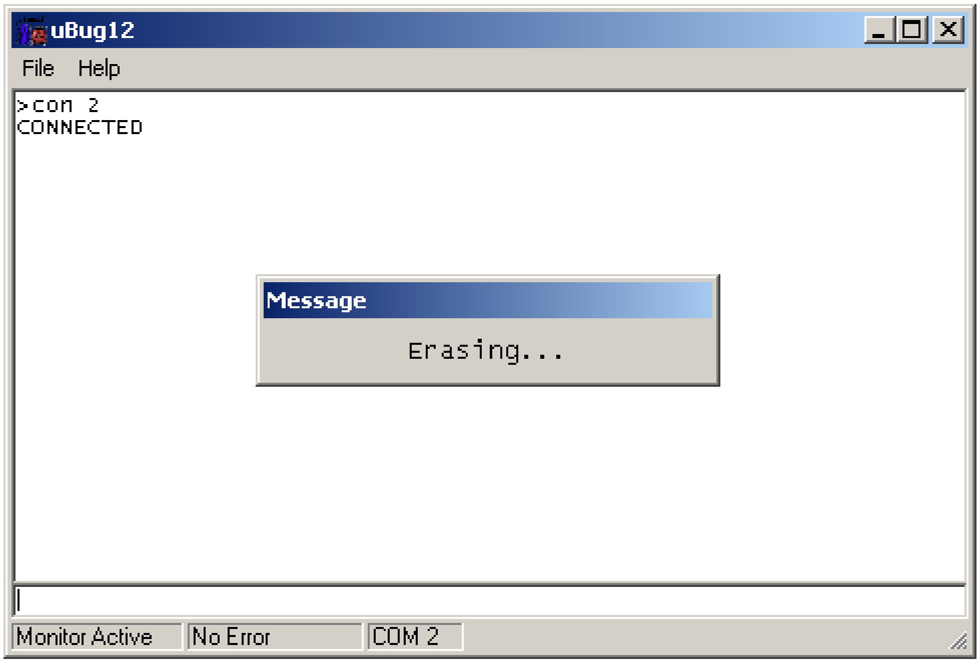
To erase flash type ***fbulk***



```
>con 2
CONNECTED
```

```
fbulk
```

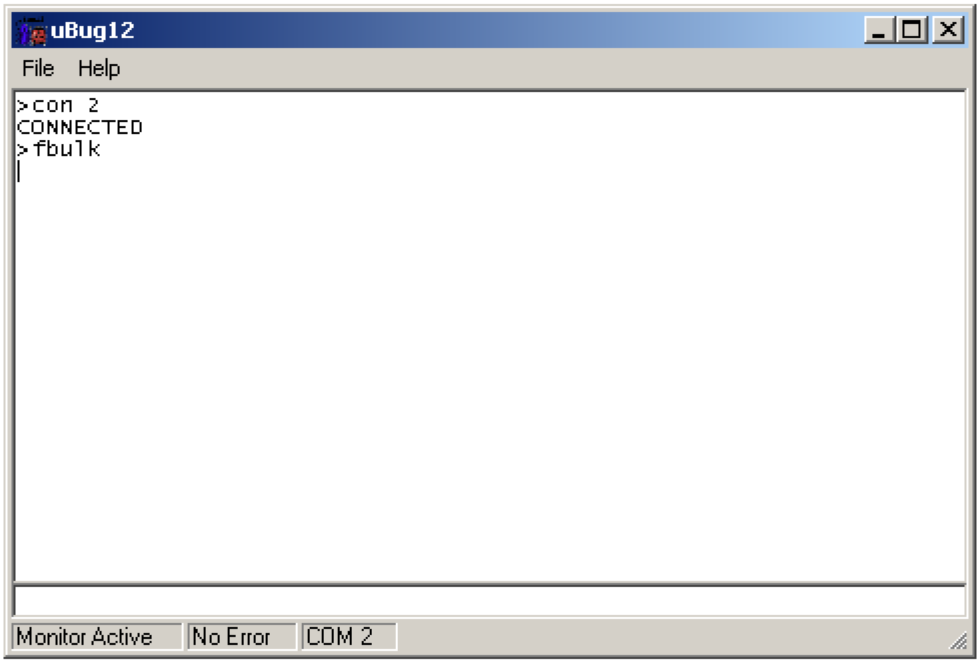
Erasing message



```
>con 2
CONNECTED
```

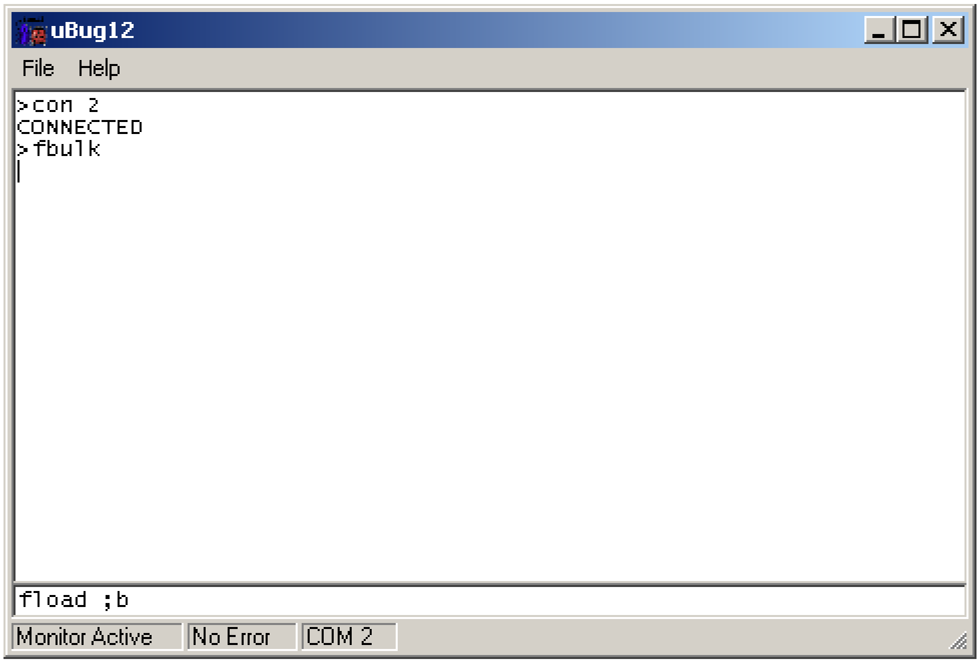
Erasing...

Erase successful



```
>con 2
CONNECTED
>fbulk
```

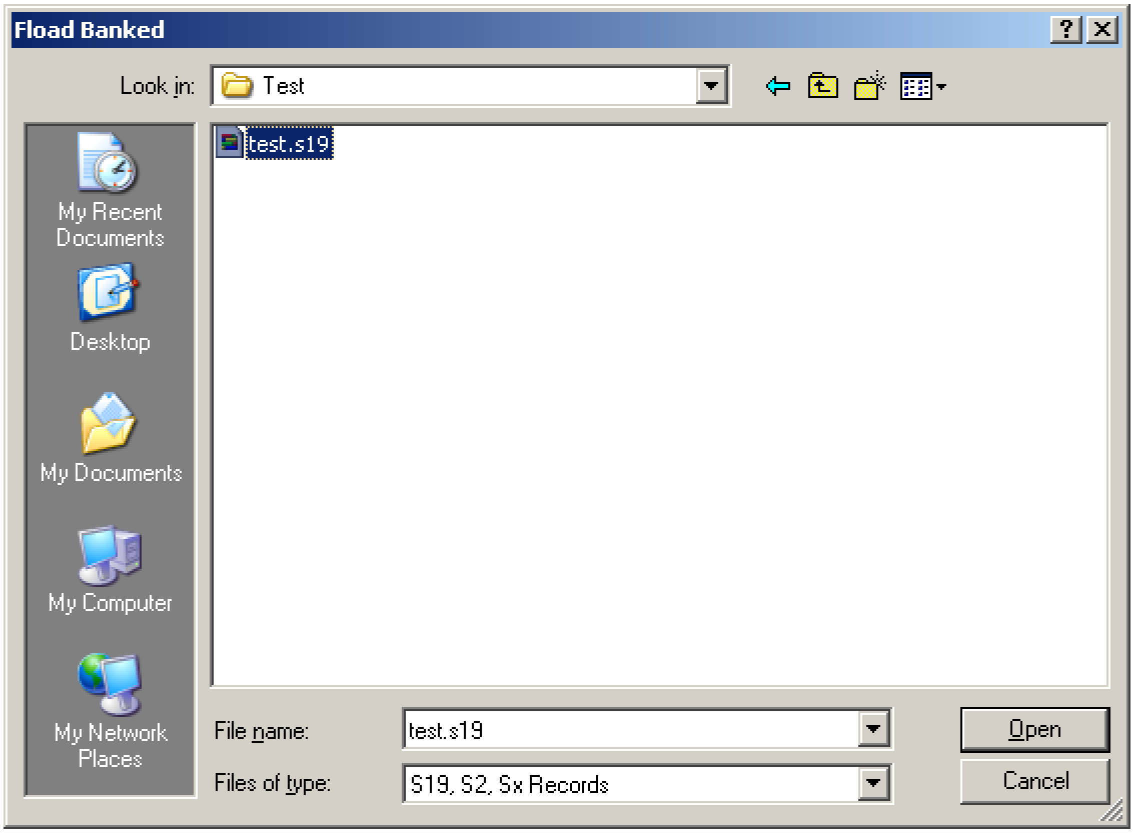
To program S-record. EGNU generates Linear S-record. This is considered to Banked record by uBUG12. The command is ***Fload ;b*** for banked and ***Fload*** for non Banked S-record.



```
>con 2
CONNECTED
>fbulk
```

```
fload ;b
```

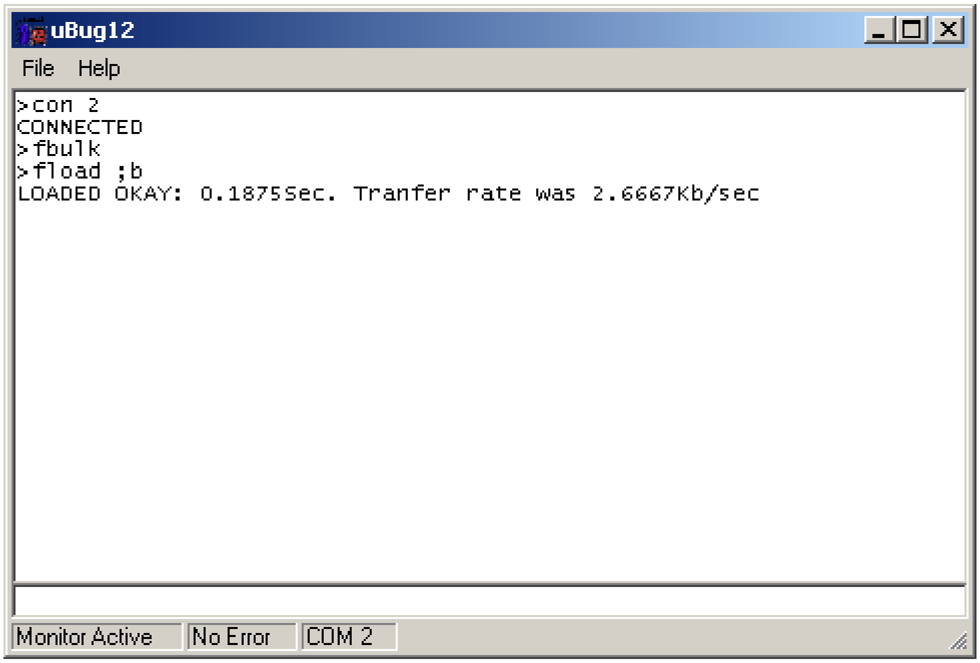
uBUG12 will open an Explorer window to help and locate the S-record.



File name: test.s19

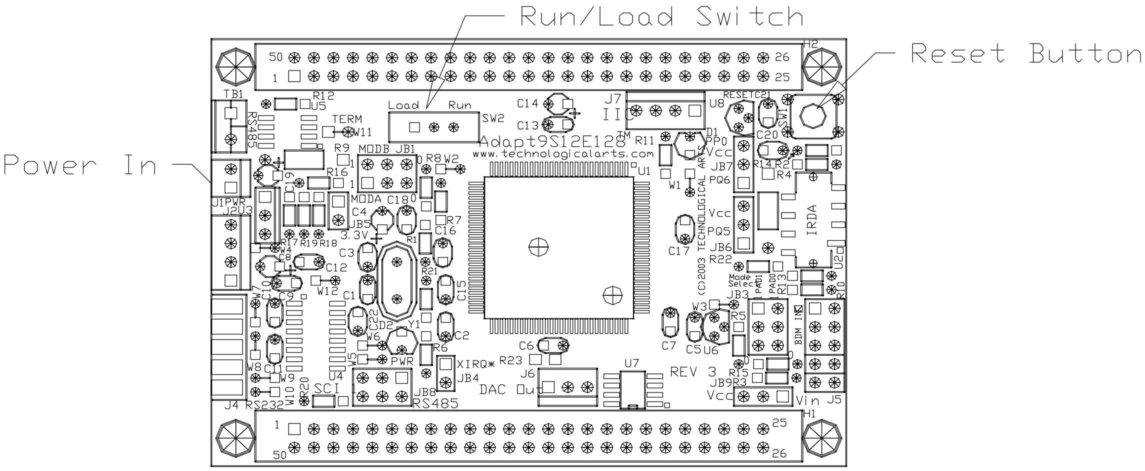
Files of type: S19, S2, Sx Records

Double click on the appropriate file to initiate download. Here the file is successfully loaded.



```
>con 2
CONNECTED
>fbulk
>fload ;b
LOADED OKAY: 0.1875Sec. Tranfer rate was 2.6667Kb/sec
```

Switch the Run/Load to Run, press the RESET button and the LED should immediately begins flash/blink.



This concludes the GCC/EGNU with uBUG12. As with all things the challenges are always to better understand how these tools are to be used. This document shows the process of using GCC/EGNU to Flashing the MCU using uBUG12.

Appendix A

Motorola S-records

NAME
srec - S-record file and record format

DESCRIPTION

An S-record file consists of a sequence of specially formatted ASCII character strings. An S-record will be less than or equal to 78 bytes in length.

The order of S-records within a file is of no significance and no particular order may be assumed.

The general format of an S-record follows:

```
+-------------------//------------------//-----------------------+
| type | count | address  |            data           | checksum |
+-------------------//------------------//-----------------------+
```

type -- A char[2] field. These characters describe the type of record (S0, S1, S2, S3, S5, S7, S8, or S9).

count -- A char[2] field. These characters when paired and interpreted as a hexadecimal value, display the count of remaining character pairs in the record.

address -- A char[4,6, or 8] field. These characters grouped and interpreted as a hexadecimal value, display the address at which the data field is to be loaded into memory. The length of the field depends on the number of bytes necessary to hold the address. A 2-byte address uses 4 characters, a 3-byte address uses 6 characters, and a 4-byte address uses 8 characters.

data -- A char [0-64] field. These characters when paired and interpreted as hexadecimal values represent the memory loadable data or descriptive information.

checksum -- A char[2] field. These characters when paired and interpreted as a hexadecimal value display the least significant byte of the ones complement of the sum of the byte values represented by the pairs of characters making up the count, the address, and the data fields.

Each record is terminated with a line feed. If any additional or different record terminator(s) or delay characters are needed during transmission to the target system it is the responsibility of the transmitting program to provide them.

S0 Record. The type of record is 'S0' (0x5330). The address field is unused and will be filled with zeros (0x0000). The header information within the data field is divided into the following subfields.

mname is char[20] and is the module name.
ver is char[2] and is the version number.
rev is char[2] and is the revision number.
description is char[0-36] and is a text comment.
Each of the subfields is composed of ASCII bytes whose associated characters, when paired, represent one byte hexadecimal values in the case of the version and revision numbers, or represent the hexadecimal values of the ASCII characters comprising the module name and description.

S1 Record. The type of record field is 'S1' (0x5331). The address field is intrepreted as a 2-byte address. The data field is composed of memory loadable data.

S2 Record. The type of record field is 'S2' (0x5332). The address field is intrepreted as a 3-byte address. The data field is composed of memory loadable data.

S3 Record. The type of record field is 'S3' (0x5333). The address field is intrepreted as a 4-byte address. The data field is composed of memory loadable data.

S5 Record. The type of record field is 'S5' (0x5335). The address field is intrepreted as a 2-byte value and contains the count of S1, S2, and S3 records previously transmitted. There is no data field.

S7 Record. The type of record field is 'S7' (0x5337). The address field contains the starting execution address and is intrepreted as 4-byte address. There is no data field.

S8 Record. The type of record field is 'S8' (0x5338). The address field contains the starting execution address and is intrepreted as 3-byte address. There is no data field.

S9 Record. The type of record field is 'S9' (0x5339). The address field contains the starting execution address and is intrepreted as 2-byte address. There is no data field.

EXAMPLE

Shown below is a typical S-record format file.

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
```

```
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S5030004F8
S9030000FC
```

The file consists of one S0 record, four S1 records, one S5 record and an S9 record.

The S0 record is comprised as follows:

- S0 S-record type S0, indicating it is a header record.
- 06 Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.
- 00 00 Four character 2-byte address field, zeroes in this example.
- 48 44 52 ASCII H, D, and R - "HDR".
- 1B The checksum.

The first S1 record is comprised as follows:

- S1 S-record type S1, indicating it is a data record to be loaded at a 2-byte address.
- 13 Hexadecimal 13 (decimal 19), indicating that nineteen character pairs, representing a 2 byte address, 16 bytes of binary data, and a 1 byte checksum, follow.
- 00 00 Four character 2-byte address field; hexidecimal address 0x0000, where the data which follows is to be loaded.
- 28 5F 24 5F 22 12 22 6A 00 04 24 29 00 08 23 7C Sixteen character pairs representing the actual binary data.
- 2A The checksum.

The second and third S1 records each contain 0x13 (19) character pairs and are ended with checksums of 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S5 record is comprised as follows:

- S5 S-record type S5, indicating it is a count record indicating the number of S1 records
- 03 Hexadecimal 03 (decimal 3), indicating that three character pairs follow.
- 00 04 Hexadecimal 0004 (decimal 4), indicating that there are four data records previous to this record.
- F8 The checksum.

The S9 record is comprised as follows:

- S9 S-record type S9, indicating it is a termination record.
- 03 Hexadecimal 03 (decimal 3), indicating that three character pairs follow.
- 00 00 The address field, hexadecimal 0 (decimal 0) indicating the starting execution address.
- FC The checksum.

Instructor Notes

- There isn't any evidence that Motorola ever has made use of the header information within the data field of the S0 record, as described above. This must have been used by some third party vendors.
- This is the only place that a 78-byte limit on total record length or 64-byte limit on data length is documented. These values shouldn't be trusted for the general case.
- The count field can have values in the range of 0x3 (2 bytes of address + 1 byte checksum = 3, a not very useful record) to 0xff; this is the count of remaining character pairs, including checksum.
- If you write code to convert S-Records, you should always assume that a record can be as long as 514 (decimal) characters in length (255 * 2 = 510, plus 4 characters for the type and count fields), plus any terminating character(s). That is, in establishing an input buffer in C, you would declare it to be an array of 515 chars, thus leaving room for the terminating null character.