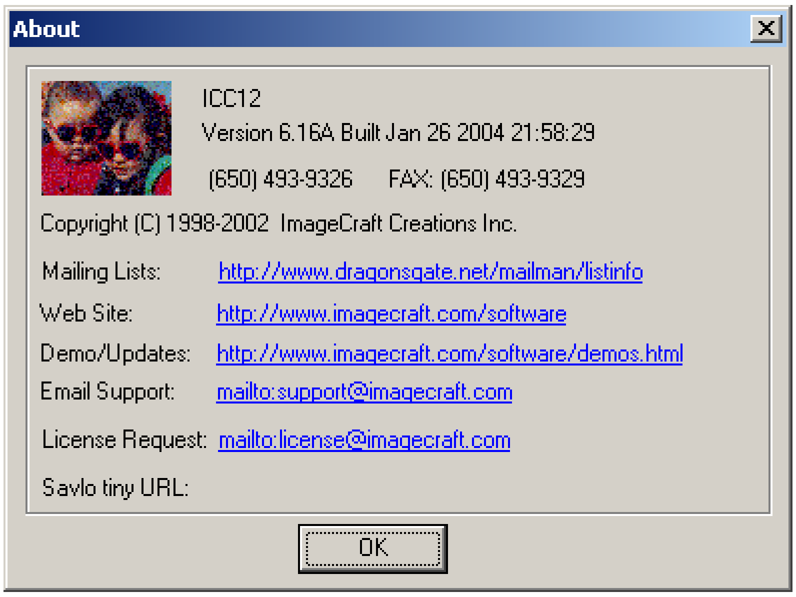
# How to use ICC12 with Adapt9S12DP256 and FLASH Loader

This document will show and demonstrate the use of ImageCraft ICC12 Latest ***Version 6*** with Technological Arts' Adapt9S12DP256 module.

The FLASH Loader (BootEVB-16.S19) is a DBUG12's loader written by Gordon Doughman. It can be found inside the ***DB12S12FW.zip*** file. The file can be downloaded from www.freescale.com website. It will be used here to erase and program FLASH after the compilation of a test program. Other method can be used to also erase and program the FLASH but in this example it will be the FLASH Loader.

This document assumes that the user is familiar with C and so will not teach how to program C here.

## ImageCraft Links:



ICC12
Version 6.16A Built Jan 26 2004 21:58:29

(650) 493-9326 FAX: (650) 493-9329

Copyright (C) 1998-2002 ImageCraft Creations Inc.

| | |
|---|---|
| Mailing Lists: | http://www.dragonsgate.net/mailman/listinfo |
| Web Site: | http://www.imagecraft.com/software |
| Demo/Updates: | http://www.imagecraft.com/software/demos.html |
| Email Support: | mailto:support@imagecraft.com |
| License Request: | mailto:license@imagecraft.com |
| Savlo tiny URL: | |

http://www.imagecraft.com/software/
http://www.ece.utexas.edu/%7Evalvano
http://www.dragonsgate.net/FAQ/cache/20.html
http://www.imagecraft.com/software/mdevtools.html
http://www.dragonsgate.net/mailman/listinfo

## Technological Arts Links:

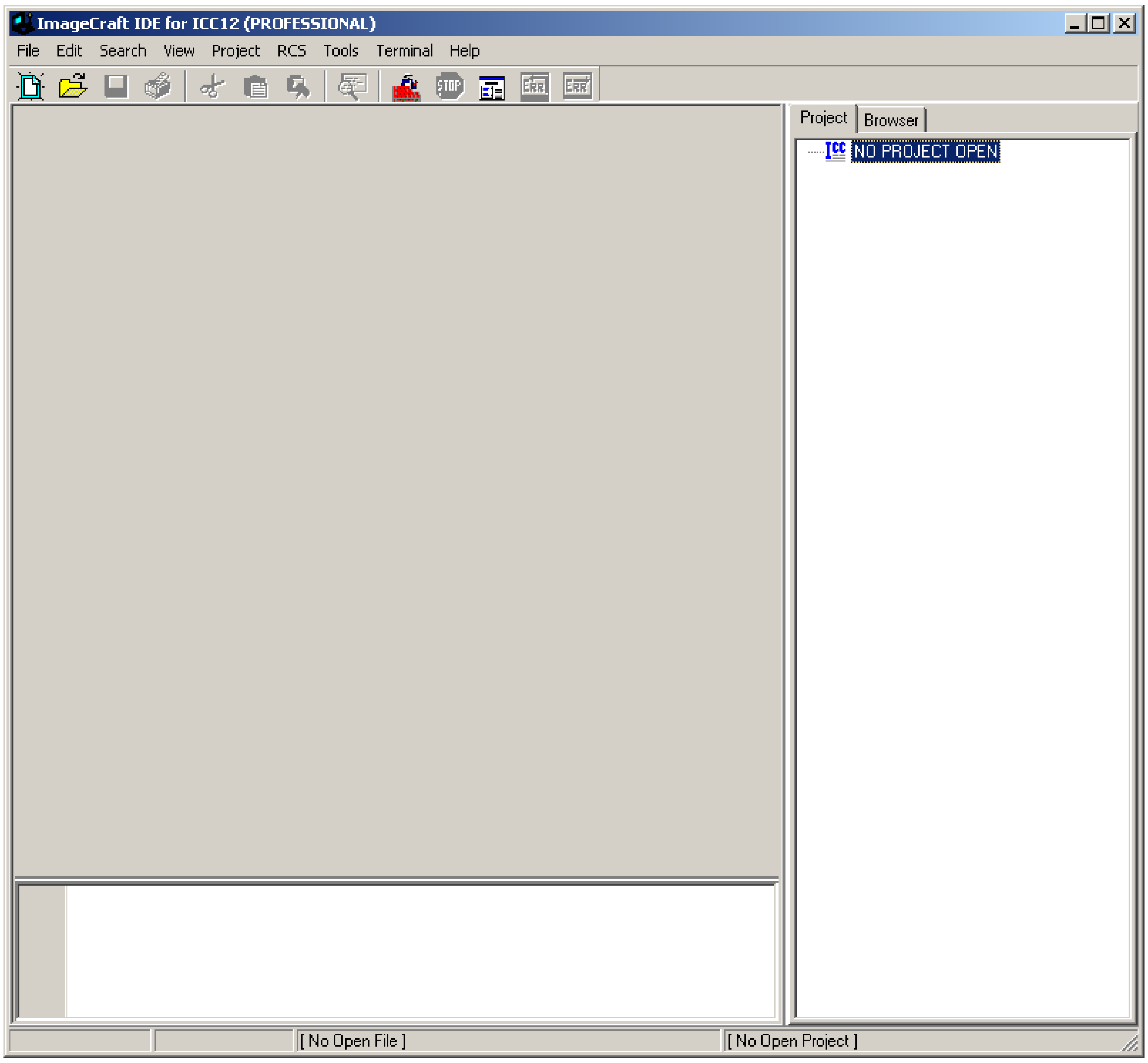
http://pages.interlog.com/%7Etechart/myfiles/AD9S12DP256.html
http://pages.interlog.com/%7Etechart/myfiles/manuals/ad9s12dp256man.pdf

## Getting Started:

Double click on the ICC12 icon. If a user has not read the ICC12 manual and just open the IDE one will wonder what to do next. Well wonder no more.

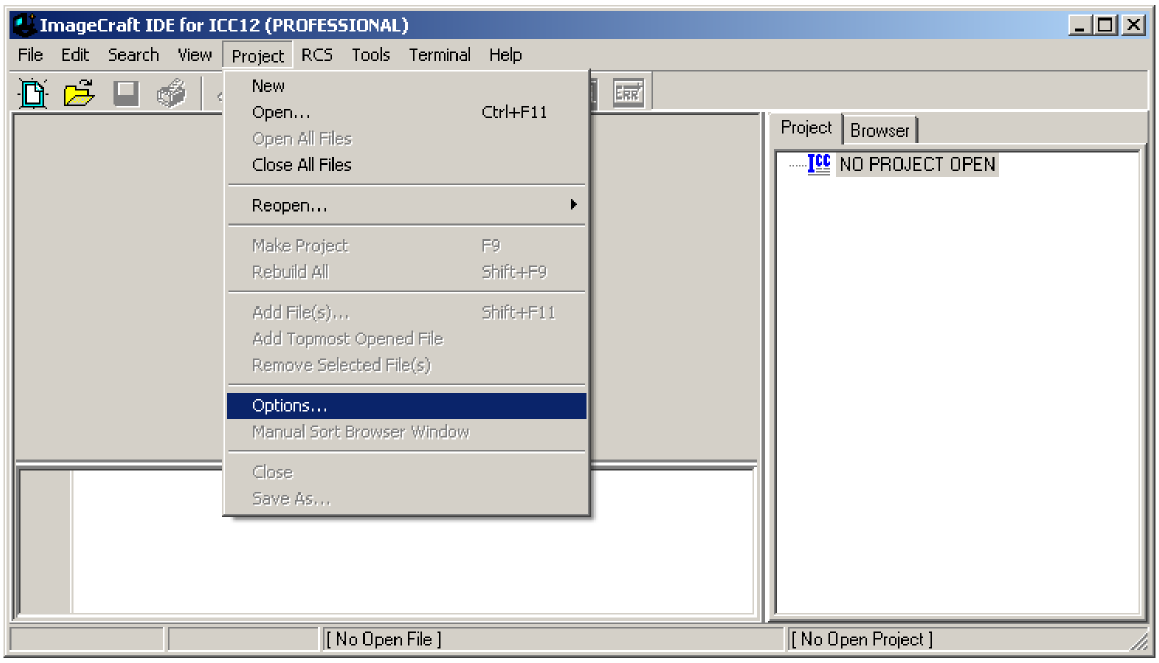
Note the 3 window panes. The top left most is greyed out and the right is the project window. The left bottom pane is where the error messages are displayed during compilation.

Before creating a new Project, the hardware target in the Compiler Options must be setup properly for the target MCU. This is to ensure that the compiler will setup the type of MCU the C program will compile for. In this example it is the Adapt9S12DP256.

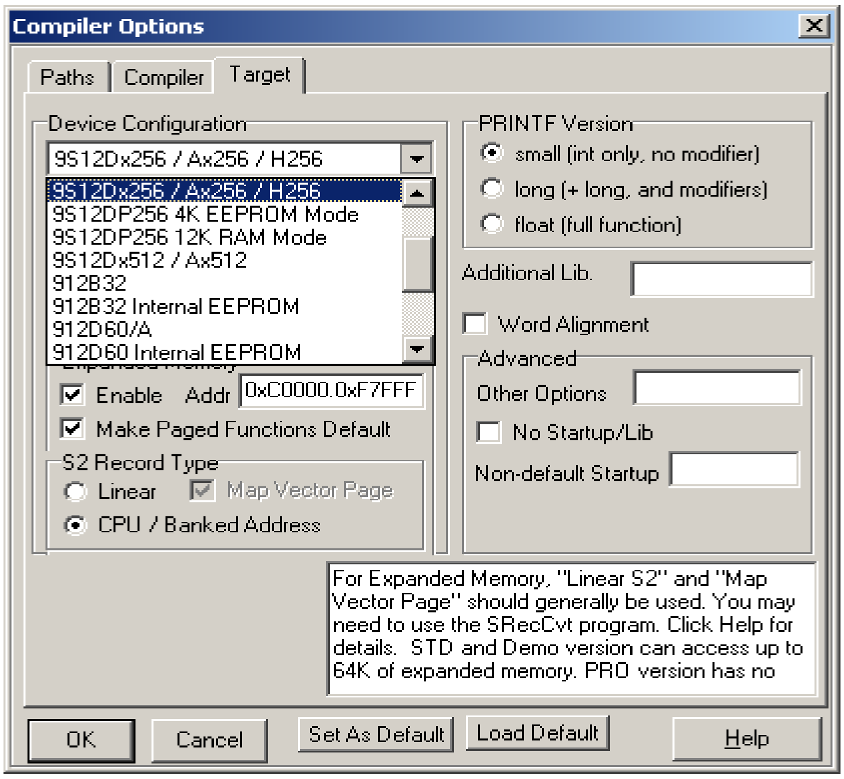

ImageCraft IDE for ICC12 (PROFESSIONAL)
File Edit Search View Project RCS Tools Terminal Help
Project Browser
NO PROJECT OPEN
[ No Open File ]
[ No Open Project ]

## Compiler Setup:

Click on Project Menu – Options – Target Tab.



Please note the Device Configuration. Click on the pull down arrow to change the device type.



Device Configuration
9S12Dx256 / Ax256 / H256

9S12Dx256 / Ax256 / H256
9S12DP256 4K EEPROM Mode
9S12DP256 12K RAM Mode
9S12Dx512 / Ax512
912B32
912B32 Internal EEPROM
912D60/A
912D60 Internal EEPROM

Expanded Memory
[✓] Enable Addr 0xC0000.0xF7FFF
[✓] Make Paged Functions Default

S2 Record Type
○ Linear [✓] Map Vector Page
● CPU / Banked Address

PRINTF Version
● small (int only, no modifier)
○ long (+ long, and modifiers)
○ float (full function)

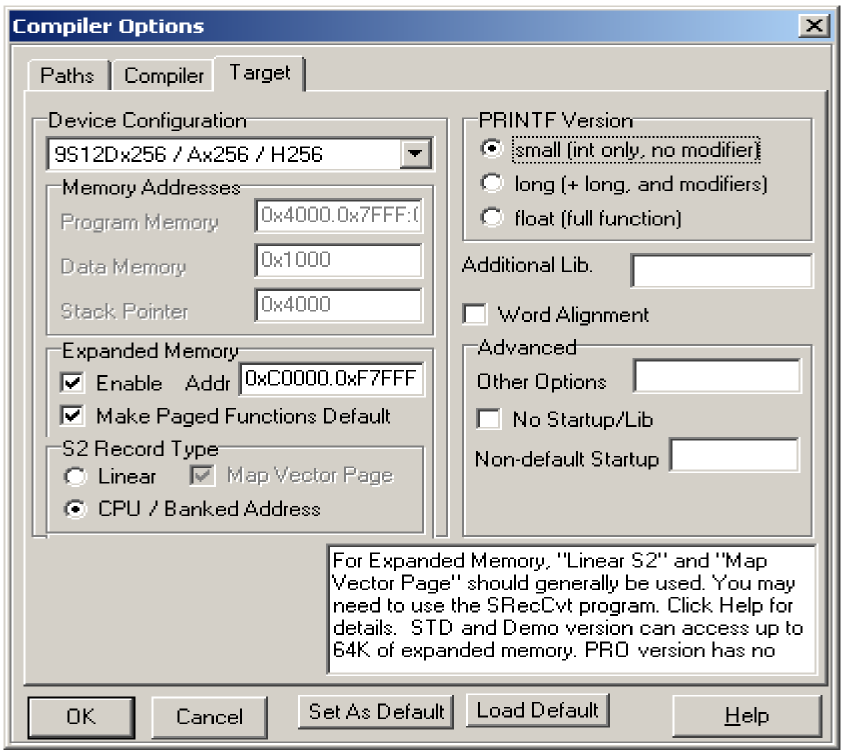
Additional Lib.

☐ Word Alignment

Advanced
Other Options
☐ No Startup/Lib
Non-default Startup

For Expanded Memory, "Linear S2" and "Map Vector Page" should generally be used. You may need to use the SRecCvt program. Click Help for details. STD and Demo version can access up to 64K of expanded memory. PRO version has no

Scroll up or down to select 9S12Dx256/Ax256/H256 as shown.



Device Configuration: 9S12Dx256 / Ax256 / H256

Memory Addresses
Program Memory: 0x4000.0x7FFF:(
Data Memory: 0x1000
Stack Pointer: 0x4000

Expanded Memory
[✓] Enable Addr 0xC0000.0xF7FFF
[✓] Make Paged Functions Default

S2 Record Type
Linear [✓ CPU / Banked Address]
[✓] Map Vector Page

PRINTF Version
[✓ small (int only, no modifier)]
long (+ long, and modifiers)
float (full function)

Additional Lib.

[ ] Word Alignment

Advanced
Other Options
[ ] No Startup/Lib
Non-default Startup

For Expanded Memory, "Linear S2" and "Map Vector Page" should generally be used. You may need to use the SRecCvt program. Click Help for details. STD and Demo version can access up to 64K of expanded memory. PRO version has no

**Device Configuration:**

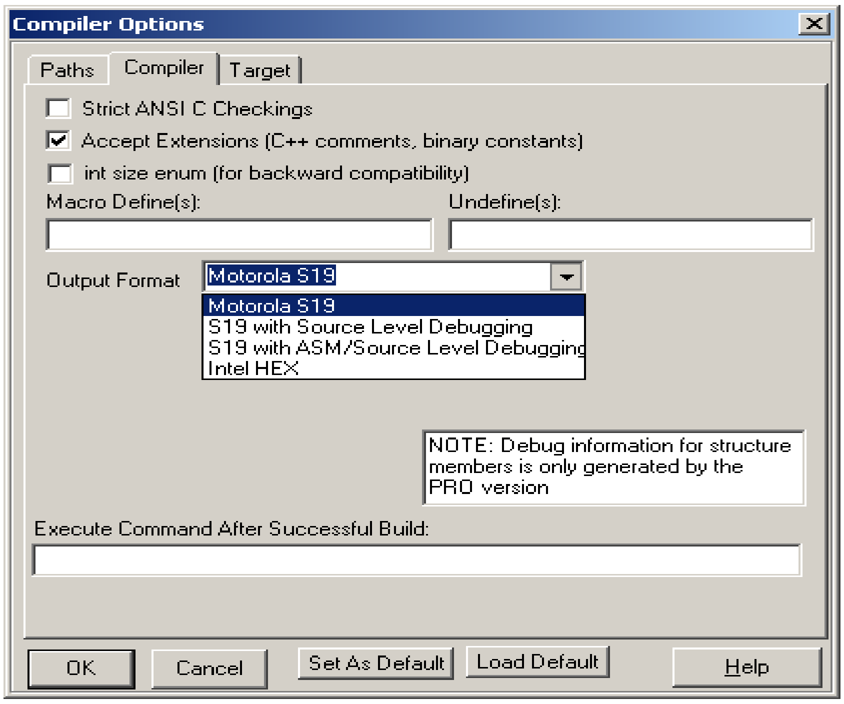
Program Memory: ***0x4000.0x7FFF:0xC000.0xFFFF***
Data Memory: ***0x1000***
Stack Pointer: ***0x4000***

**Expanded Memory:**

Note the address range is ***0xC0000. 0xF7FFF***. It defines the available PPAGE to be from $30 to $3D. PPAGE $3E and $3F are the fixed memory and are allocated for ***0x4000.0x7FFF:0xC000.0xFFFF***

On the compiler tab there are several choices of S-record output as shown.

Select which one that suits you.



Compiler Options

Paths | Compiler | Target

- [ ] Strict ANSI C Checkings
- [✓] Accept Extensions (C++ comments, binary constants)
- [ ] int size enum (for backward compatibility)

Macro Define(s):

Undefine(s):

Output Format: Motorola S19

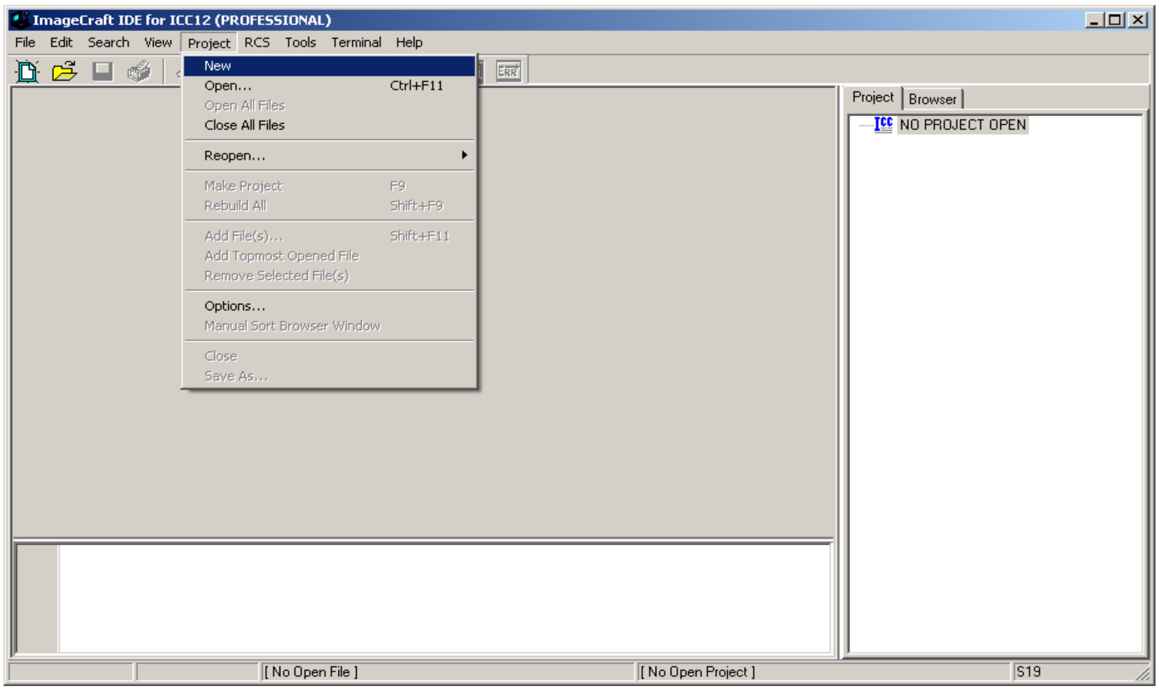
- Motorola S19
- S19 with Source Level Debugging
- S19 with ASM/Source Level Debugging
- Intel HEX

NOTE: Debug information for structure members is only generated by the PRO version

Execute Command After Successful Build:

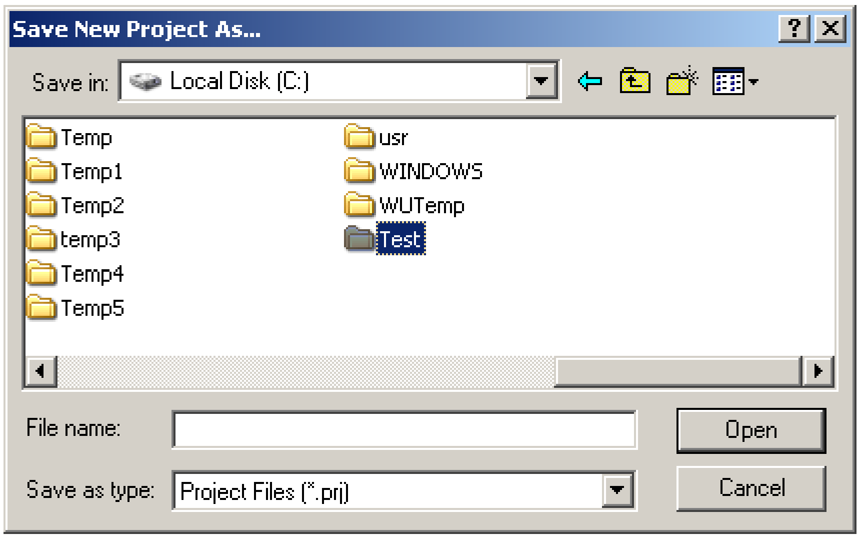
**Starting a new Project:**

Once the compiler options are setup, a new project can be created. Click Project menu – New.

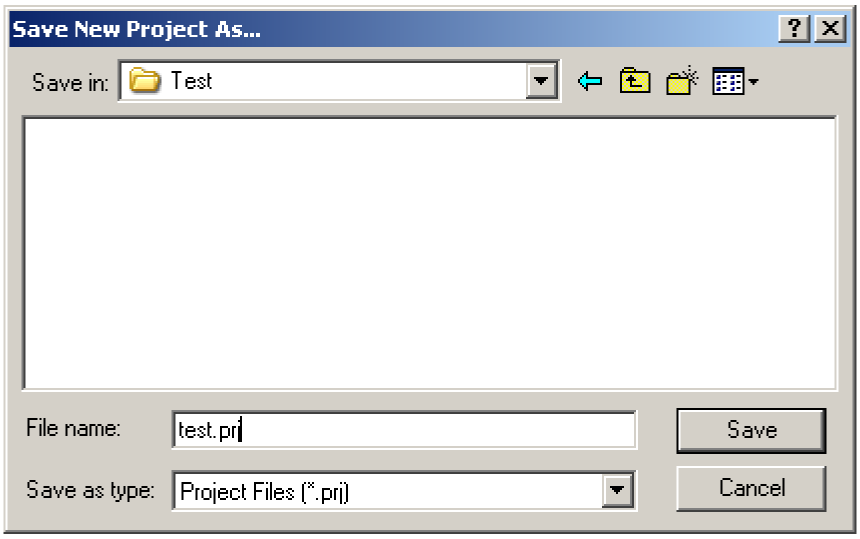


ImageCraft IDE for ICC12 (PROFESSIONAL)

The ICC12 will prompt to save the new project. The user should decide whether to create a new directory to save the new project. In this example a new directory called ***Test*** is created and the file is saved as file ***test.prj***.

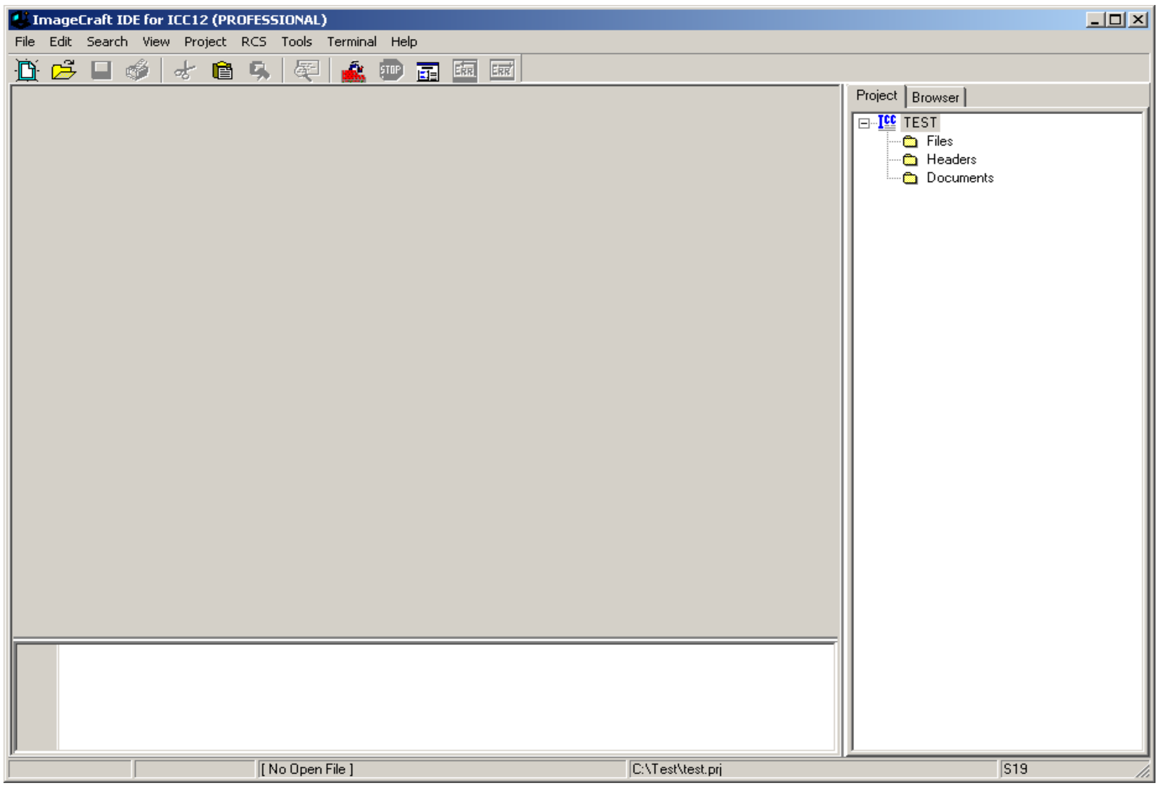


Type the filename as ***test.prj*** and click on the Save button.



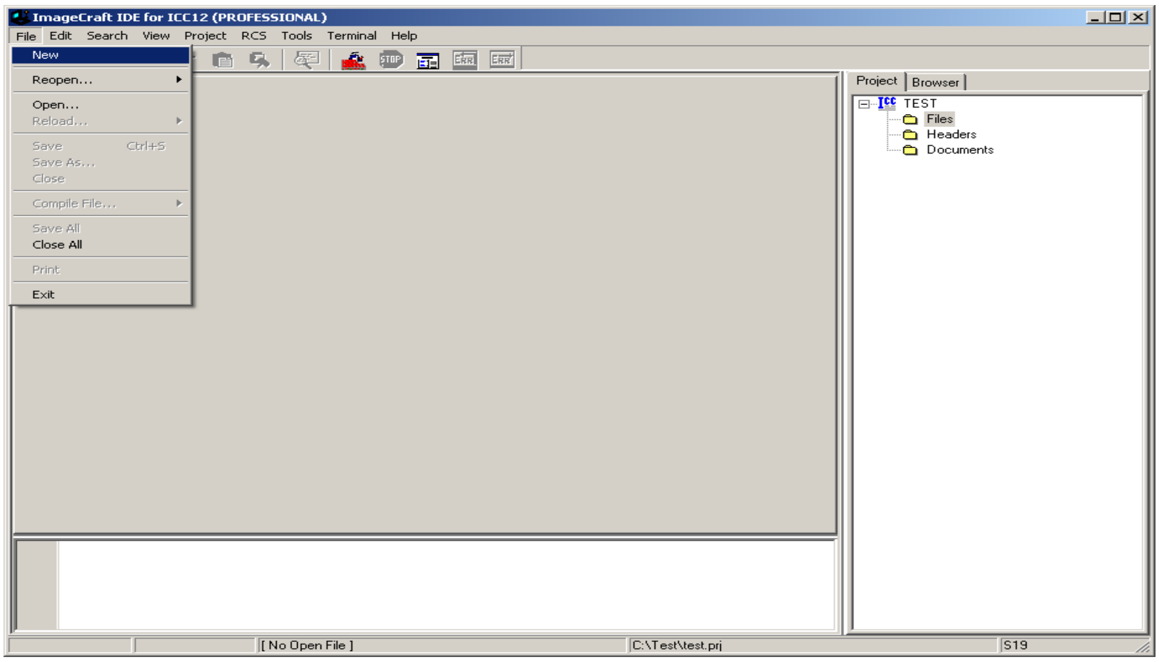
Note that the project window has changed to add Files, Headers and

Documents.

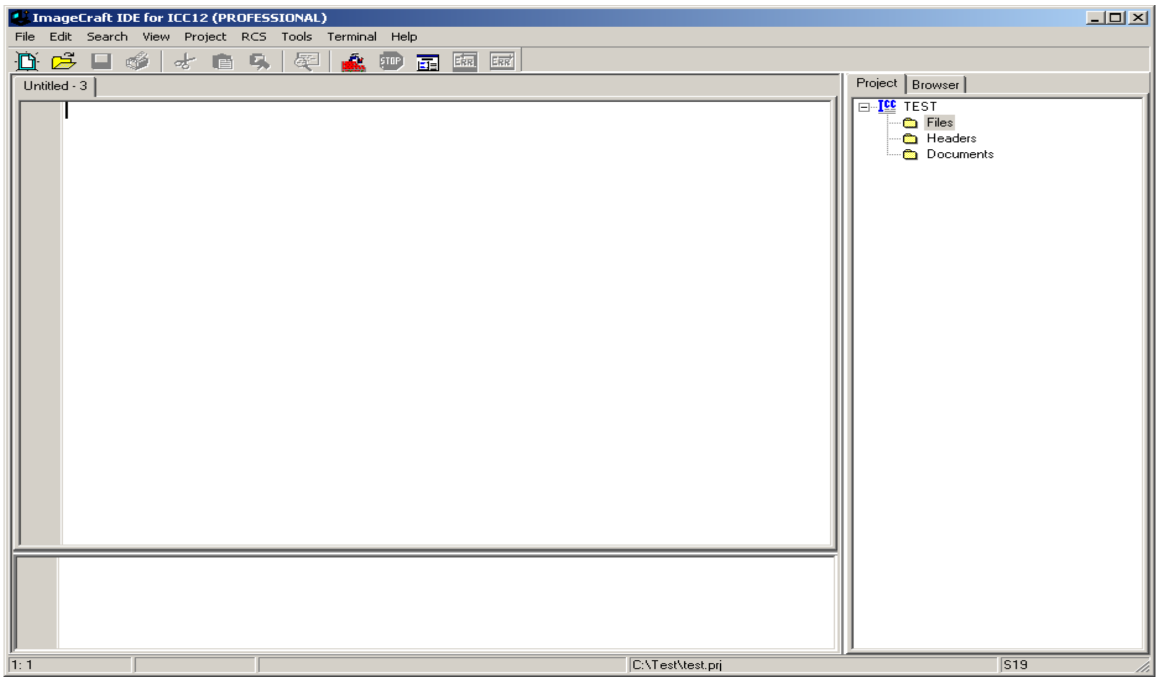

ImageCraft IDE for ICC12 (PROFESSIONAL)
File Edit Search View Project RCS Tools Terminal Help
Project Browser
TEST
Files
Headers
Documents
[ No Open File ]
C:\Test\test.prj
S19

## Creating a new file to the project:

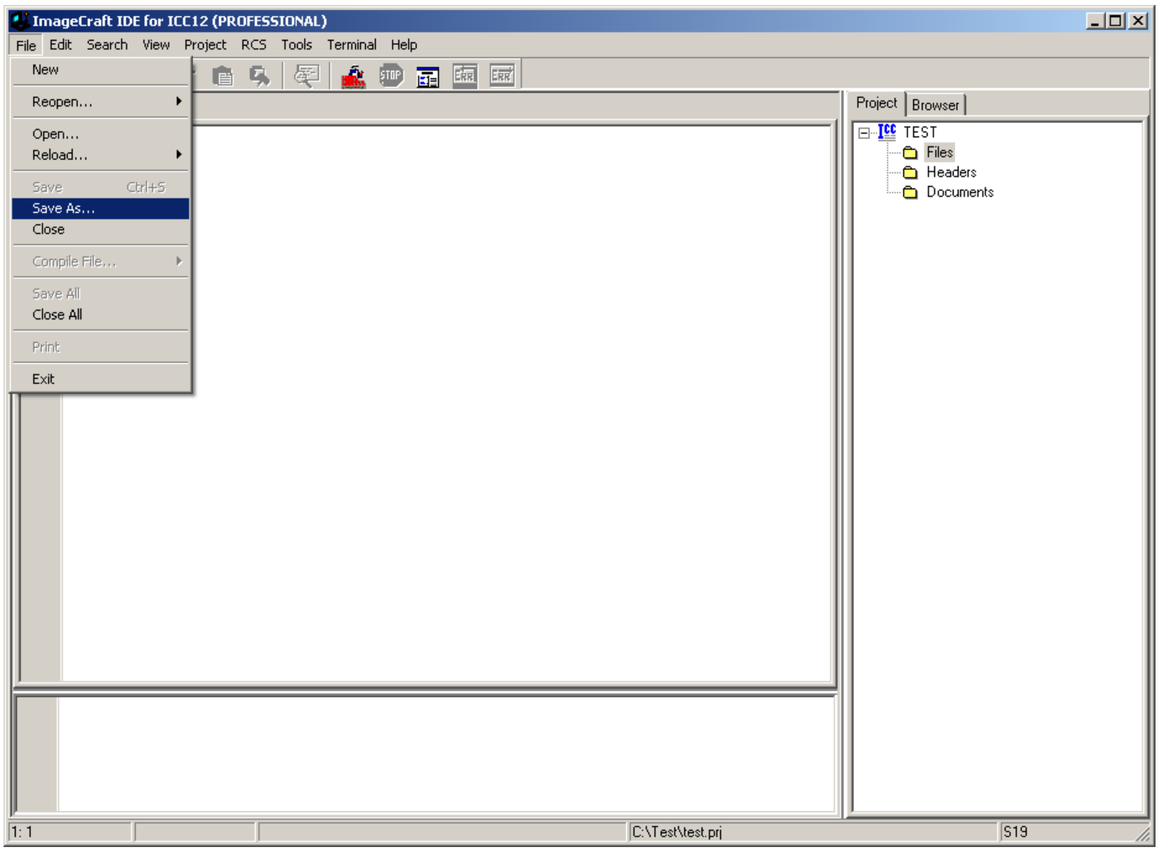
To add files to the project, click on the File menu – new as shown.



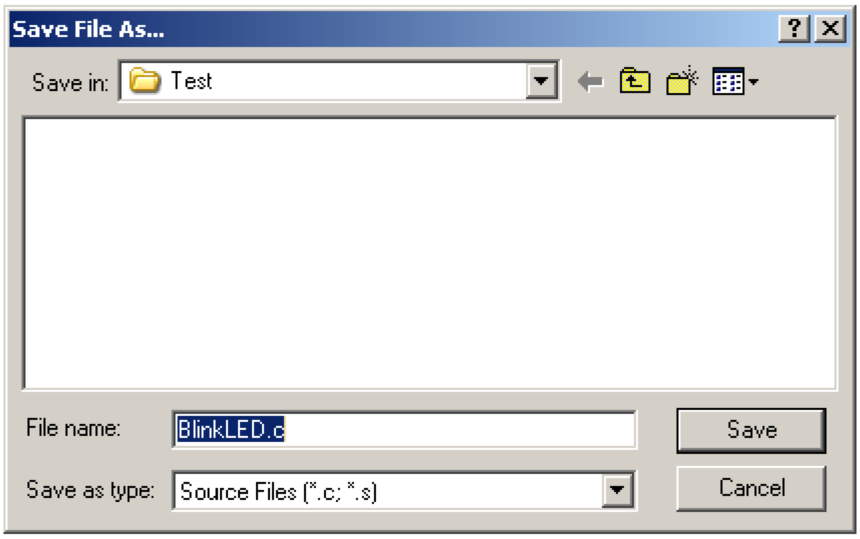
Note that ICC12 created an untitled file. Save the file as ***BlinkLED.C***.


ImageCraft IDE for ICC12 (PROFESSIONAL)
File Edit Search View Project RCS Tools Terminal Help
Untitled - 3
Project
Browser
TEST
Files
Headers
Documents
1: 1
C:\Test\test.prj
S19

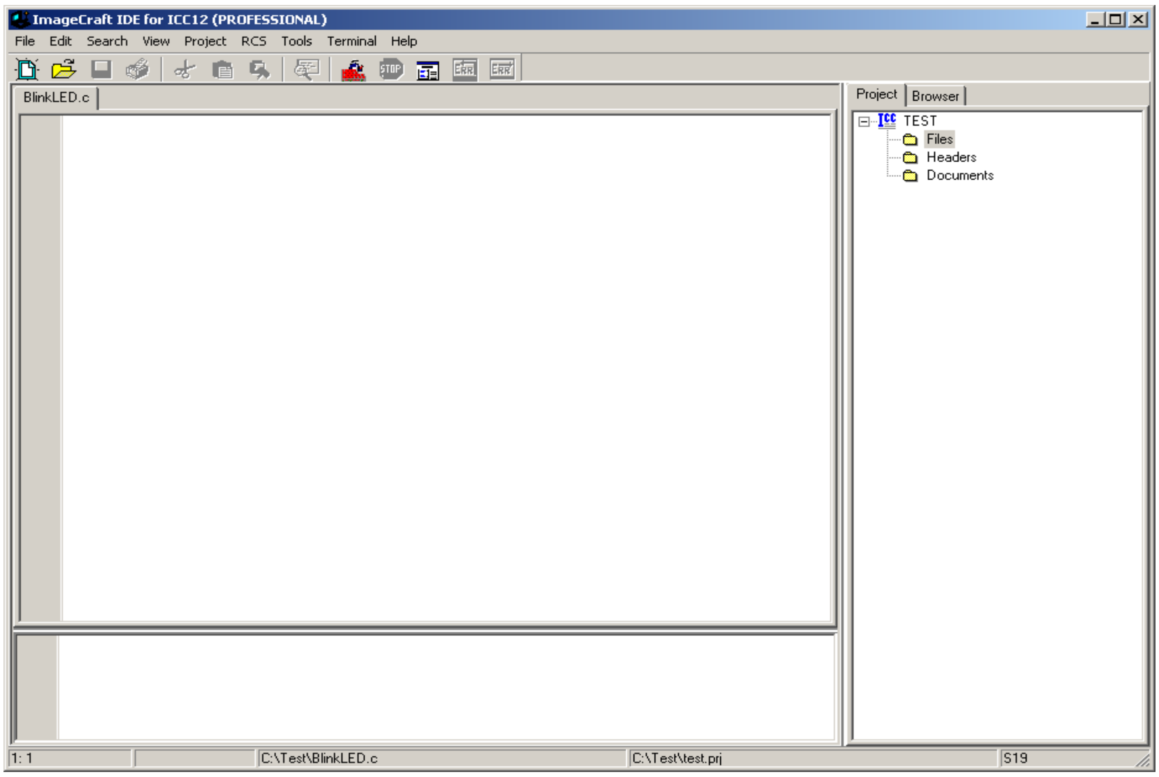
To save, click on File menu – Save As



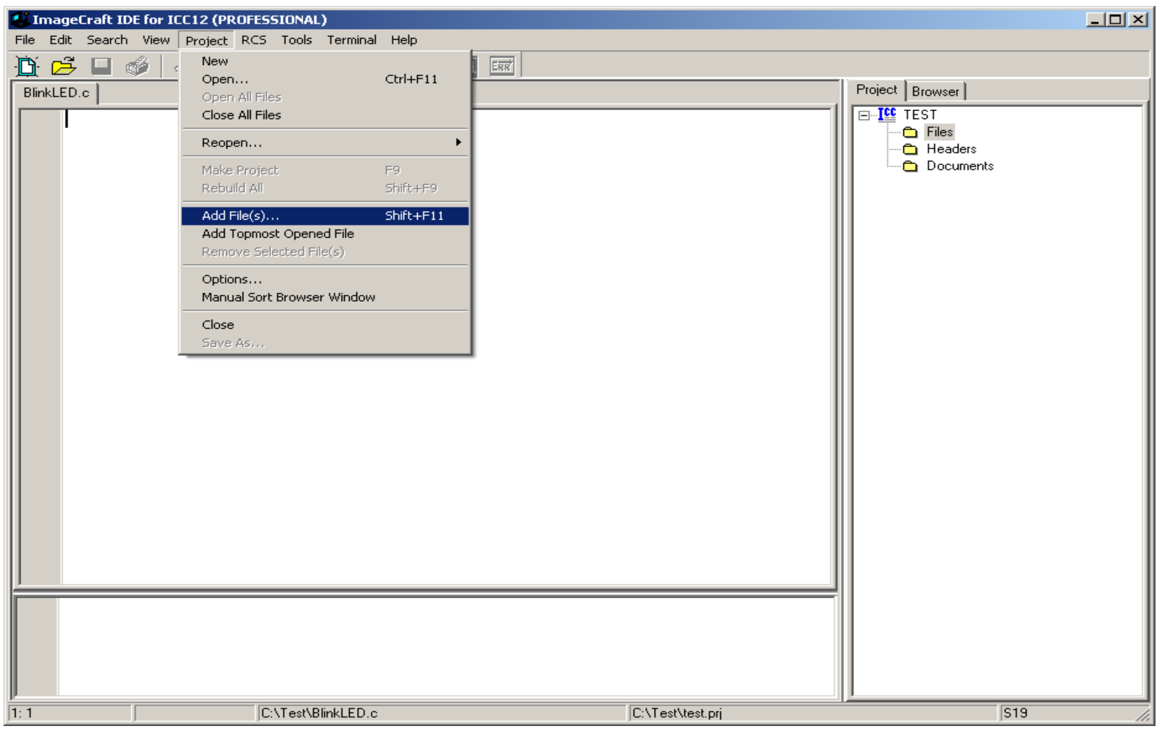
ICC12 will open an explorer window to help save the file. Type BlinkLED.c then press the save button.



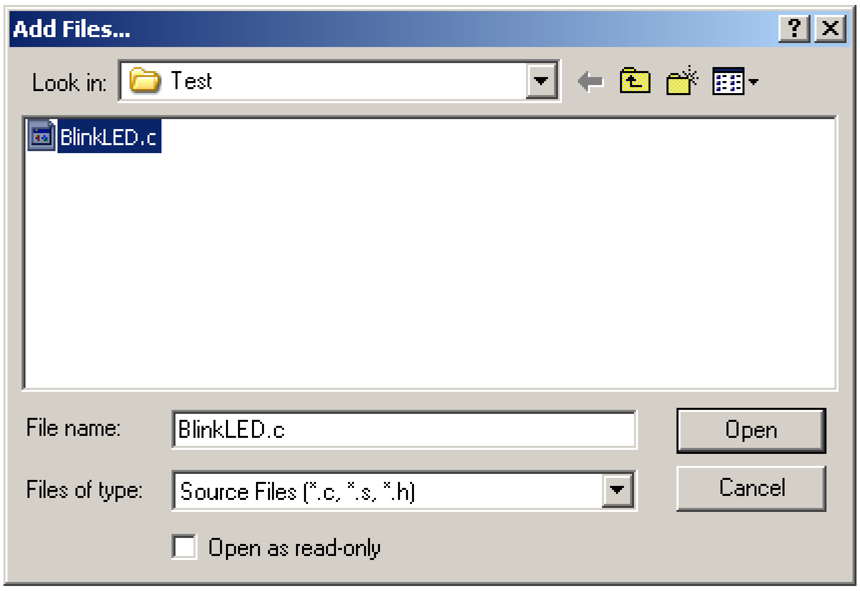
Note that ICC12 has renamed the file to BlinkLED.c.


ImageCraft IDE for ICC12 (PROFESSIONAL)
File Edit Search View Project RCS Tools Terminal Help
BlinkLED.c
Project Browser
TEST
Files
Headers
Documents
1: 1
C:\Test\BlinkLED.c
C:\Test\test.prj
S19

To add BlinkLED.c to the Project, click on the Project menu – Add File(s)



ICC12 will open an explorer window to help and locate the file of interest.



Look in: Test

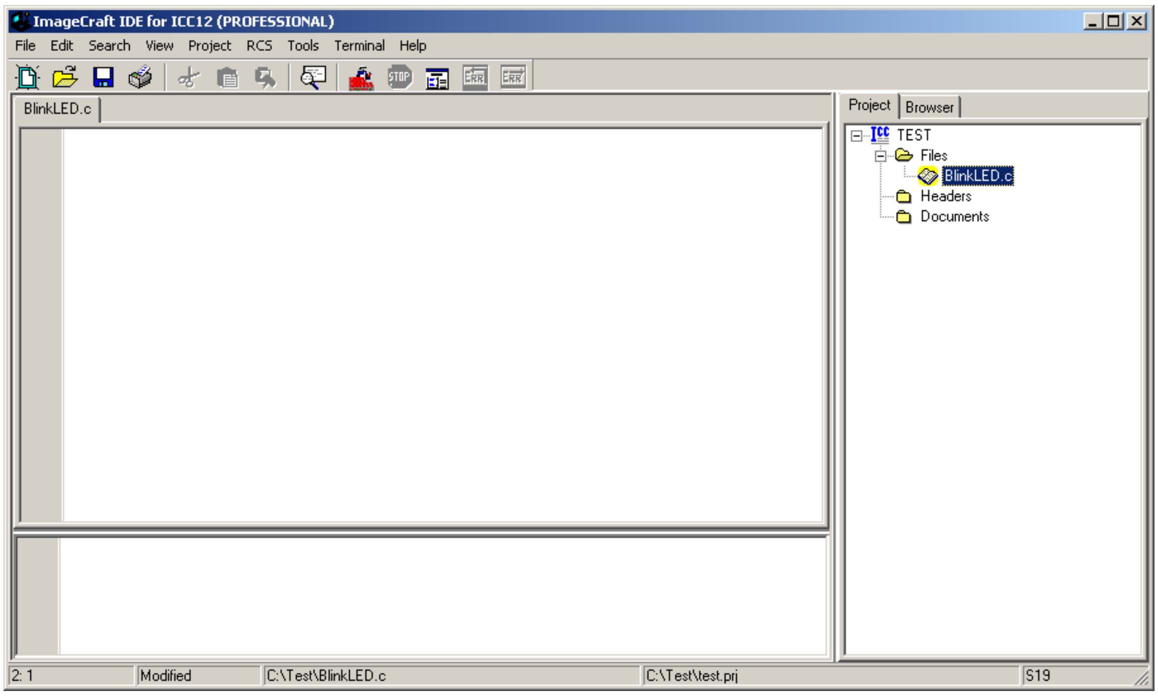
BlinkLED.c

File name: BlinkLED.c

Files of type: Source Files (*.c, *.s, *.h)

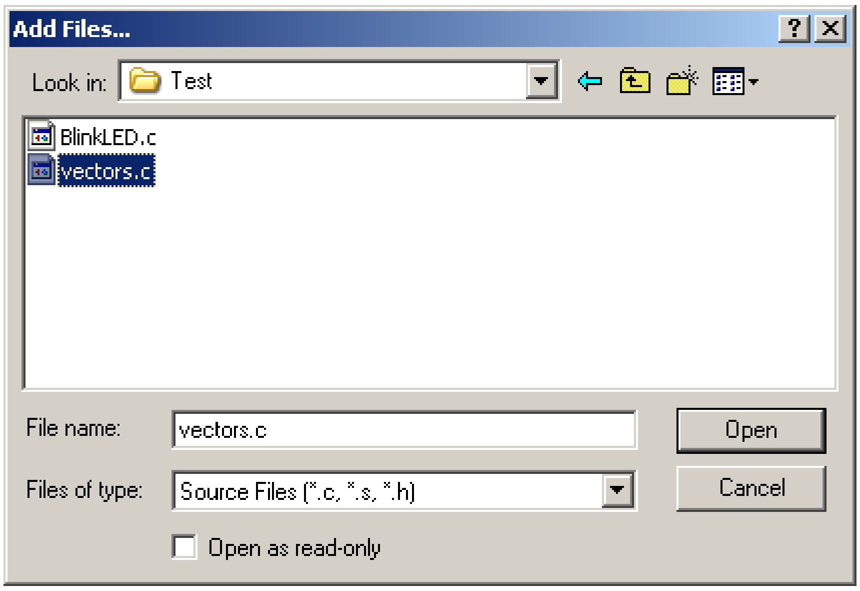
☐ Open as read-only

Note that the right window pane has changed to include BlinkLED.c under the Files Project.



Locate ***vectors.c*** and copy file to Test directory.  The major reason why this must be done is because of project to project dependency.  It is not good to keep editing a single ***vectors.c*** if other projects are using this same file.  It becomes a problem to keep track of the changes made to the different projects.

To add ***vectors.c*** to the Project, click on the Project menu – Add File(s)



Add Files...

Look in: Test

BlinkLED.c
vectors.c

File name: vectors.c

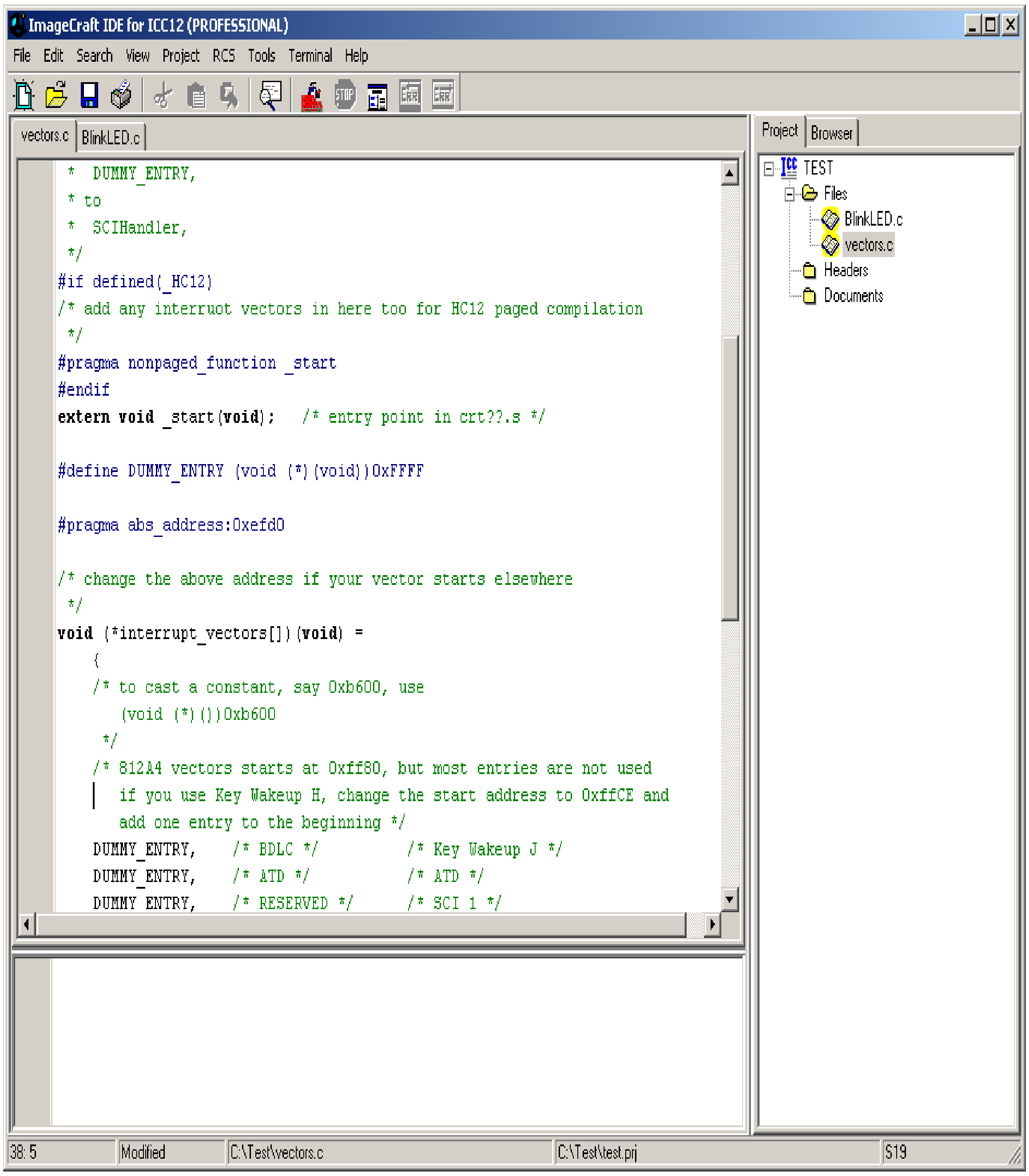
Files of type: Source Files (*.c, *.s, *.h)

Open as read-only

Open
Cancel

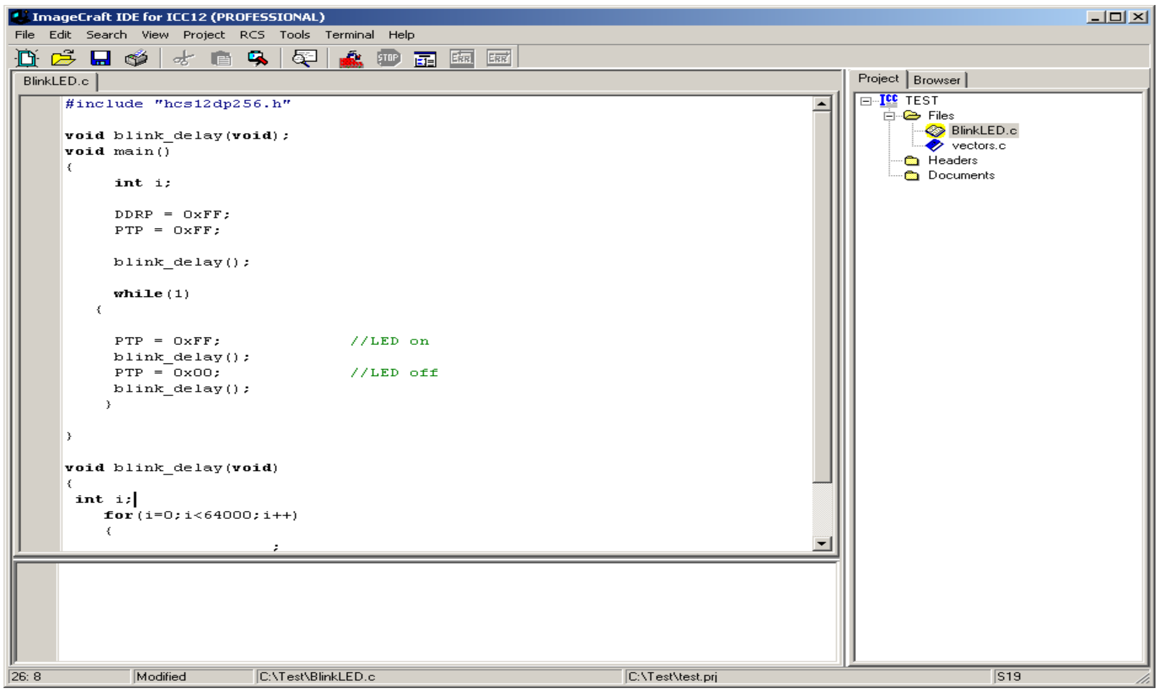
Note that ICC12 has changed to include ***vectors.c***  It is important to note that the **vectors.c** was written for the 68HC912B32 and 812A4 MCUs.  One should edit the file to include other ISR addresses for the 9S12DP256. This example edits the line ***#pragma abs_address:0xffd0*** to ***#pragma abs_address:0xefd0***

The original vector address 0xFFD0 is changed to 0xEFD0. This is because the FLASH Loader resides from $F000 to $FFFF.



```
  *  DUMMY_ENTRY,
  * to
  *  SCIHandler,
  */
#if defined(_HC12)
/* add any interruot vectors in here too for HC12 paged compilation
  */
#pragma nonpaged_function _start
#endif
extern void _start(void);   /* entry point in crt??.s */

#define DUMMY_ENTRY (void (*)(void))0xFFFF

#pragma abs_address:0xefd0

/* change the above address if your vector starts elsewhere
  */
void (*interrupt_vectors[])(void) =
    {
    /* to cast a constant, say 0xb600, use
       (void (*)())0xb600
      */
    /* 812A4 vectors starts at 0xff80, but most entries are not used
       if you use Key Wakeup H, change the start address to 0xffCE and
       add one entry to the beginning */
    DUMMY_ENTRY,    /* BDLC */         /* Key Wakeup J */
    DUMMY_ENTRY,    /* ATD */          /* ATD */
    DUMMY_ENTRY,    /* RESERVED */     /* SCI 1 */
```

Write the codes below into BlinkLED.c file. Once it is written we can then compile/make/build the code.

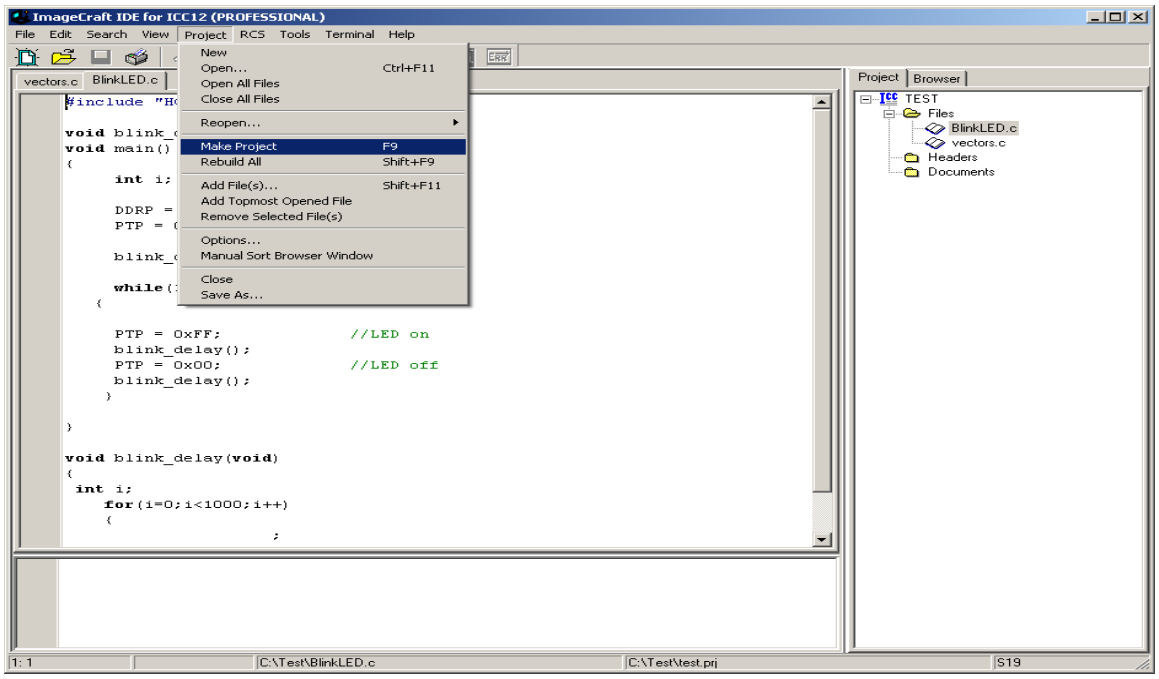


```
#include "hcs12dp256.h"

void blink_delay(void);
void main()
{
     int i;

     DDRP = 0xFF;
     PTP = 0xFF;

     blink_delay();

     while(1)
   {

     PTP = 0xFF;               //LED on
     blink_delay();
     PTP = 0x00;               //LED off
     blink_delay();
    }

}

void blink_delay(void)
{
 int i;
    for(i=0;i<64000;i++)
    {
                 ;
```

```
#include "hcs12dp256.h"

void blink_delay(void);
void main()
{
        int i;

        DDRP = 0xFF;
        PTP = 0xFF;

        blink_delay();

        while(1)
   {

        PTP = 0xFF;                    //LED on
        blink_delay();
        PTP = 0x00;                    //LED off
        blink_delay();
       }

}

void blink_delay(void)
{
 int i;
       for(i=0;i<64000;i++)
       {
                                   ;

       }
}
```

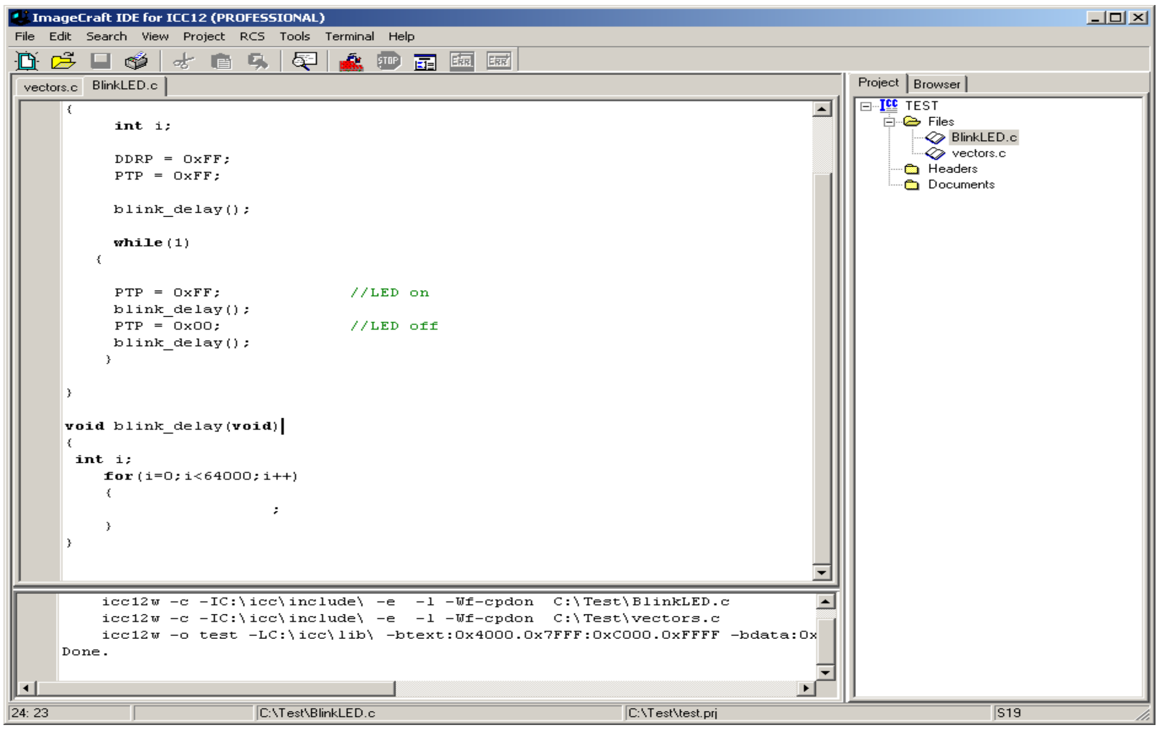
## Compiling/Build/Make the file:

To make the file click Project menu – make project as shown.



```
#include "Hc
void blink_c
void main()
{
     int i;

     DDRP =
     PTP = (

     blink_c

     while(:
   {

     PTP = 0xFF;               //LED on
     blink_delay();
     PTP = 0x00;               //LED off
     blink_delay();
    }

}

void blink_delay(void)
{
 int i;
    for(i=0;i<1000;i++)
    {
                    ;
```

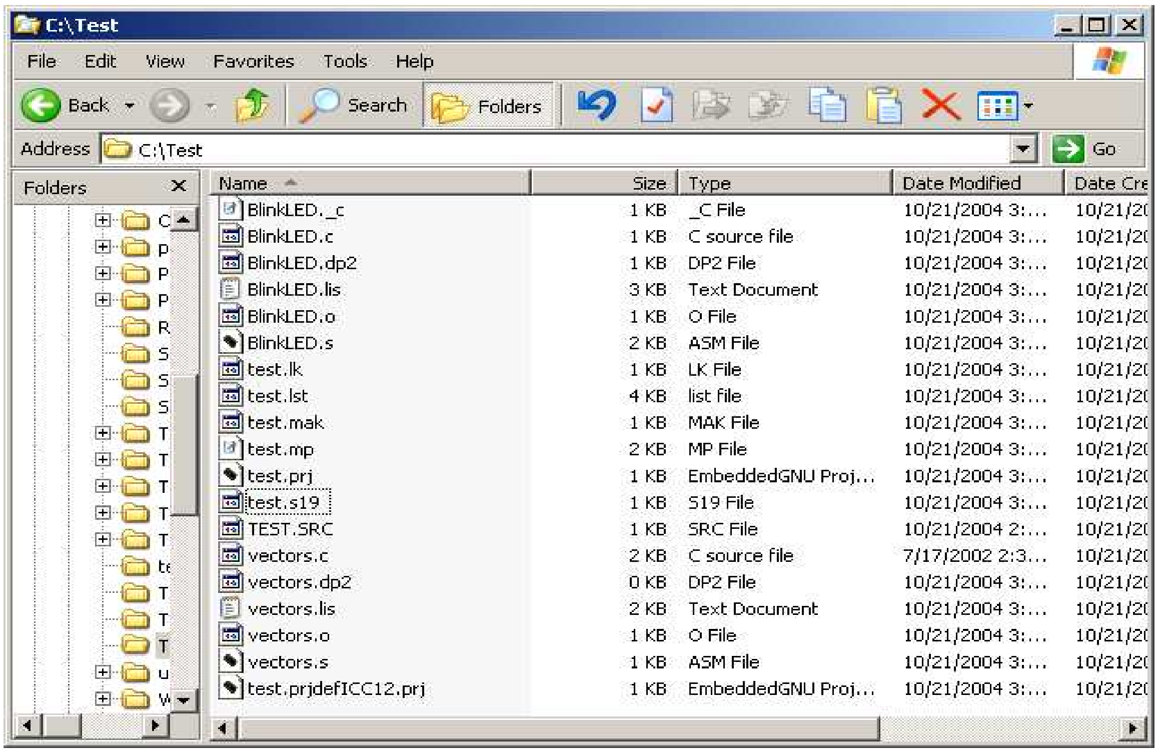
Note the bottom window pane will show messages to display how the build progressed.  Any errors, if any, are shown in this window.  The build was without error so we can progress to erasing and programming the 9S12DP256.



```
{
     int i;

     DDRP = 0xFF;
     PTP = 0xFF;

     blink_delay();

     while(1)
   {

     PTP = 0xFF;               //LED on
     blink_delay();
     PTP = 0x00;               //LED off
     blink_delay();
    }

}

void blink_delay(void)
{
 int i;
    for(i=0;i<64000;i++)
    {
                    ;
    }
}
```

```
    icc12w -c -IC:\icc\include\ -e  -l -Wf-cpdon  C:\Test\BlinkLED.c
    icc12w -c -IC:\icc\include\ -e  -l -Wf-cpdon  C:\Test\vectors.c
    icc12w -o test -LC:\icc\lib\ -btext:0x4000.0x7FFF:0xC000.0xFFFF -bdata:0x
Done.
```

Note the other extraneous files are created after a make.



| Name | Size | Type | Date Modified | Date Cre |
|---|---|---|---|---|
| BlinkLED._c | 1 KB | _C File | 10/21/2004 3:... | 10/21/2( |
| BlinkLED.c | 1 KB | C source file | 10/21/2004 3:... | 10/21/2( |
| BlinkLED.dp2 | 1 KB | DP2 File | 10/21/2004 3:... | 10/21/2( |
| BlinkLED.lis | 3 KB | Text Document | 10/21/2004 3:... | 10/21/2( |
| BlinkLED.o | 1 KB | O File | 10/21/2004 3:... | 10/21/2( |
| BlinkLED.s | 2 KB | ASM File | 10/21/2004 3:... | 10/21/2( |
| test.lk | 1 KB | LK File | 10/21/2004 3:... | 10/21/2( |
| test.lst | 4 KB | list file | 10/21/2004 3:... | 10/21/2( |
| test.mak | 1 KB | MAK File | 10/21/2004 3:... | 10/21/2( |
| test.mp | 2 KB | MP File | 10/21/2004 3:... | 10/21/2( |
| test.prj | 1 KB | EmbeddedGNU Proj... | 10/21/2004 3:... | 10/21/2( |
| test.s19 | 1 KB | S19 File | 10/21/2004 3:... | 10/21/2( |
| TEST.SRC | 1 KB | SRC File | 10/21/2004 2:... | 10/21/2( |
| vectors.c | 2 KB | C source file | 7/17/2002 2:3... | 10/21/2( |
| vectors.dp2 | 0 KB | DP2 File | 10/21/2004 3:... | 10/21/2( |
| vectors.lis | 2 KB | Text Document | 10/21/2004 3:... | 10/21/2( |
| vectors.o | 1 KB | O File | 10/21/2004 3:... | 10/21/2( |
| vectors.s | 1 KB | ASM File | 10/21/2004 3:... | 10/21/2( |
| test.prjdefICC12.prj | 1 KB | EmbeddedGNU Proj... | 10/21/2004 3:... | 10/21/2( |

Using WordPad to check the content of ***test.s19*** file. Note that the S-records are of different lengths.

**S10E4000CF400016405587CE10008E04**
**S110400B100027056A000820F6CE405ACDAB**
**S111401810008E405A2706180A307020F51644**
**S1074026402A20FE0A**
**S110402A34B7751B9EC6FF7B025AC6FF7B90**
**S110403702584A8000302010C6FF7B02585A**
**S10D40444A8000307902584A8000D7**
**S10A404E3020EEB757303DAE**
**S21230800034B7751B9ECC00006C1E2007EC1E9D**
**S21230800EC300016C1EEC1E8CFA0025F2B7572C**
**S20630801C300AF3**
**S111EFD0FFFFFFFFFFFFFFFFFFFFFFFFFFFF3D**
**S111EFDEFFFFFFFFFFFFFFFFFFFFFFFFFFFF2F**
**S111EFECFFFFFFFFFFFFFFFFFFFFFFFFFFFF21**
**S109EFFAFFFFFFFF4000D1**
**S10840551D0016073DEB**
**S9034000BC**

**Examining S-record:**

If one looks closely at the S-record one can see a mixture of S1 and S2 lines. This is atypical S-record generated by ICC12. S1 records are programmed in the ***$4000 - $7FFF*** and ***$C000 - $FFFF*** memory blocks. Any ISR routines are always in the fix memory location. The ISR can call any routine inside a PPAGE when necessary. The S2 can also be a fixed memory but are typically paged by ICC12

As stated previously, FLASH Loader occupies $F000 to $FFFF therefore the vector address at below $F000.

Note the content of the memory address at $E7FE:$E7FF is $4000, the RESET vector.

```
S111EFD0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF3D
S111EFDEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF2F
S111EFECFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF21
S109EFFAFFFFFFFF4000D1
```

The S-record below is the start of code. The content of address beginning at $4000 to $4050

```
S10E4000CF400016405587CE10008E04
S110400B100027056A000820F6CE405ACDAB
S111401810008E405A2706180A307020F51644
S1074026402A20FE0A
S110402A34B7751B9EC6FF7B025AC6FF7B90
S110403702584A8000302010C6FF7B02585A
S10D40444A8000307902584A8000D7
S10A404E3020EEB757303DAE
```

Please note the PPAGE = 30 the first available PPAGE as defined by address range 0xC0000 to 0xF7FFF.

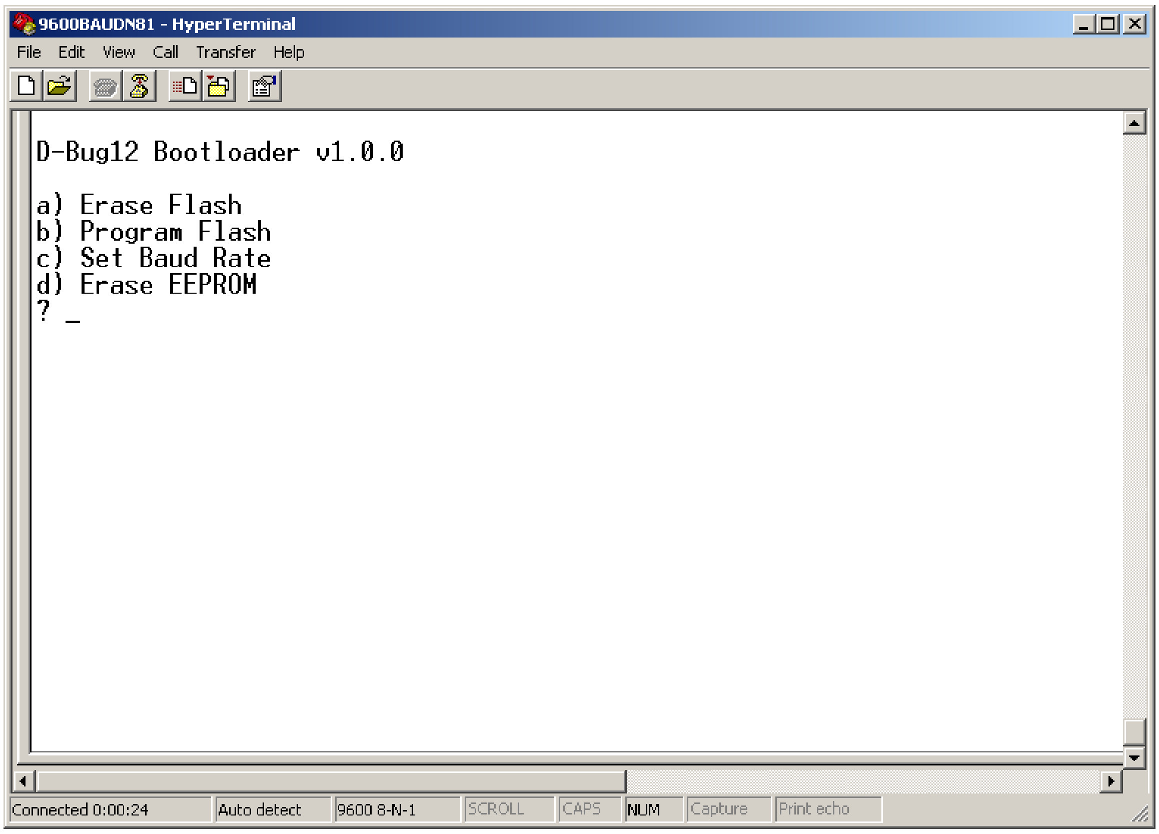
```
S21230800034B7751B9ECC00006C1E2007EC1E9D
S21230800EC300016C1EEC1E8CFA0025F2B7572C
S20630801C300AF3
```

**Programming the Adapt9S12DP256:**

Any terminal program can be use to communicate to the FLASH Loader. In this example HyperTerm is used. Connect Serial cable to any available PC COM port and the other end to Adapt9S12DP256.

Open HyperTerm and set BAUD = 9600, 8, N, 1. Make sure to enable XON/OFF.

Move jumpers on JB3, PAD1:PAD0 = 1:1. Power up unit or press the RESET button if already powered up.

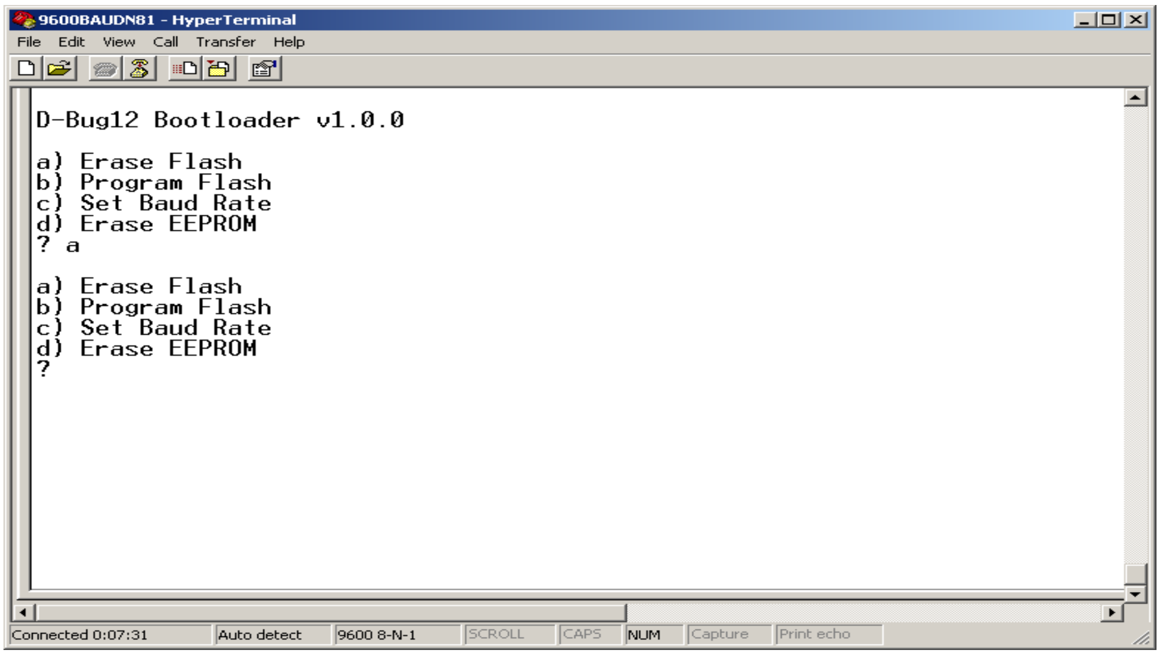

9600BAUDN81 - HyperTerminal
File Edit View Call Transfer Help

```
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? _
```

Connected 0:00:24 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

**Erasing:**

To erase, select ***a***



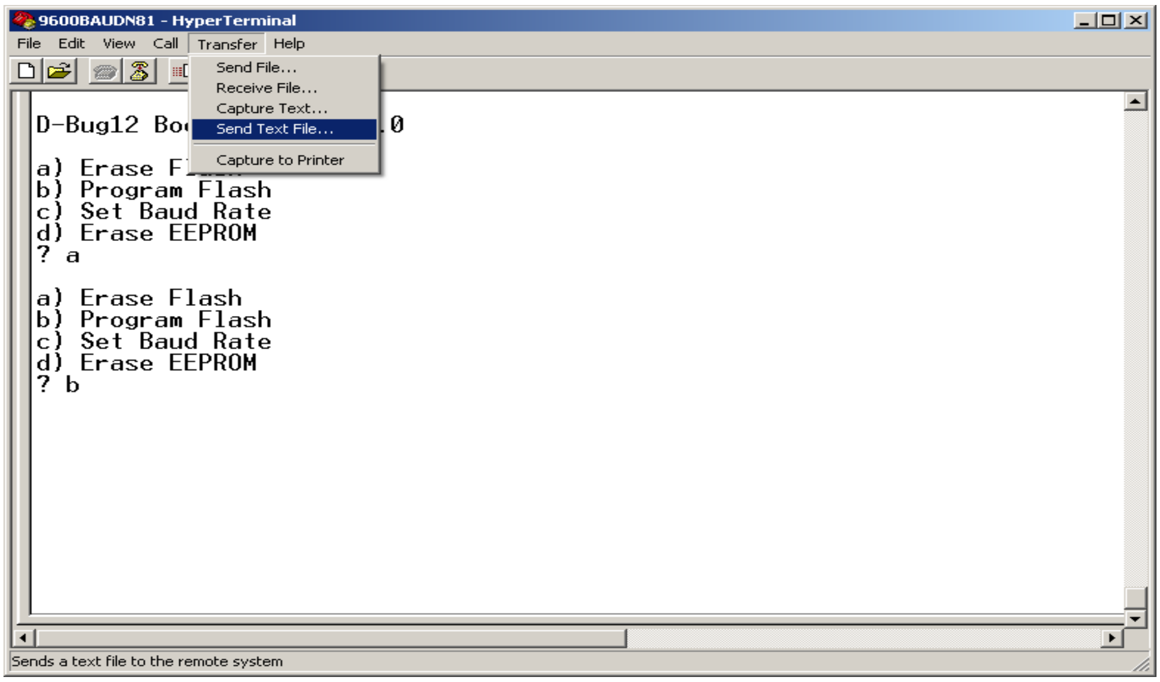
9600BAUDN81 - HyperTerminal
File Edit View Call Transfer Help

```
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? a

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?
```

Connected 0:07:31 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

**Programming:**

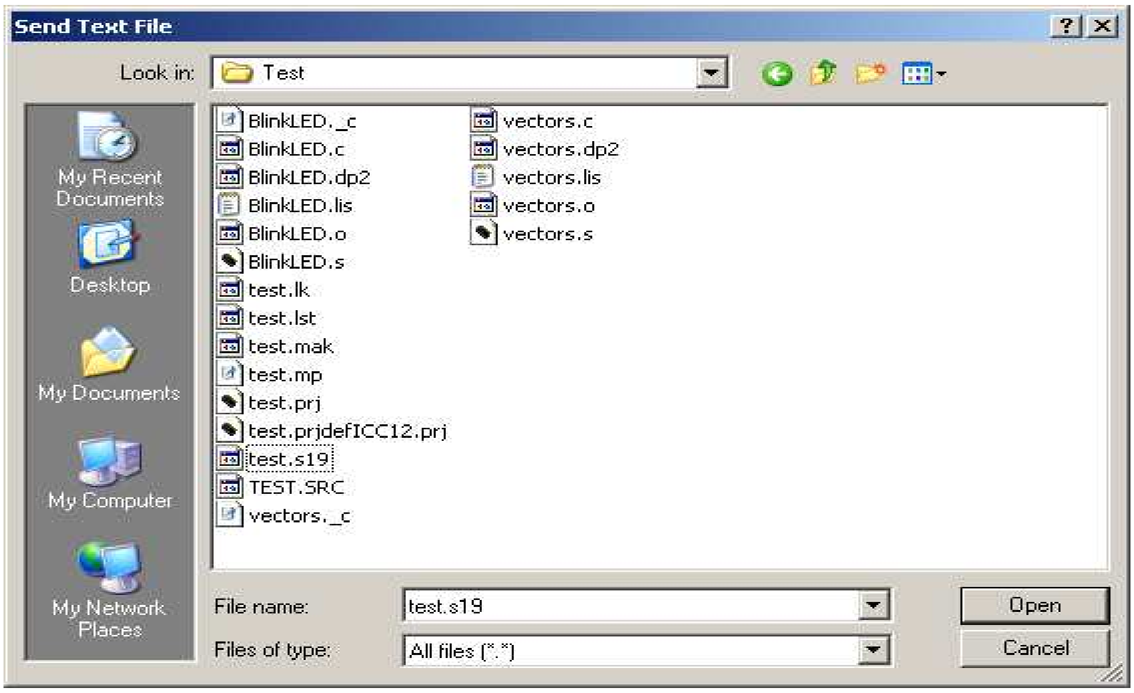
To program - select ***b.*** Once ***b*** is pressed, locate the file to upload. It must be sent as text. Press Transfer menu – Send Text File



```
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? a

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? b
```

Sends a text file to the remote system

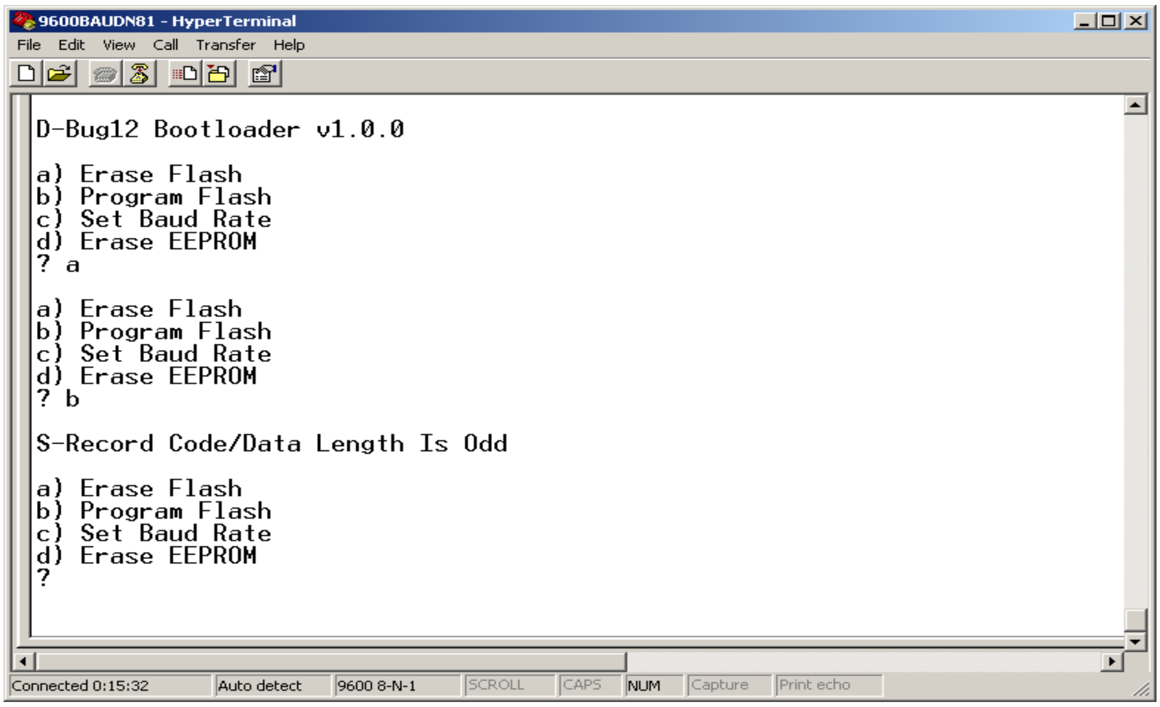
HyperTerm will open an explorer window to help and locate the S-record. Double click on test.s19 to initiate upload.



File name: test.s19

Files of type: All files (*.*)

ICC12 generates odd length S-records so an error will appear ***S-Record Code/Data Length Is Odd.*** The test.s19 must go thru an SrecCVT convert program to make the length even.

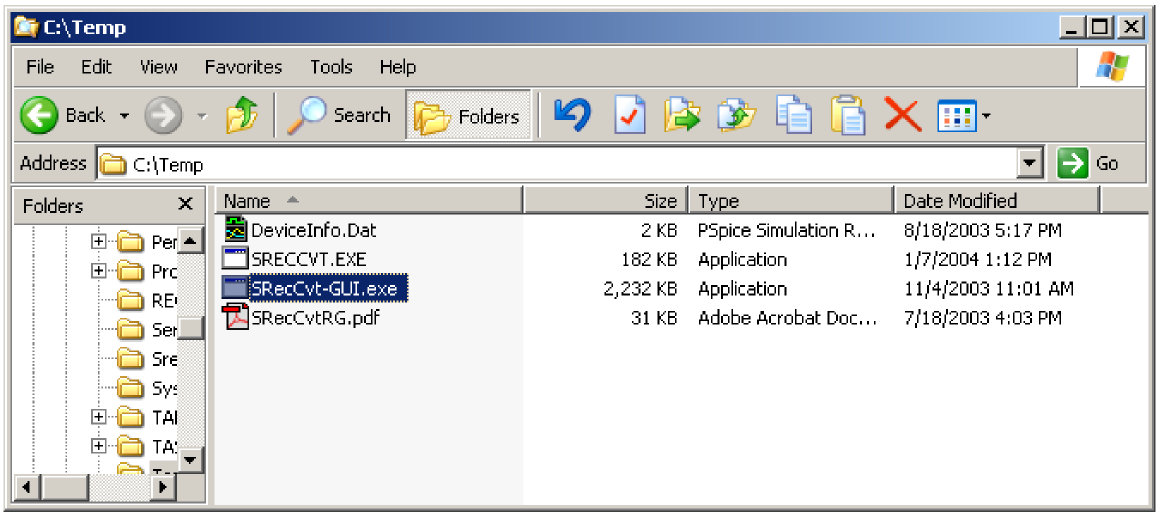

9600BAUDN81 - HyperTerminal
File Edit View Call Transfer Help

```
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? a

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? b

S-Record Code/Data Length Is Odd

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?
```

Connected 0:15:32 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

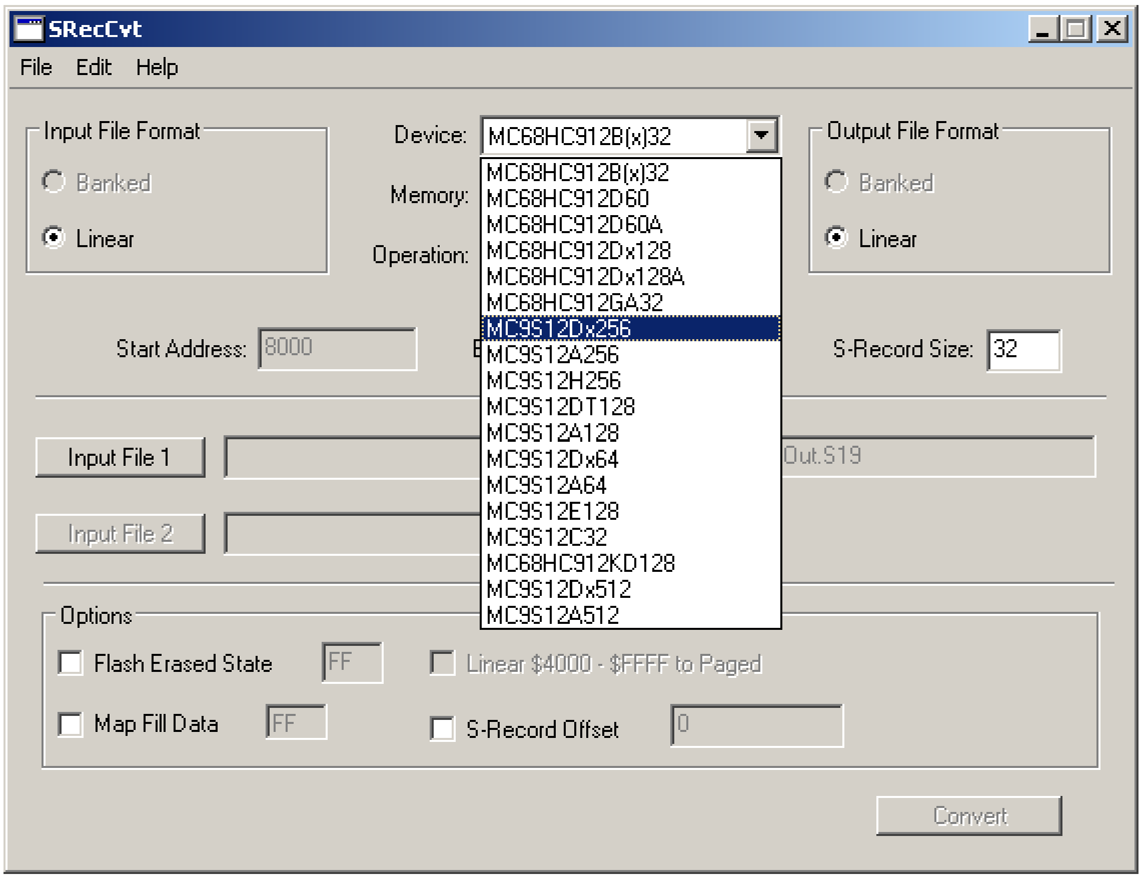
**SrecCVT:**

The ***DB12S12FW.zip*** from Freescale contains the SrecCVT convert program. This document will use the GUI base program to format the S-record. To start double click on the ***SRecCvt-GUI.exe***



| Name | Size | Type | Date Modified |
|---|---|---|---|
| DeviceInfo.Dat | 2 KB | PSpice Simulation R... | 8/18/2003 5:17 PM |
| SRECCVT.EXE | 182 KB | Application | 1/7/2004 1:12 PM |
| SRecCvt-GUI.exe | 2,232 KB | Application | 11/4/2003 11:01 AM |
| SRecCvtRG.pdf | 31 KB | Adobe Acrobat Doc... | 7/18/2003 4:03 PM |

Click on the pulldown to select Device to be MC9S12Dx256 as shown.



Input File Format: Banked [✓ Linear]

Device: MC68HC912B(x)32

- MC68HC912B(x)32
- MC68HC912D60
- MC68HC912D60A
- MC68HC912Dx128
- MC68HC912Dx128A
- MC68HC912GA32
- MC9S12Dx256
- MC9S12A256
- MC9S12H256
- MC9S12DT128
- MC9S12A128
- MC9S12Dx64
- MC9S12A64
- MC9S12E128
- MC9S12C32
- MC68HC912KD128
- MC9S12Dx512
- MC9S12A512

Output File Format: Banked [✓ Linear]

Start Address: 8000

S-Record Size: 32

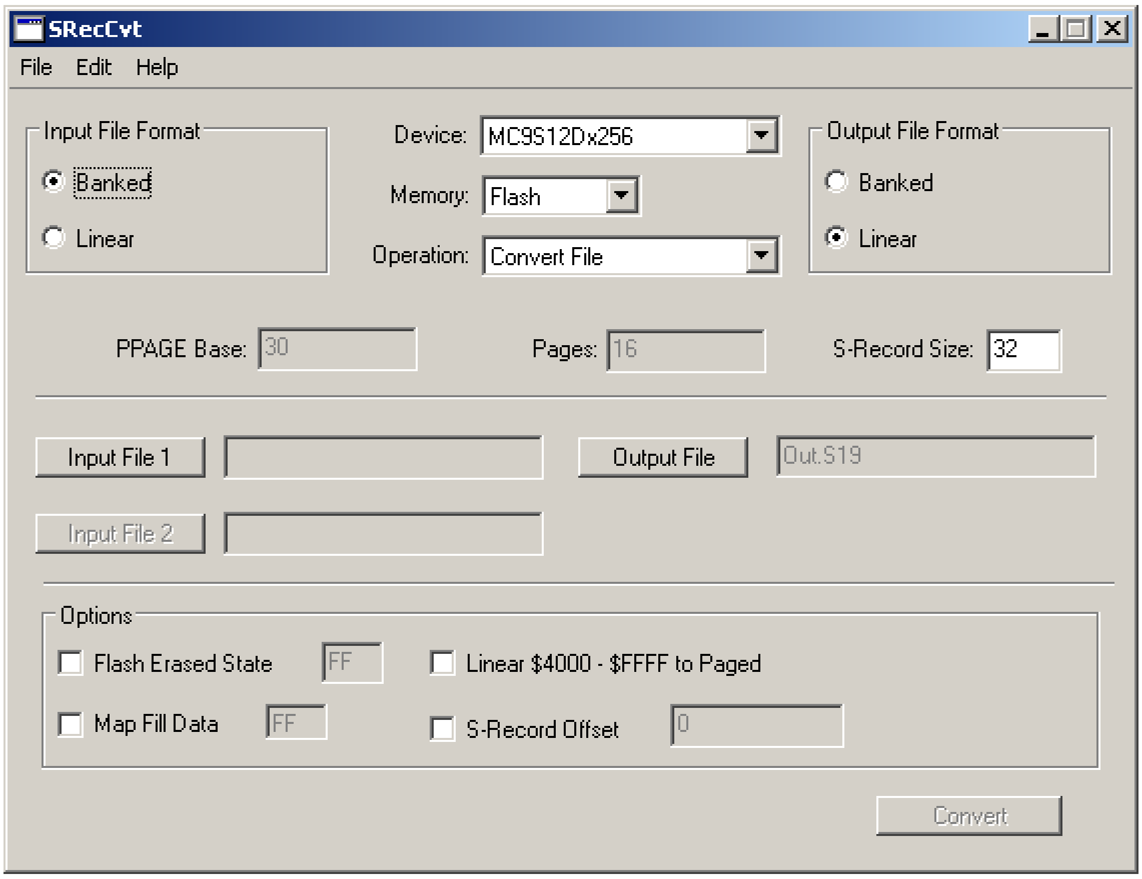
Out.S19

Flash Erased State: FF

Map Fill Data: FF

S-Record Offset: 0

Select the Input File Format as Banked and the Output File Format as Linear.



SRecCvt

File Edit Help

Input File Format: [✓ Banked] Linear

Device: MC9S12Dx256

Memory: Flash

Operation: Convert File

Output File Format: Banked [✓ Linear]

PPAGE Base: 30

Pages: 16

S-Record Size: 32

Output File: Out.S19

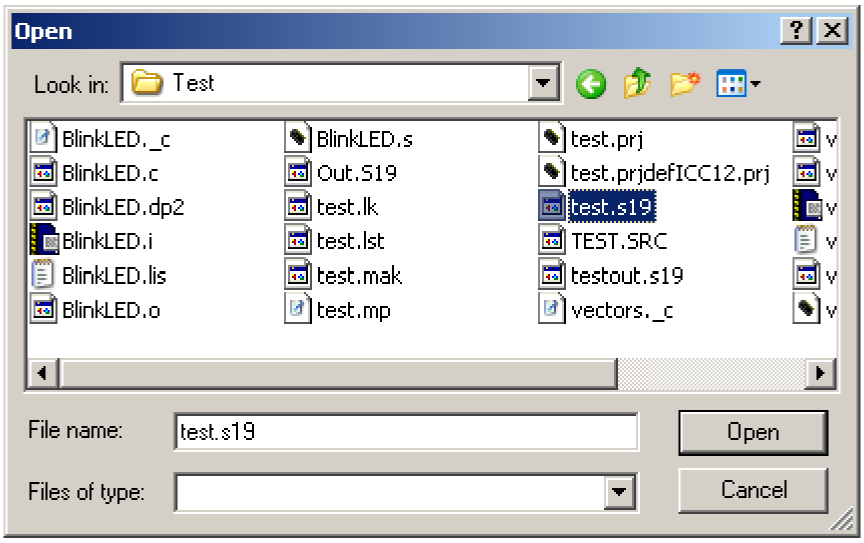
Options

Flash Erased State: FF

Map Fill Data: FF

S-Record Offset: 0

Press on the Input File 1 button to locate ***test.s19***



Open

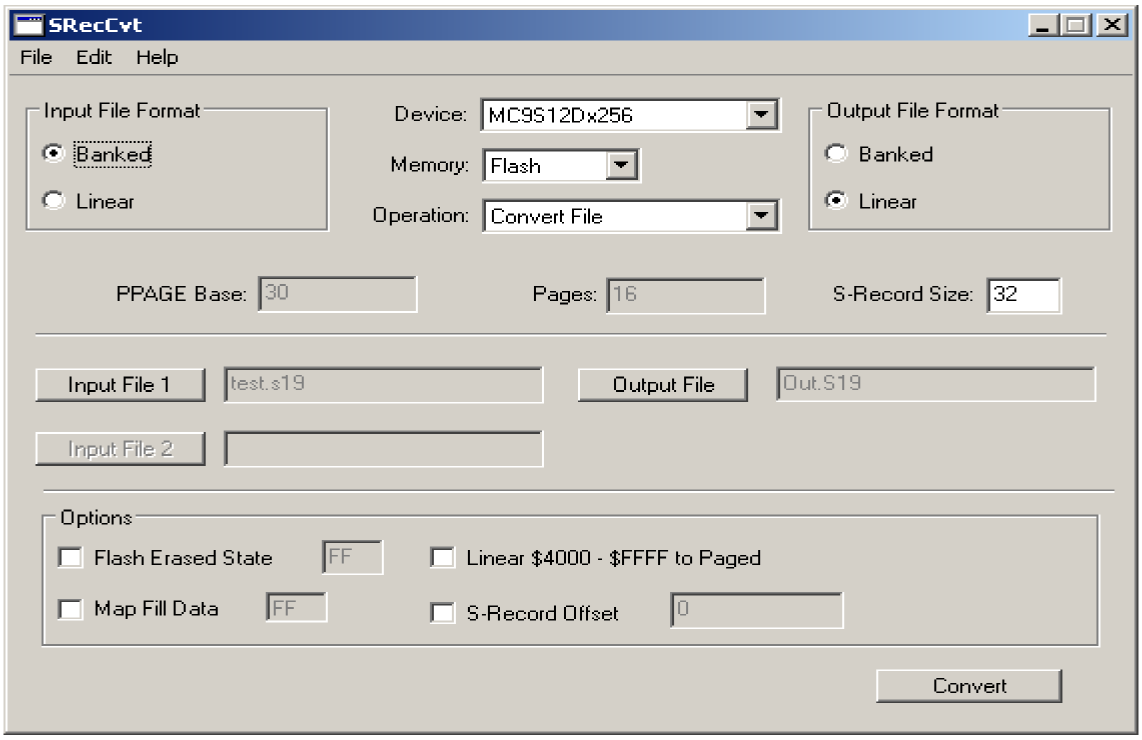
Look in: Test

BlinkLED._c
BlinkLED.c
BlinkLED.dp2
BlinkLED.i
BlinkLED.lis
BlinkLED.o
BlinkLED.s
Out.S19
test.lk
test.lst
test.mak
test.mp
test.prj
test.prjdefICC12.prj
test.s19
TEST.SRC
testout.s19
vectors._c

File name: test.s19

Files of type:

The output file is set as out.s19 but one can use or select a different filename if so desired.


SRecCvt

File Edit Help

Input File Format: [✓ Banked] Linear

Device: MC9S12Dx256

Memory: Flash

Operation: Convert File

Output File Format: Banked [✓ Linear]

PPAGE Base: 30

Pages: 16

S-Record Size: 32

Input File 1: test.s19

Output File: Out.S19

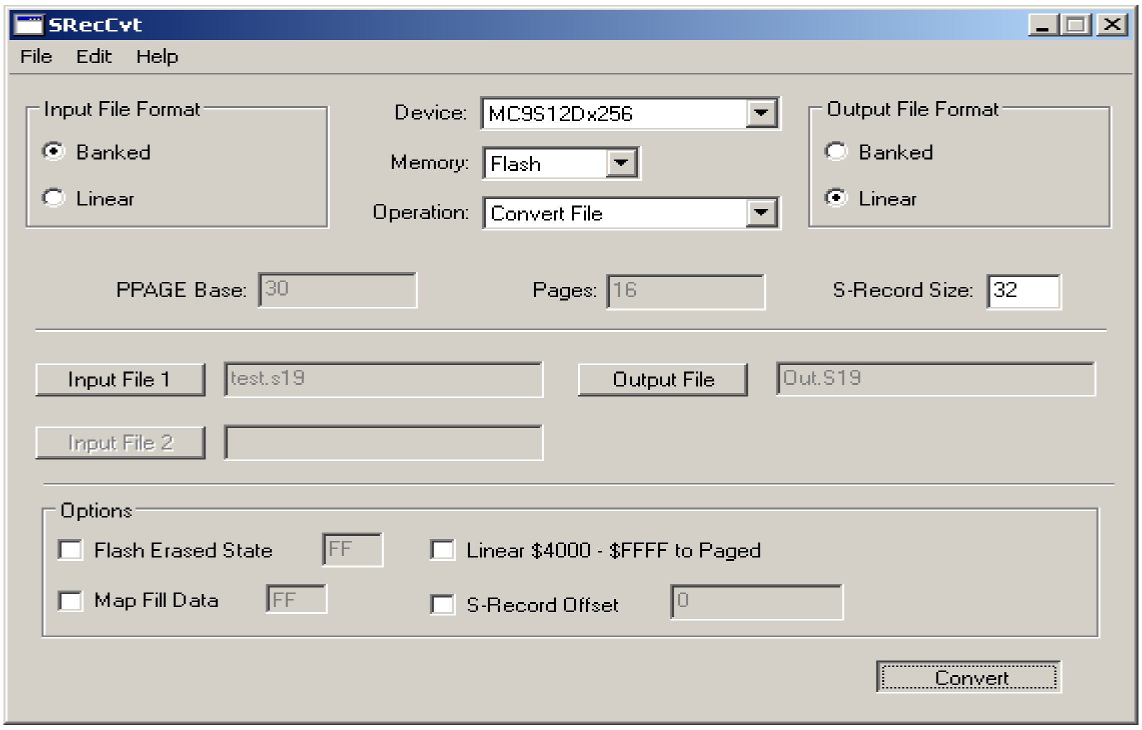
Input File 2:

Options

- Flash Erased State: FF
- Linear $4000 - $FFFF to Paged
- Map Fill Data: FF
- S-Record Offset: 0

Convert

Press on the Convert button to initiate conversion.


SRecCvt

File Edit Help

Input File Format: [✓ Banked] Linear

Device: MC9S12Dx256

Memory: Flash

Operation: Convert File

Output File Format: Banked [✓ Linear]

PPAGE Base: 30

Pages: 16

S-Record Size: 32

Input File 1: test.s19

Output File: Out.S19

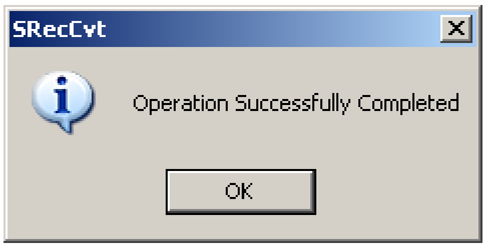
Input File 2:

Options

- Flash Erased State: FF
- Linear $4000 - $FFFF to Paged
- Map Fill Data: FF
- S-Record Offset: 0

Convert

After a successful conversion.

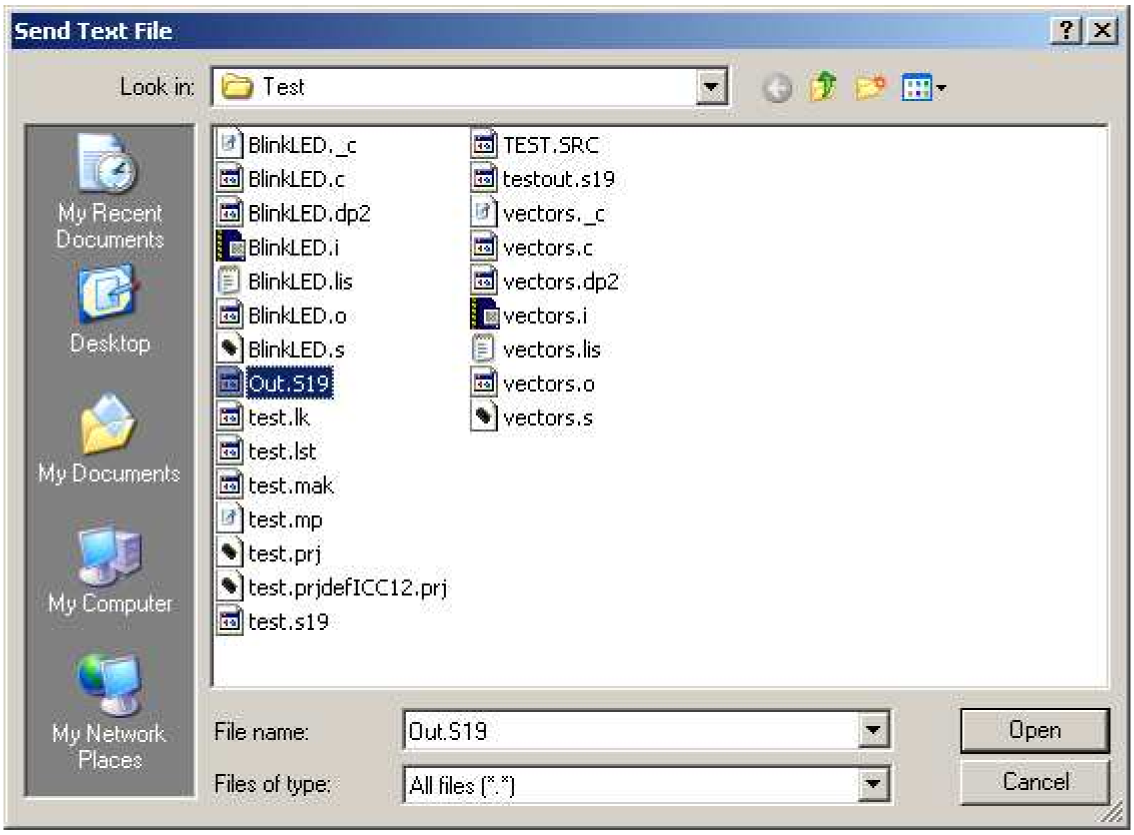

SRecCvt

Operation Successfully Completed

Below is the converted file.  Note that all lines, except for the S9, are even.

```
S2240C000034B7751B9ECC00006C1E2007EC1EC300016C1EEC1E8CFA0025F2B757300AFFFFF4
S2240F8000CF400016405587CE10008E100027056A000820F6CE405ACD10008E405A27061829
S2240F80200A307020F516402A20FE34B7751B9EC6FF7B025AC6FF7B02584A8000302010C690
S2240F8040FF7B02584A8000307902584A80003020EEB757303D1D0016073DFFFFFFFFFFFF77
S2240FEFE0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF4000D
B
S9030000FC
```

Select ***b*** to program and locate out.s19.  Double click on the file to initiate upload.


Send Text File

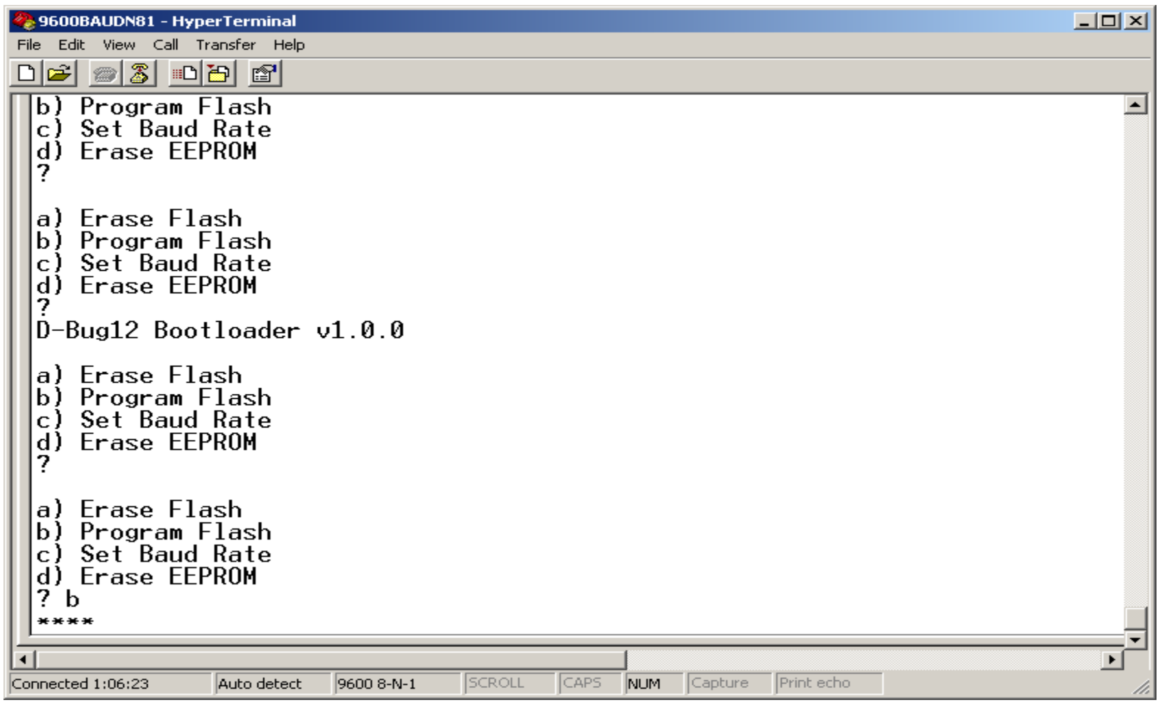
Look in: Test

BlinkLED._c
BlinkLED.c
BlinkLED.dp2
BlinkLED.i
BlinkLED.lis
BlinkLED.o
BlinkLED.s
Out.S19
test.lk
test.lst
test.mak
test.mp
test.prj
test.prjdefICC12.prj
test.s19
TEST.SRC
testout.s19
vectors._c
vectors.c
vectors.dp2
vectors.i
vectors.lis
vectors.o
vectors.s

File name: Out.S19

Files of type: All files (*.*)

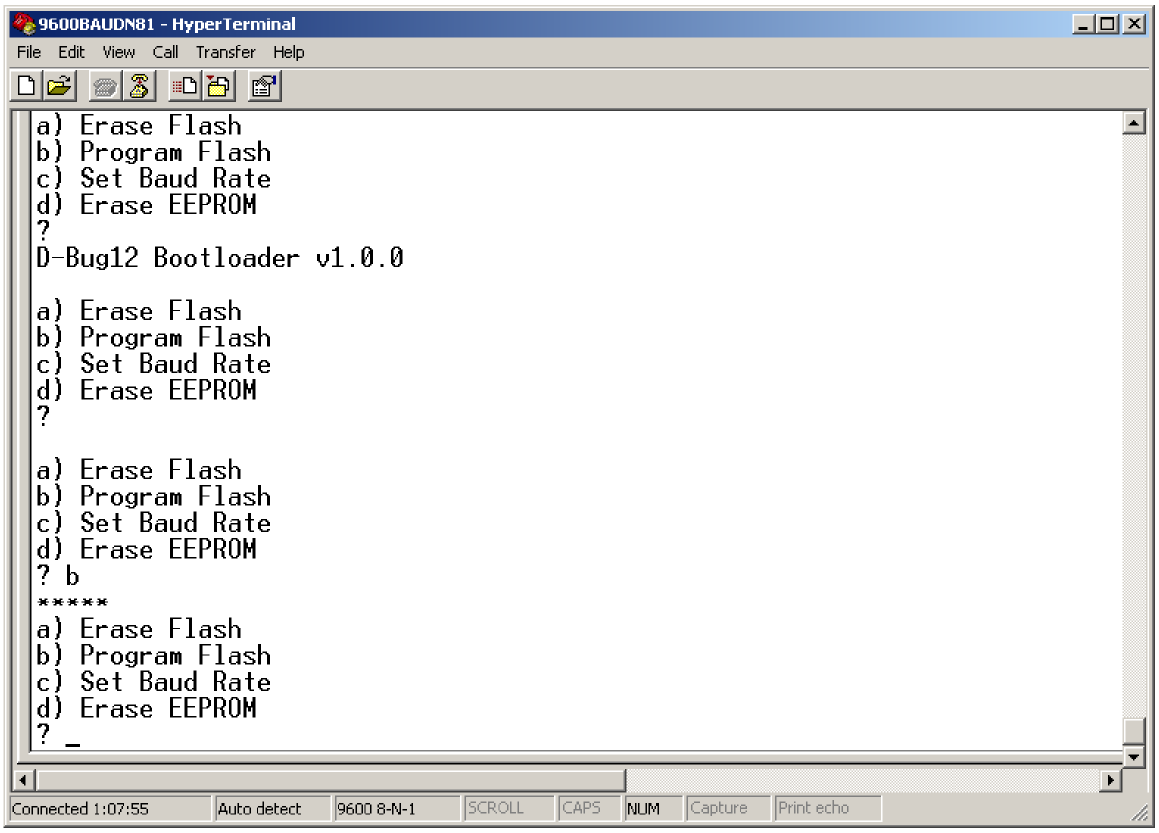
The characters ** will be seen to indicate that the file is being uploaded.


9600BAUDN81 - HyperTerminal
File Edit View Call Transfer Help

```
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? b
****
```

Connected 1:06:23 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

After the file is uploaded it will show the menu again.


9600BAUDN81 - HyperTerminal
File Edit View Call Transfer Help

```
a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?
D-Bug12 Bootloader v1.0.0

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? b
*****
a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
? _
```

Connected 1:07:55 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

After programming move the jumpers on JB3 to PAD1:PAD0 = 0:0 then preset the RESET button.

The LED on the Adapt9S12DP256 that is connected to PP7 will begin to blink.

This concludes the use of ICC12 with Adapt9S12DP256 to using the FLASH Loader.

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.