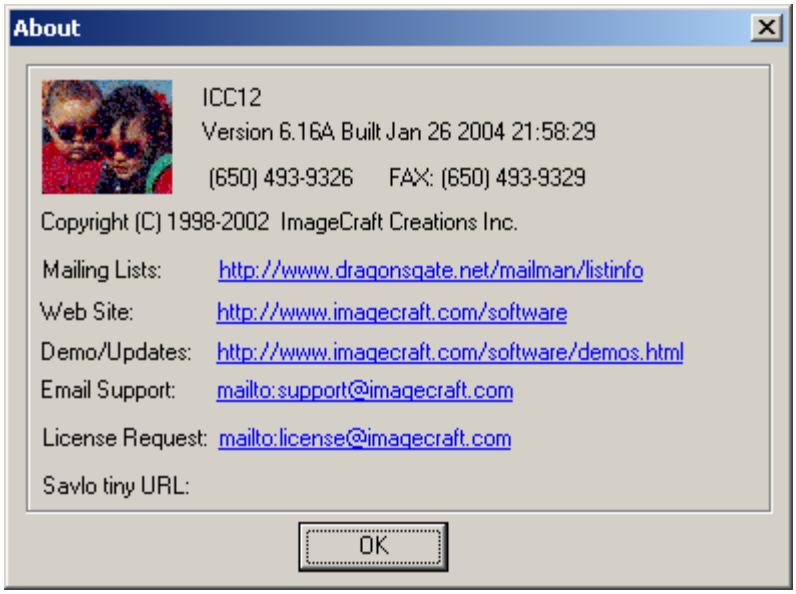
How to use ICC12 with Adapt9S12E128 and uBUG12

This document will demonstrate the use of ImageCraft ICC12 ***Version 6*** C compiler with Technological Arts' Adapt9S12E128 module. uBUG12 is used to erase and program flash memory after the compilation of a test program with ICC12. While other methods can be used to erase and program flash, this example will focus on uBUG12.

This document assumes that the user is already familiar with programming in C. It also assumes that the Serial Monitor has not been erased and is presently in the 9S12E128 MCU.

ImageCraft Links:



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:

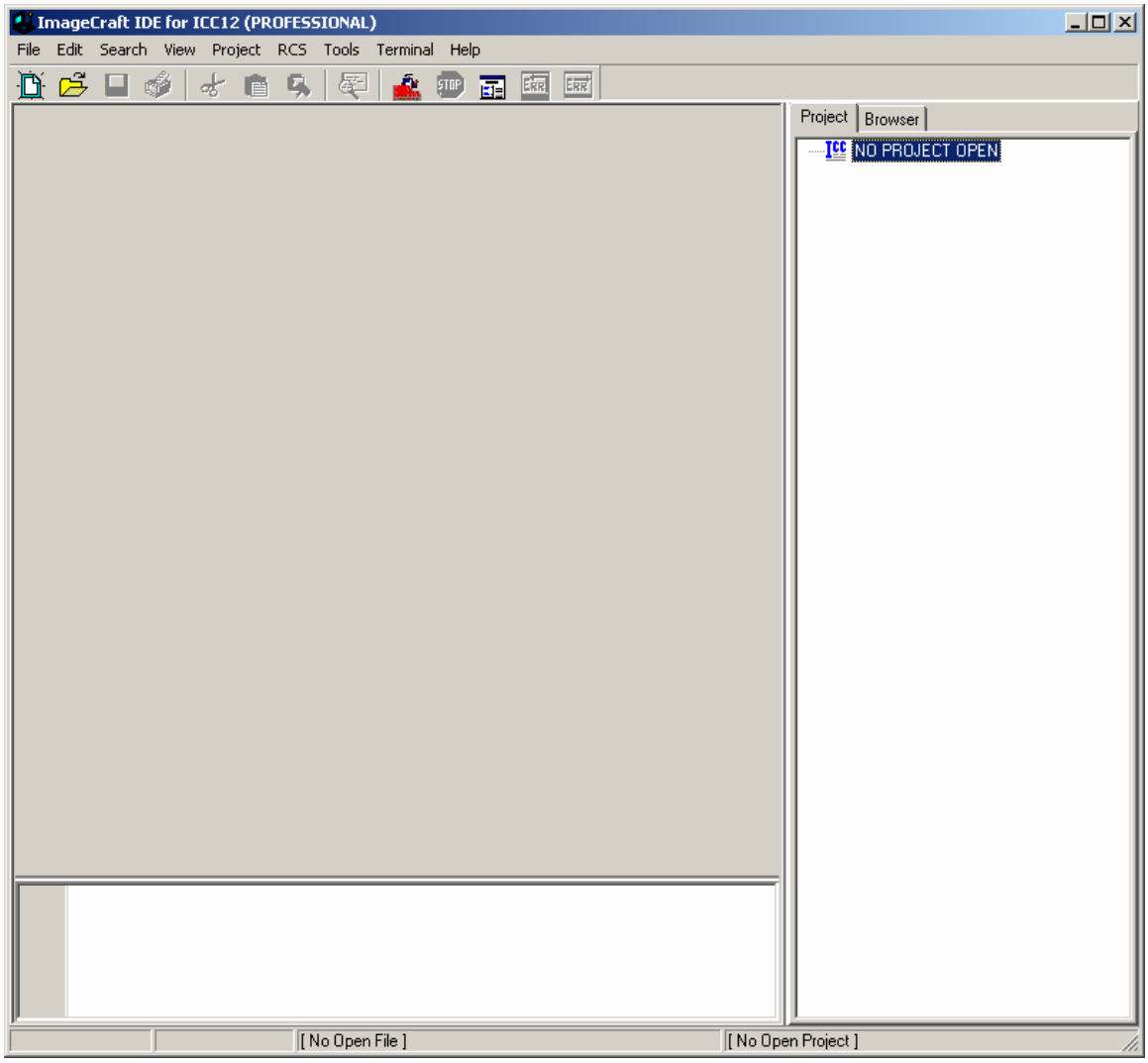
http://www.technologicalarts.com/myfiles/AD9S12E128.html
http://support.technologicalarts.ca/files/uBug12.zip
http://www.technologicalarts.com/myfiles/manuals/ad9s12e128man.pdf

Getting Started:

Double click on the ICC12 icon. If you have not read the ICC12 manual yet, and you just opened the IDE, you will no doubt wonder what to do next. Well wonder no more.

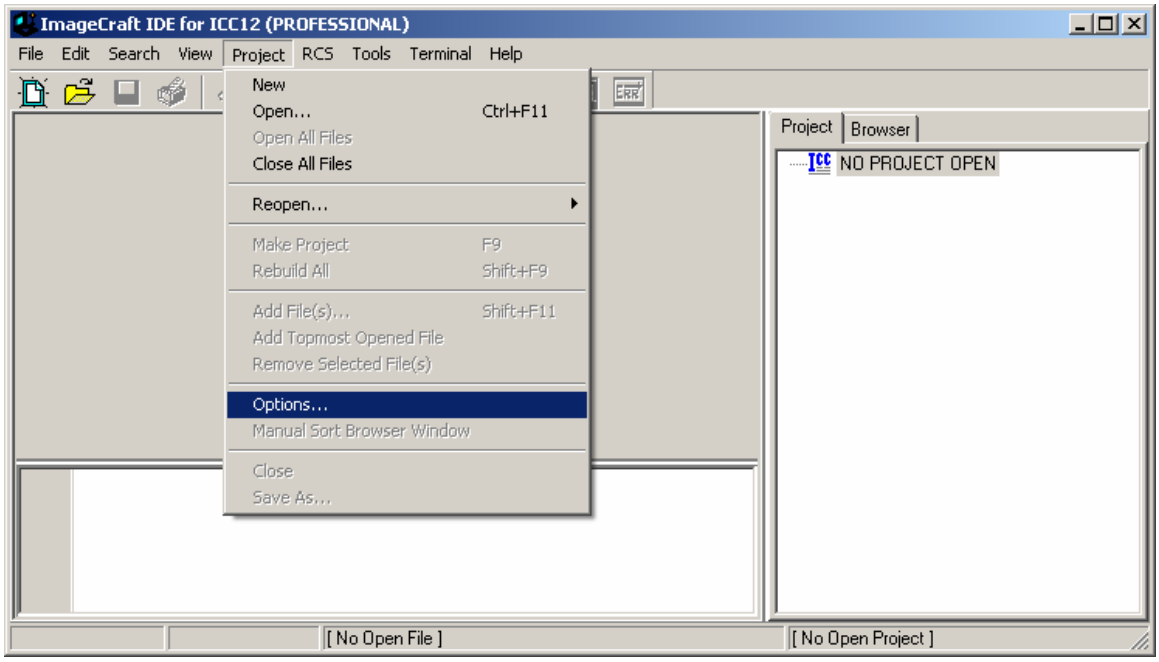
Note the three window panes. The top left pane is greyed out and the right pane displays the project window. The bottom left pane is where the error messages are displayed during compilation.

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 know where program memory, data memory, and registers are located during the linking process. In this example it is Adapt9S12E128.

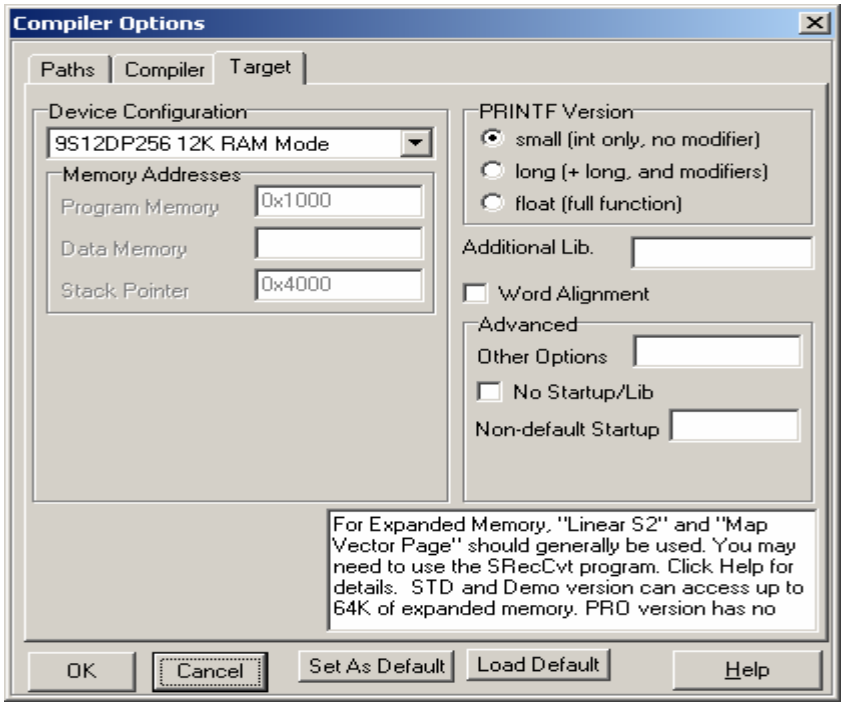


Compiler Setup:

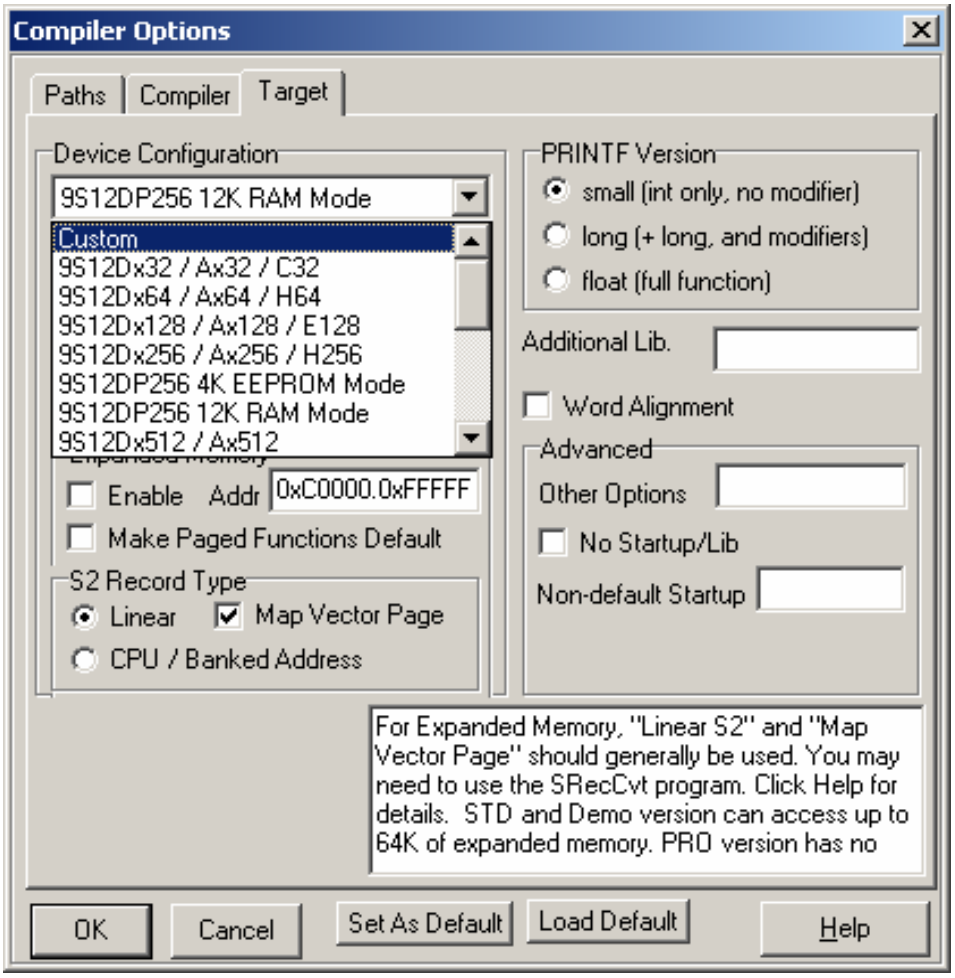
Click on Project Menu – Options – Target Tab.



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



Scroll up or down to select Custom as shown. You may notice that an E128 Device Configuration already exists. Unfortunately, the addresses are not setup properly for using with the Serial Monitor. Therefore Custom configuration must be selected and the memory parameters must be.

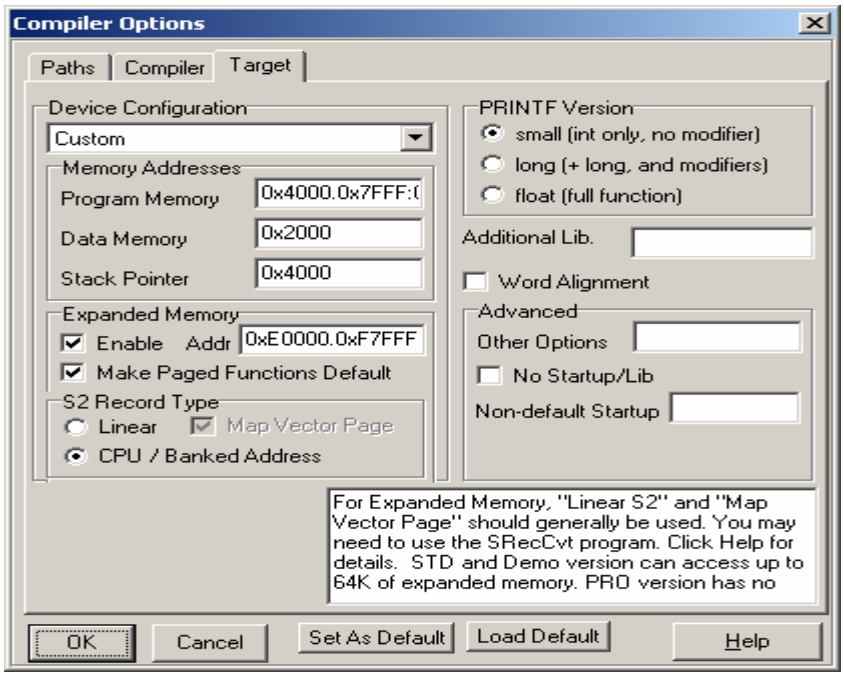


Custom Device Configuration:

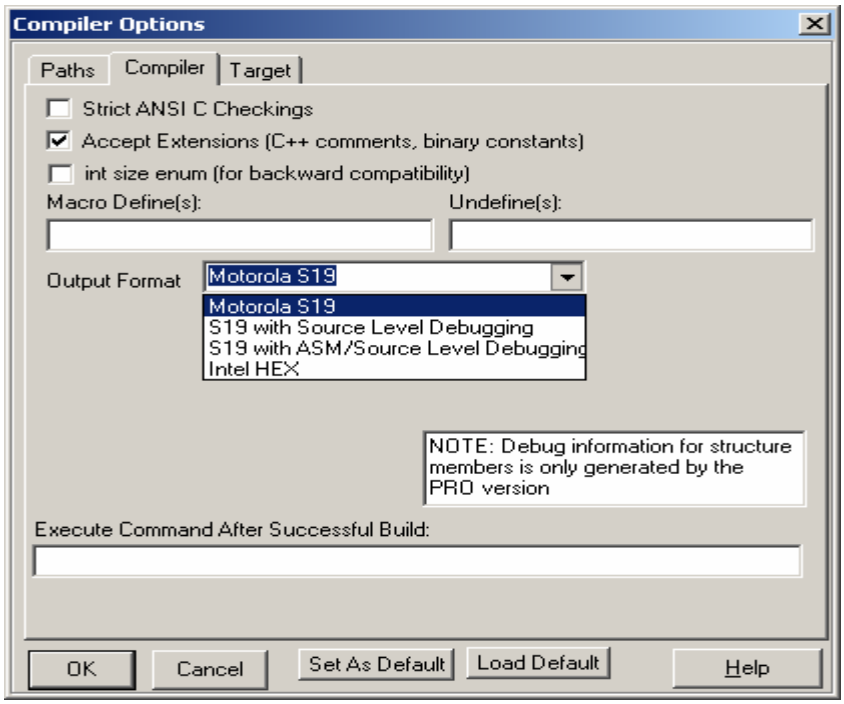
Program Memory: ***0x4000.0x7FFF:0xC000.0xFFFF***
Data Memory: ***0x2000***
Stack Pointer: ***0x4000***

Expanded Memory:

Note the address range is ***0xE0000. 0xF7FFF***. That implies that the valid PPAGE range is from $38 to $3D. PPAGE $3E and $3F correspond to the fixed memory area, and are allocated to ***0x4000.0x7FFF:0xC000.0xFFFF***

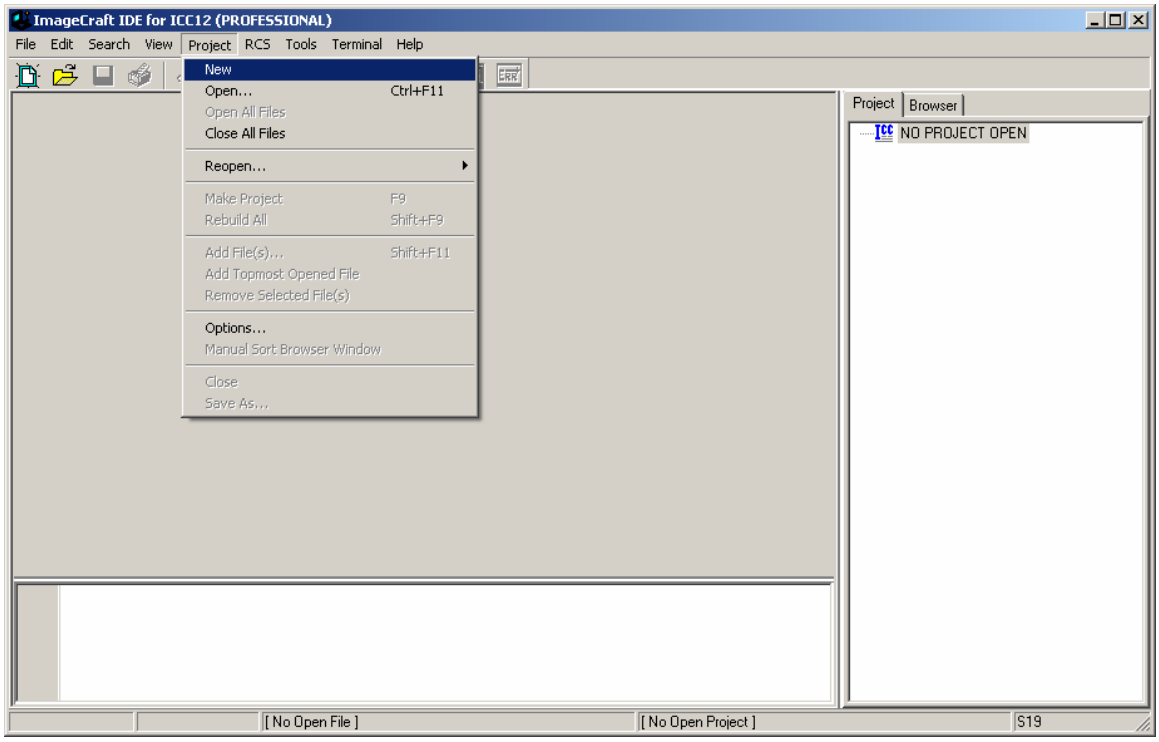


On the compiler tab there are several choices of S-record output as shown. Select one that suits you.

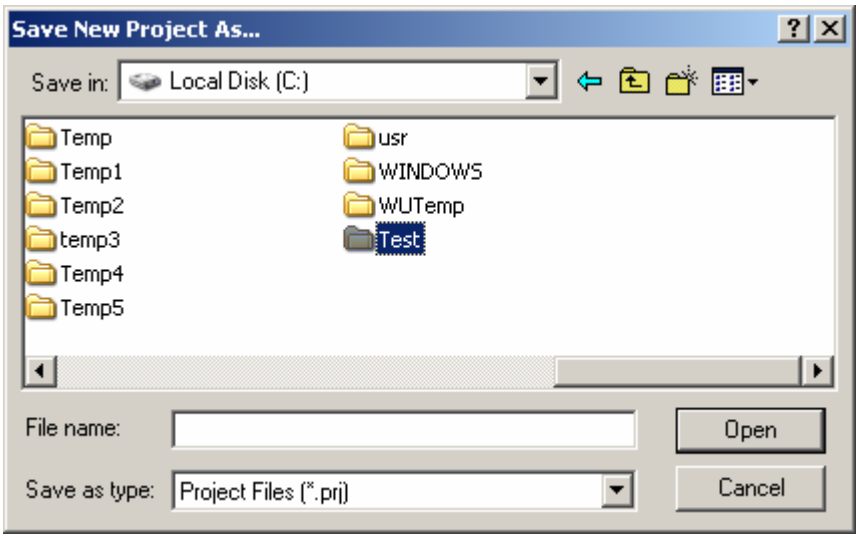


Starting a new Project:

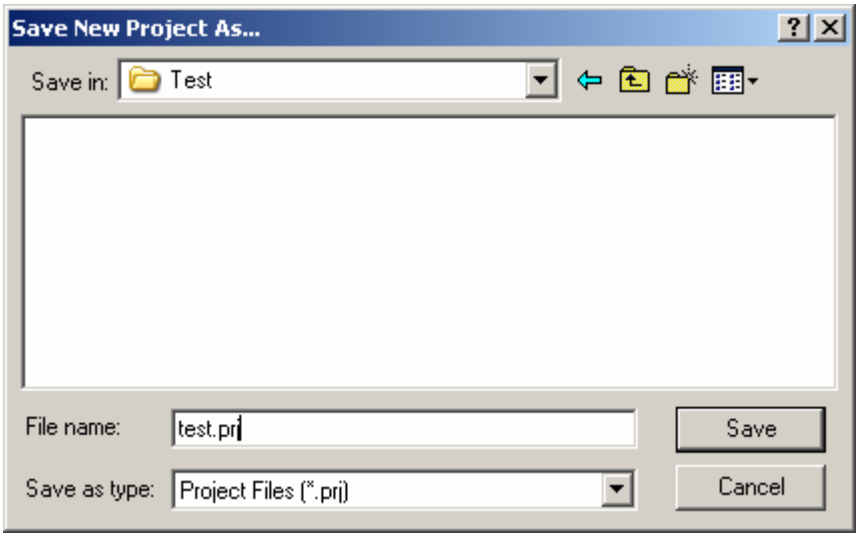
Once the compiler options are setup, a new project can be created. Click Project menu – New.



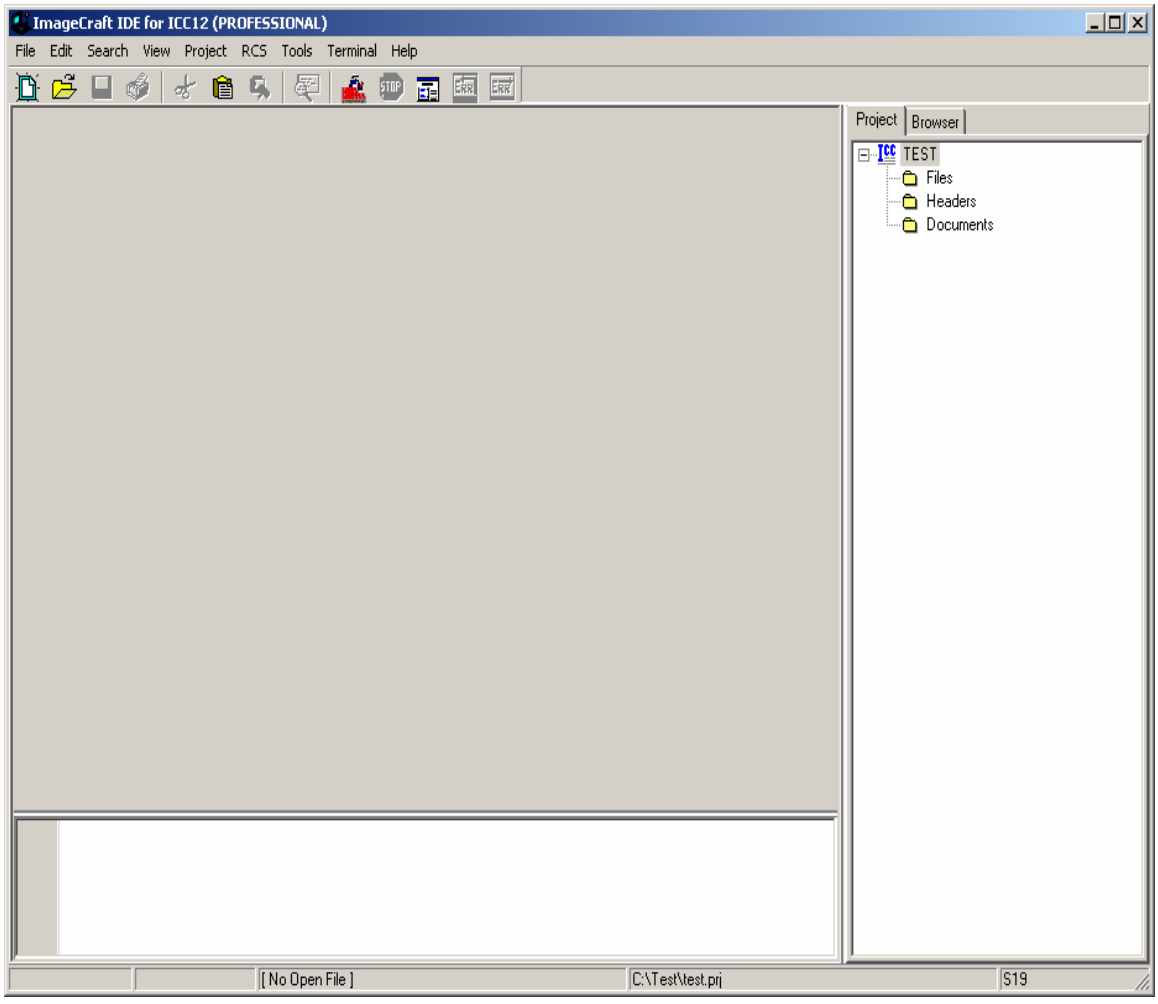
ICC12 will prompt to save the new project. You'll need to decide whether to create a new directory to save the new project. In this example a new directory called ***Test*** was created and the file was saved as ***test.prj***.



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

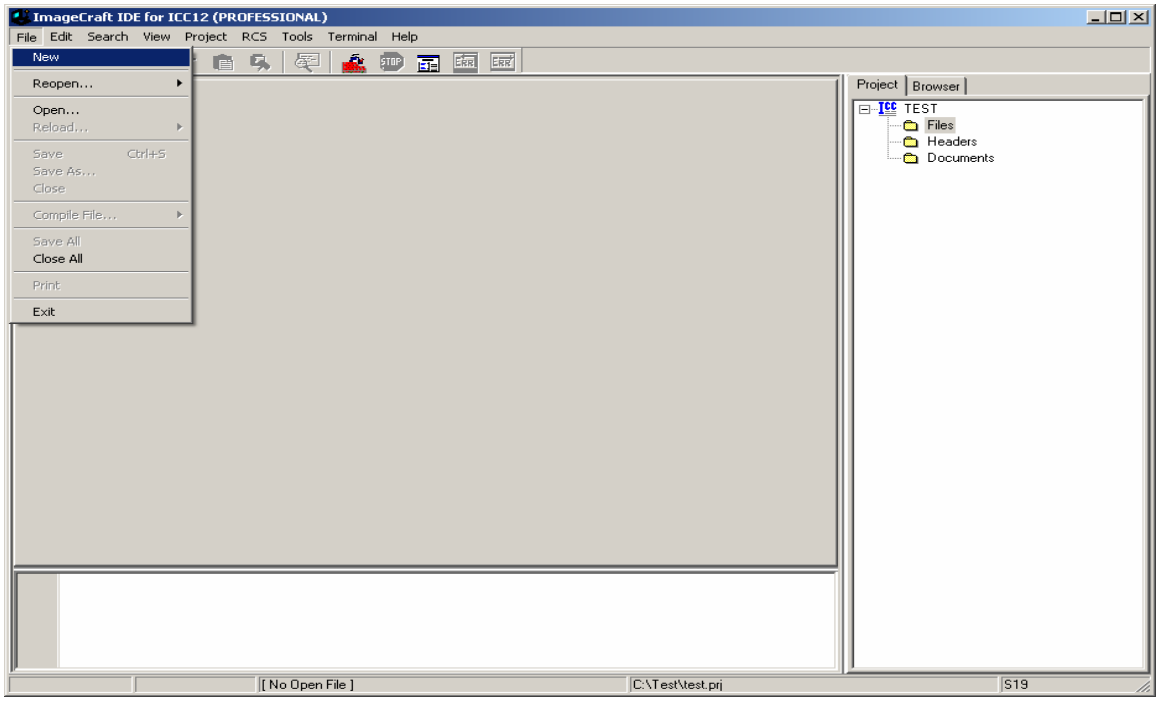


Note that the project window has changed to add Files, Headers and Documents.

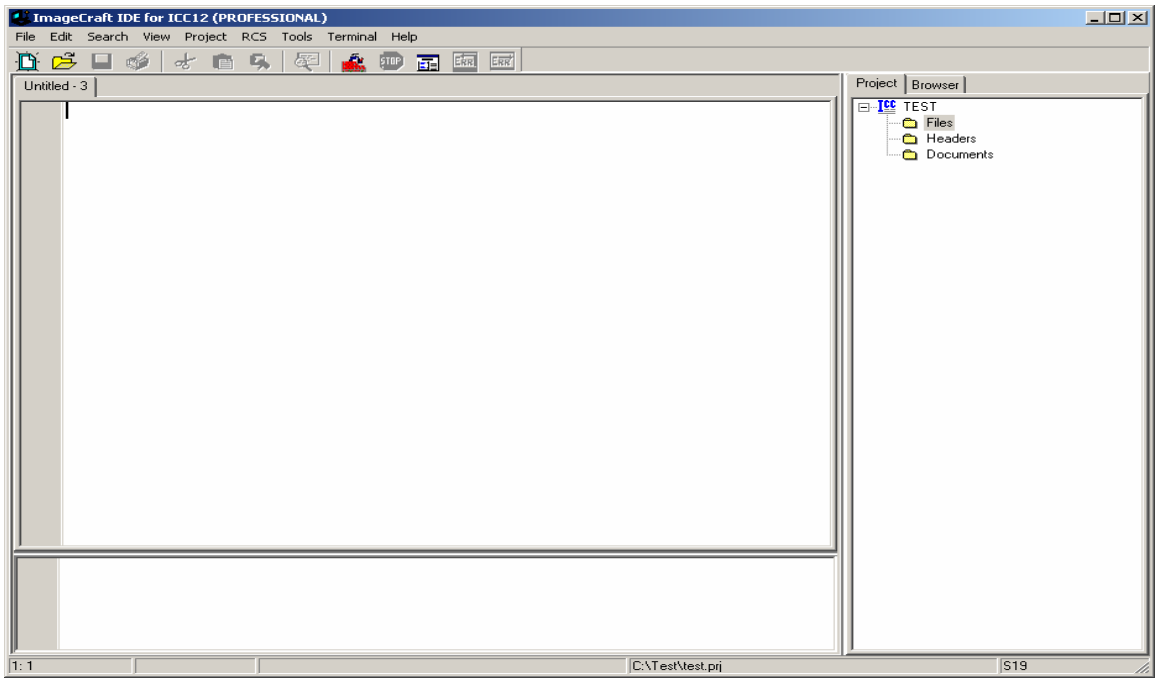


Adding new files to the project:

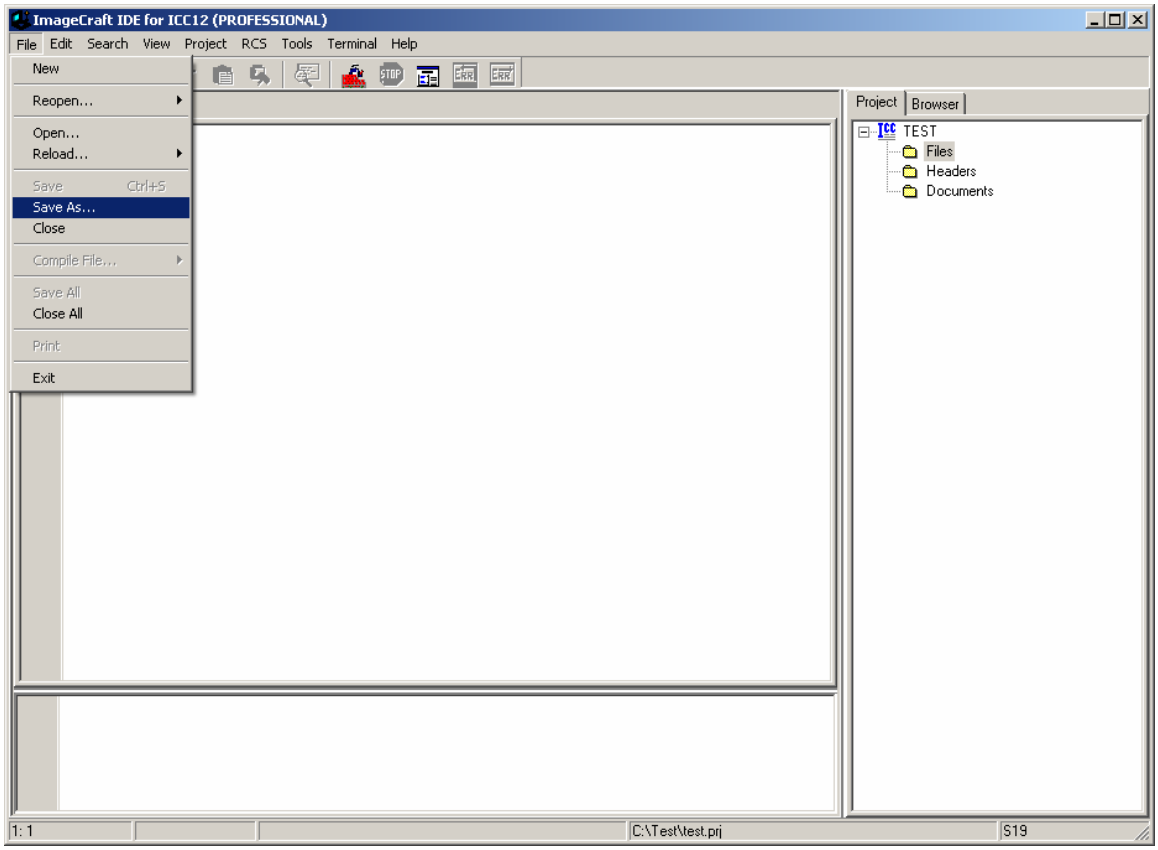
To add files to the project, click on the File menu and select New, as shown.



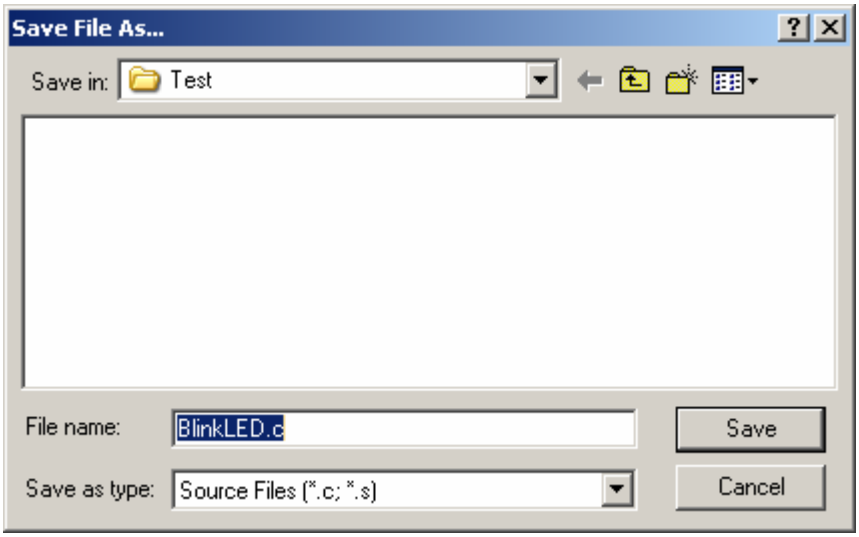
Note that ICC12 created an untitled file. Save the file as ***BlinkLED.C***.



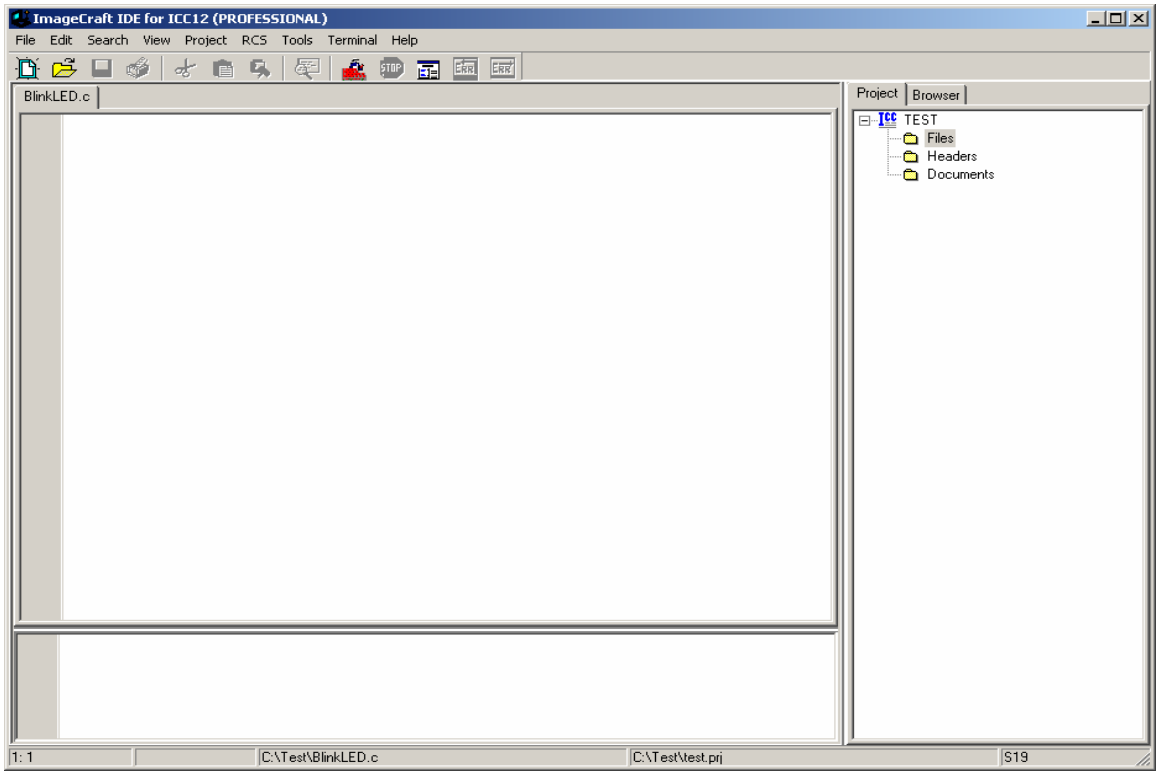
To save, click on File menu – Save As



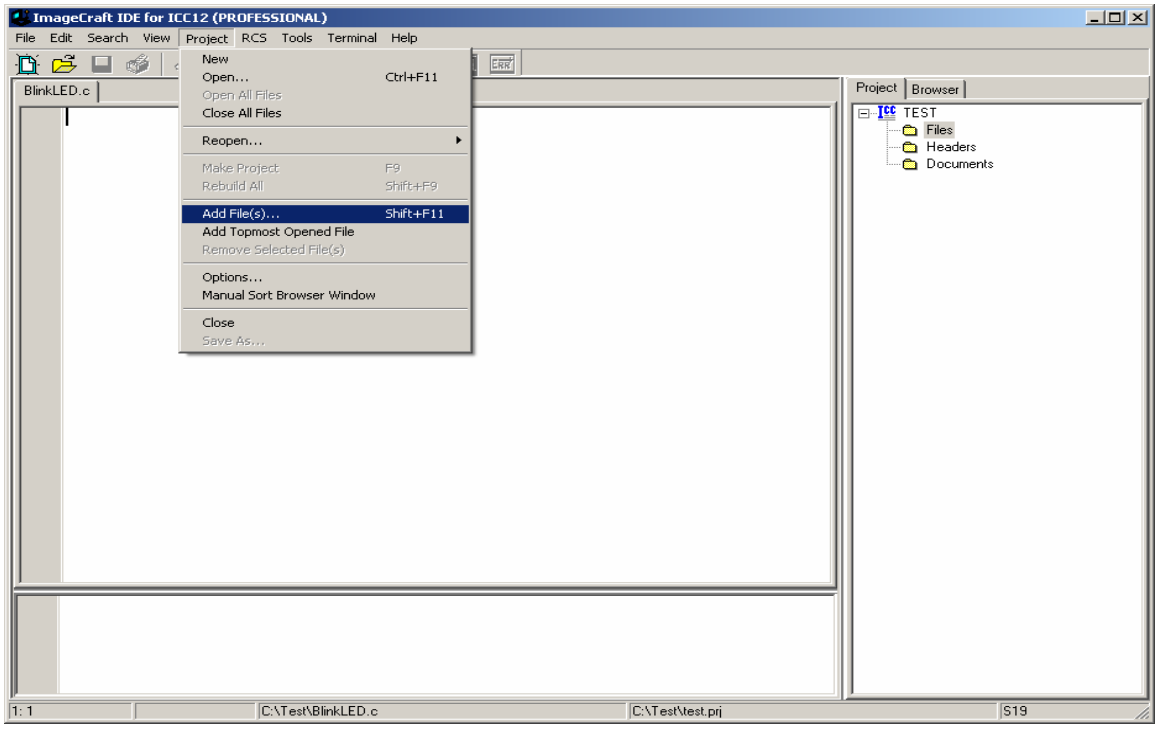
ICC12 will open an explorer window to help save the file. Type BlinkLED.c then press the save button.



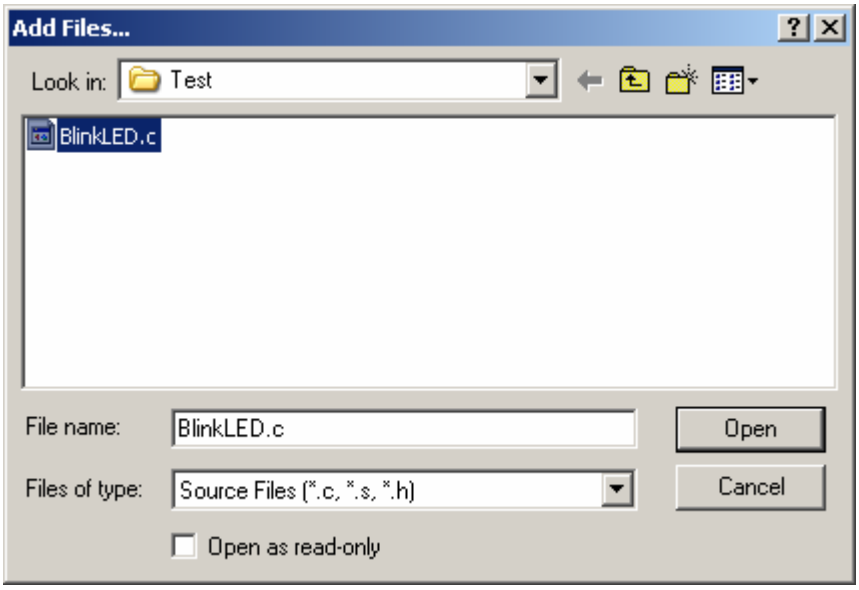
Now you'll see that ICC12 has renamed the file to BlinkLED.c.



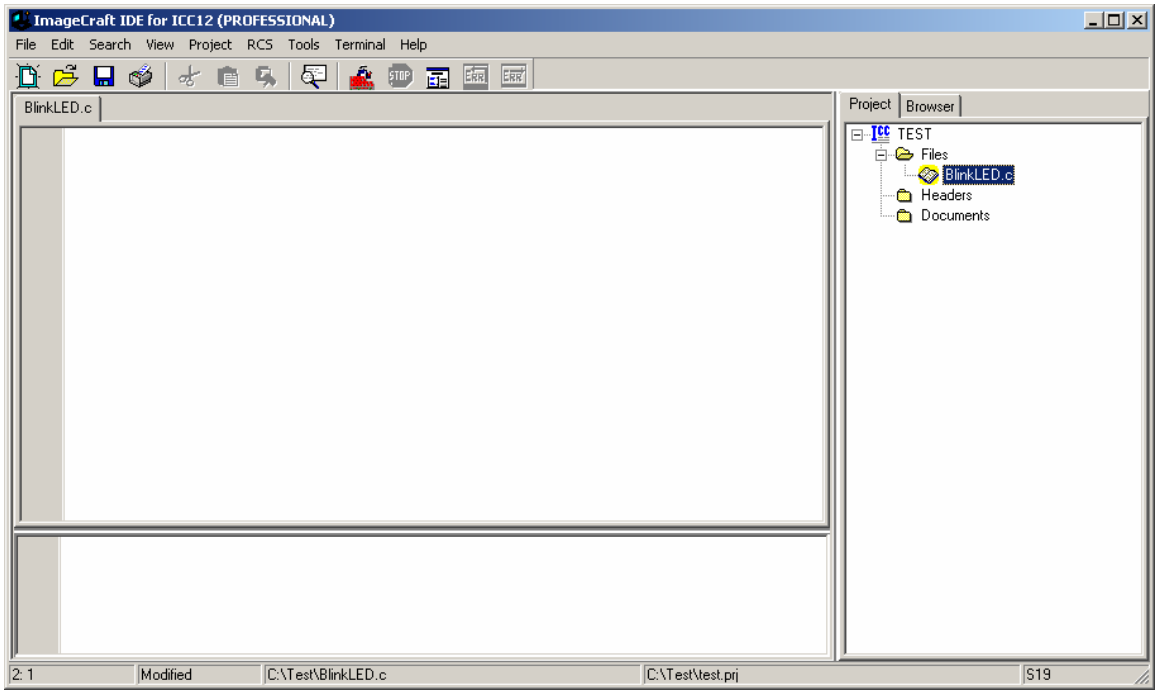
To add BlinkLED.c to the Project, click on the Project menu and select Add File(s)



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

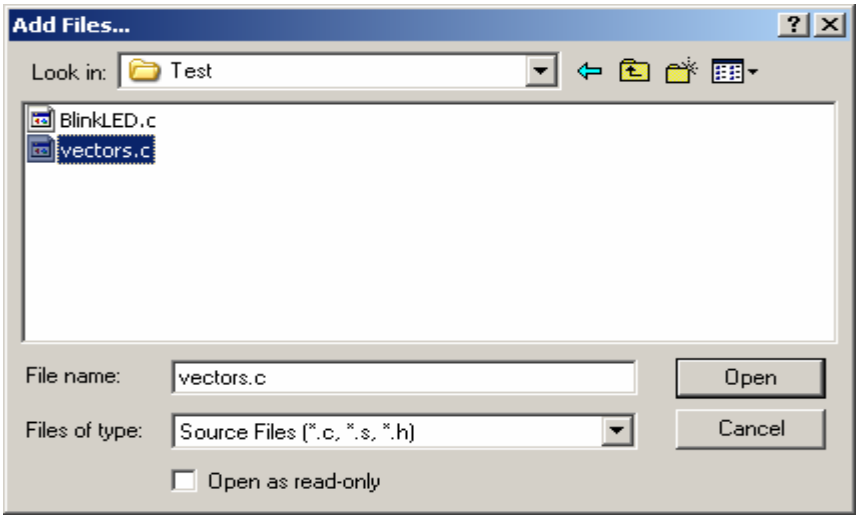


You'll notice that the right window pane has changed to include BlinkLED.c in the Project Files list.

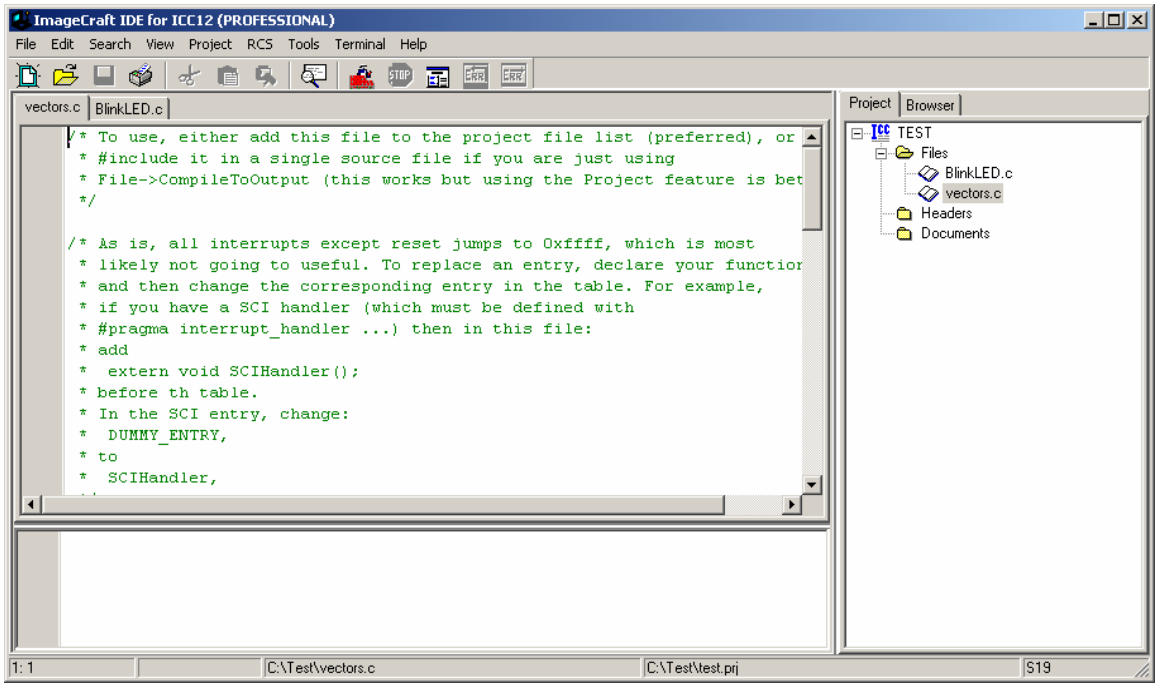


Locate ***vectors.c*** and copy it to Test directory. The main reason to do this is project dependencies. It is not good to keep editing a global ***vectors.c*** if other projects are using this same file. It becomes a problem to keep track of the changes made to the different projects.

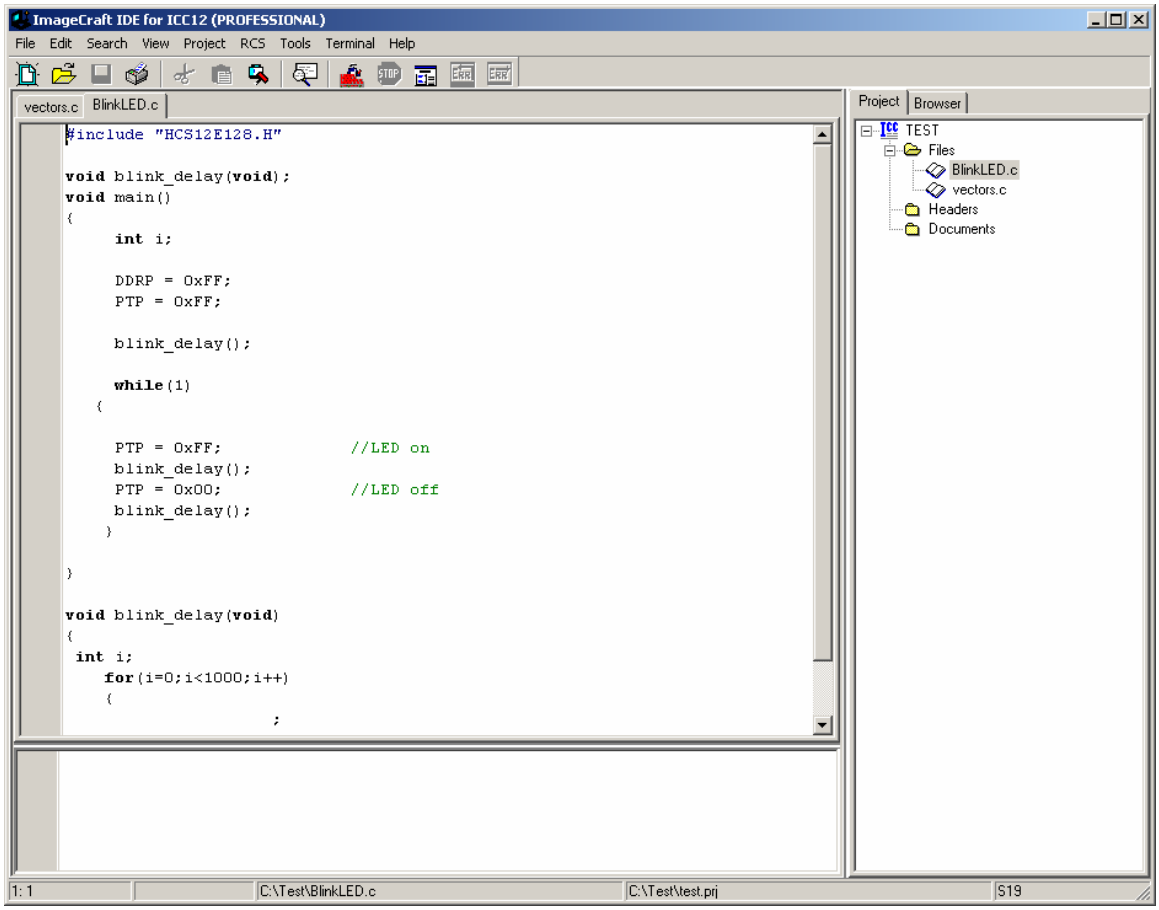
To add ***vectors.c*** to the Project, click on the Project menu – Add File(s)



Note that the Project Files list has updated to include ***vectors.c*** It is important to note that the default **vectors.c** included with ICC12 was written for the 68HC912B32 and 812A4 MCUs. Technically, you should edit the file to include other interrupt service routine (ISR) addresses specific to the 9S12E128. For this example, though, we'll use the file "as is".



Enter the lines of program code shown below into the BlinkLED.c file. The next step is to compile/make/build the code. Note: ***HCS12E128.H*** can be found at the end of this document.

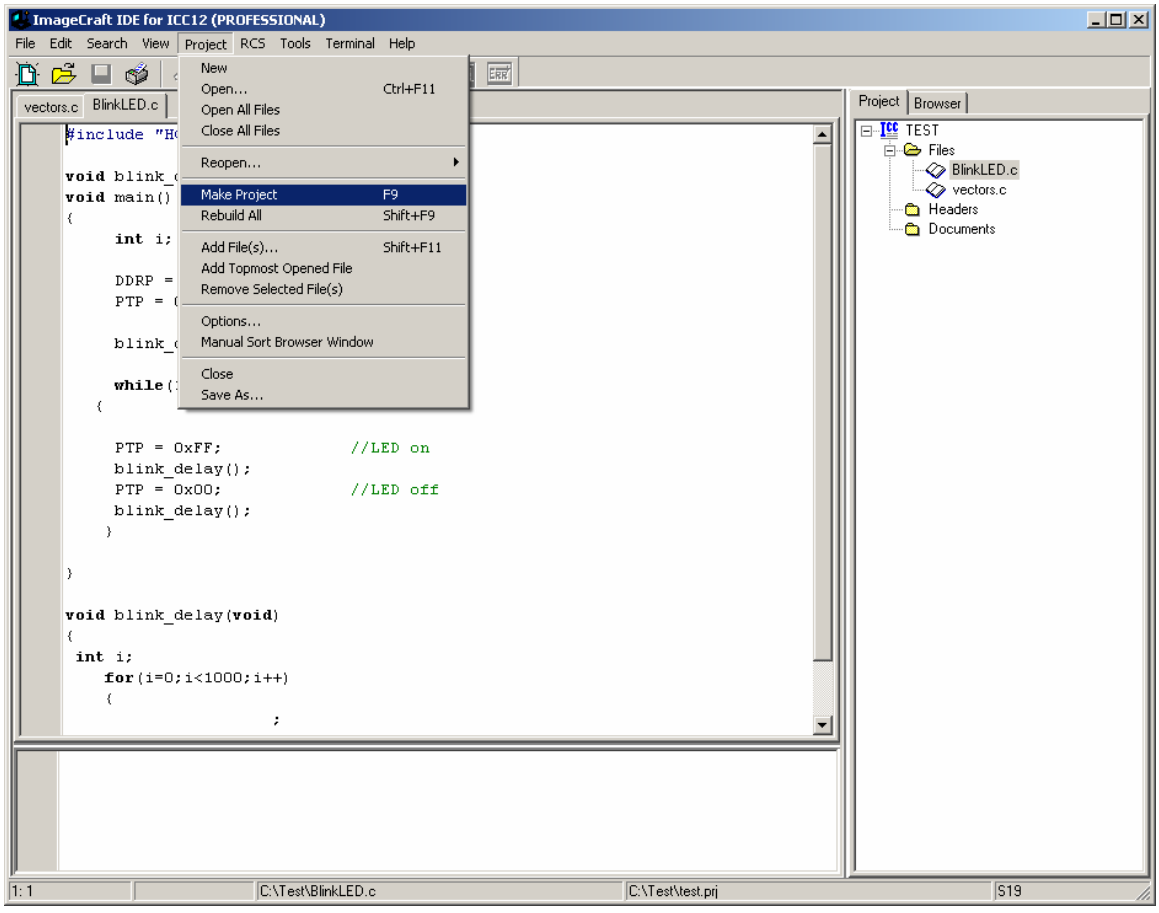



```
#include "HCS12E128.H"

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();
       }
```

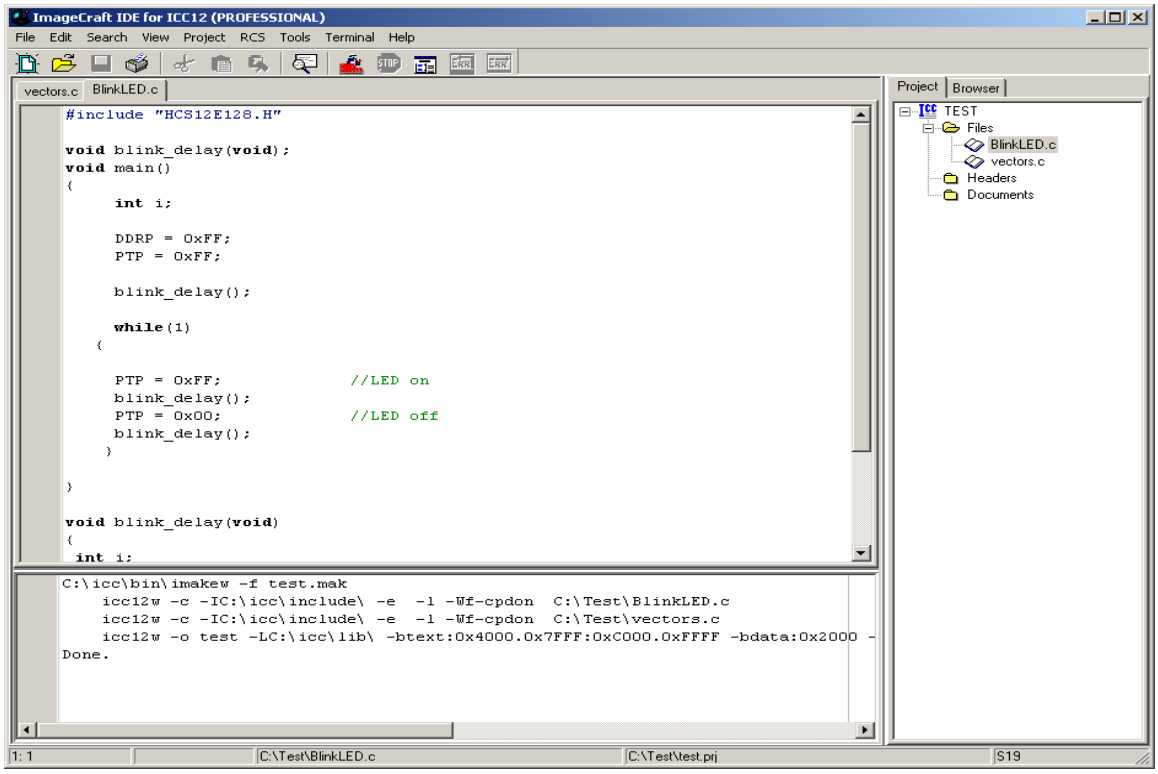
```
}

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

Compile/Build/Make the file:

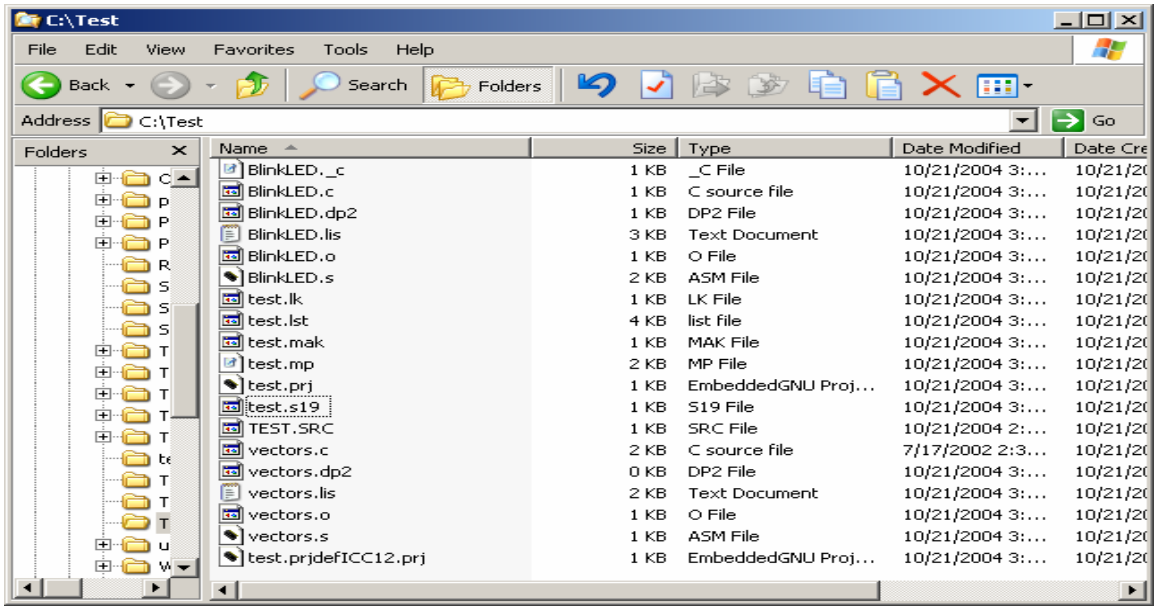
To make the file, click Project menu and select Make Project, as shown.



You'll notice that the bottom window pane shows messages to display how the build progressed. Any errors will be shown in this window. In this case, the build completed without error, so we can move on to erasing and programming the 9S12E128.



Note in passing that some other intermediate files are created during a Make.



Use WordPad to open and inspect the content of ***test.s19***

S10E4000CF3FFF16405587CE20008EF6
S110400B200027056A000820F6CE405ACD9B
S111401820008E405A2706180A307020F51634
S1074026402A20FE0A
S110402A34B7751B9EC6FF7B025AC6FF7B90
S110403702584A8000382010C6FF7B025852
S10D40444A8000387902584A8000CF
S10A404E3820EEB757303DA6
S21138800034B7751B9ECC00006C1EEC1EC3FA
S21238800D00016C1EEC1E8C03E82DF2B75730BF
S20538801B0A1D
S111FFD0FFFFFFFFFFFFFFFFFFFFFFFFFF2D
S111FFDEFFFFFFFFFFFFFFFFFFFFFFFFFF1F
S111FFECFFFFFFFFFFFFFFFFFFFFFFFFFF11
S109FFFAFFFFFFFF4000C1
S10840551D0016073DEB
S9034000BC

If you look closely at the S-record you'll see a mixture of S1 and S2 lines. This is a typical file of S-records generated by ICC12. S1 records are programmed in the ***0x4000 – 0x7FFF*** and ***0xC000 – 0xFFFF*** memory blocks. ISRs are always placed in the fixed memory region. An ISR can call any routine inside a PPAGE when necessary. S2 records can also reference fixed memory regions, but are typically paged by ICC12

Below is the vector address as S1 record.

S111FFD0FFFFFFFFFFFFFFFFFFFFFFFFFF2D
S111FFDEFFFFFFFFFFFFFFFFFFFFFFFFFF1F
S111FFECFFFFFFFFFFFFFFFFFFFFFFFFFF11
S109FFFAFFFFFFFF4000C1

Note the address at 0xFFFE|0xFFFF contains 0x4000, telling the MCU where to start executing code following power up or RESET.

S109FFFAFFFFFFFF***4000***C1

The S-record below shows the actual first few bytes of code in the program

S10E***4000***CF3FFF16405587CE20008EF6

ICC12 has generated a banked S2 record.

S211***38***800034B7751B9ECC00006C1EEC1EC3FA

PPAGE = 0x38

S21138***8000***34B7751B9ECC00006C1EEC1EC3FA

Memory address = 0x8000.

Programming:

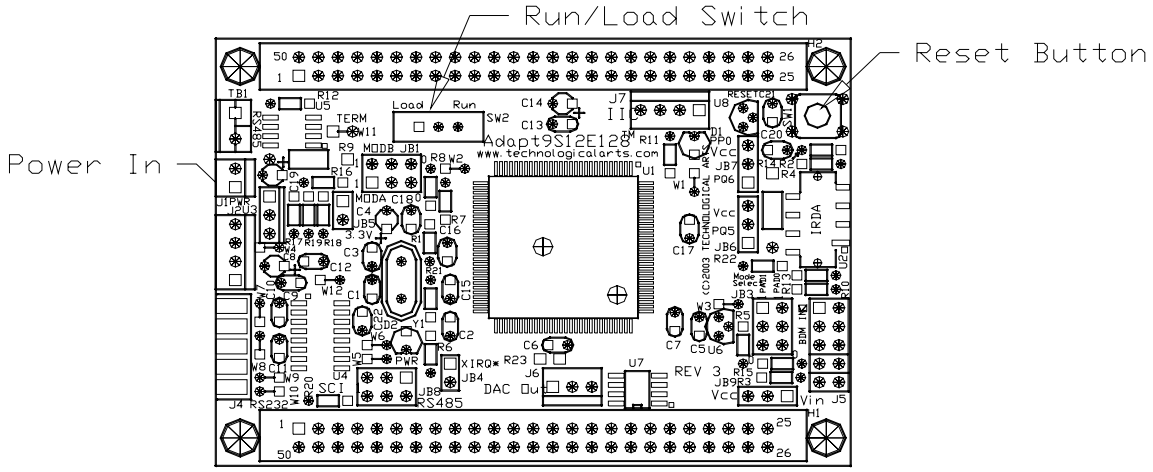
This document assumes that the Serial monitor is present on Adapt9S12E128. If not, you'll need to loda it into flash using a BDM pod. That procedure is beyond the scope of this document.

Find uBug12 on the CD that came with your evaluation or demo package. If you don't have it, you can download uBUG12 from Technological Arts, at http://support.technologicalarts.ca/files/uBug12.zip

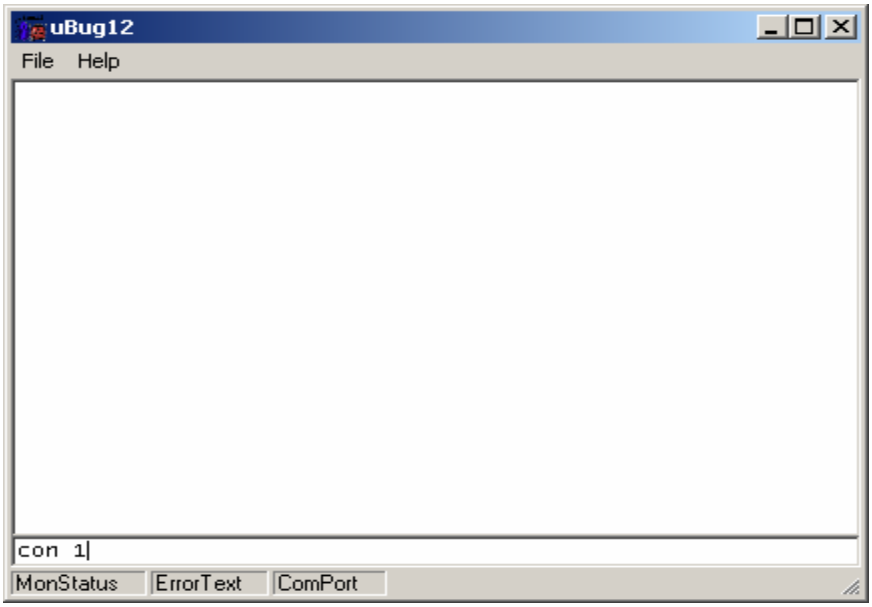
For Windows98 users the .NET framework must be installed before running uBUG12. The Microsoft site link is

http://www.microsoft.com/downloads/details.aspx?FamilyID=d7158dee-a83f-4e21-b05a-009d06457787&displaylang=en

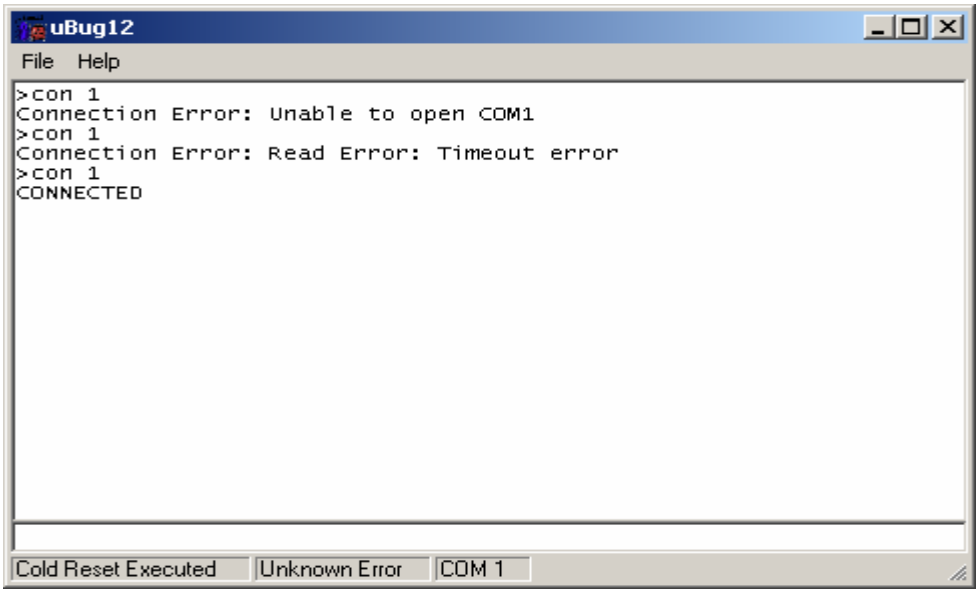
After installing uBug12, slide the Run/Load switch on Adapt9S12E128 to the Load position, and apply power to the board.



Double click on the uBUG12 icon to launch it.

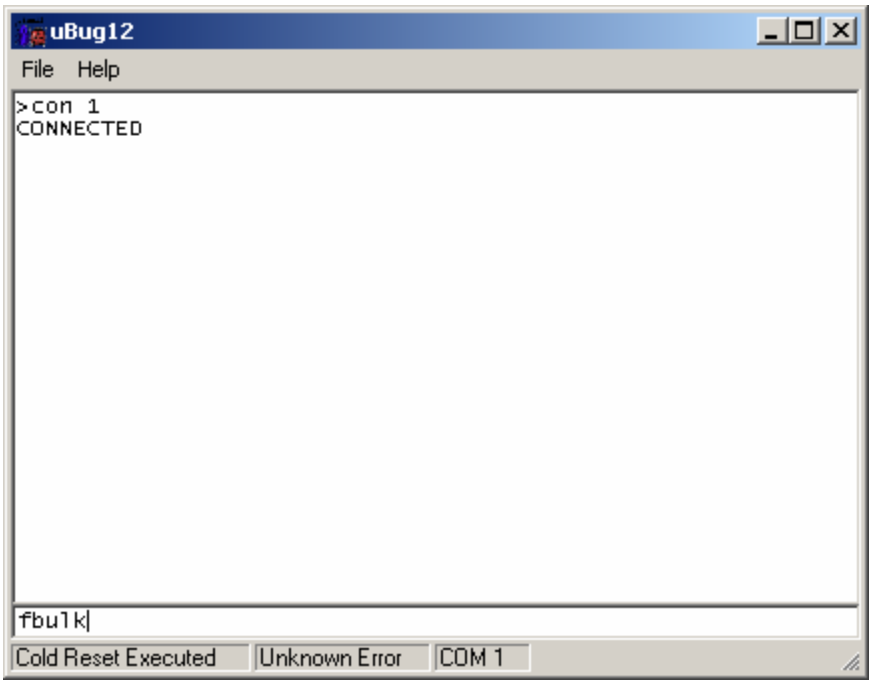


In the command bar type ***con 1*** to connect to COM 1 (or ***con 2*** if you're using COM 2). A ***CONNECTED*** message will appear to indicate that a connection between your PC and Adapt9S12E128 has been established.

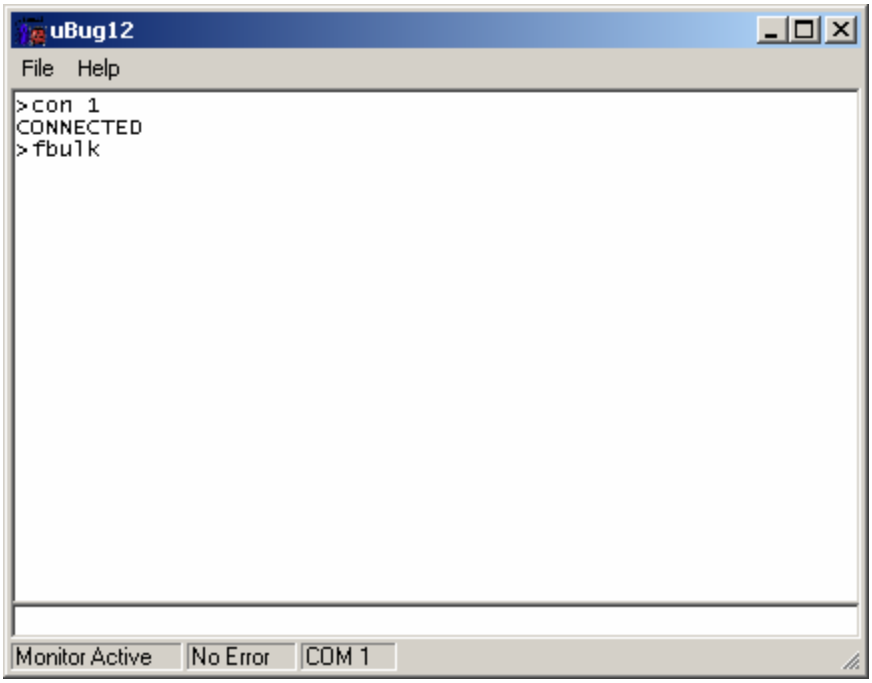


Two possible errors can occur:
Connection Error: Unable to open COM1 <- Another application is using the COM port

Connection Error: Read Error: Timeout error <- The MCU is not currently in LOAD mode, not powered up, or the cable is disconnected from either the PC or the E128 board.



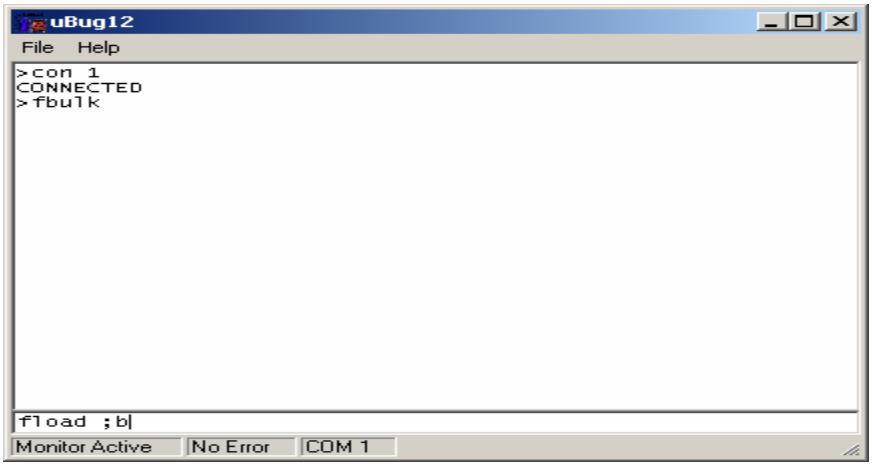
Now erase the flash memory by typing the command ***FBULK***.

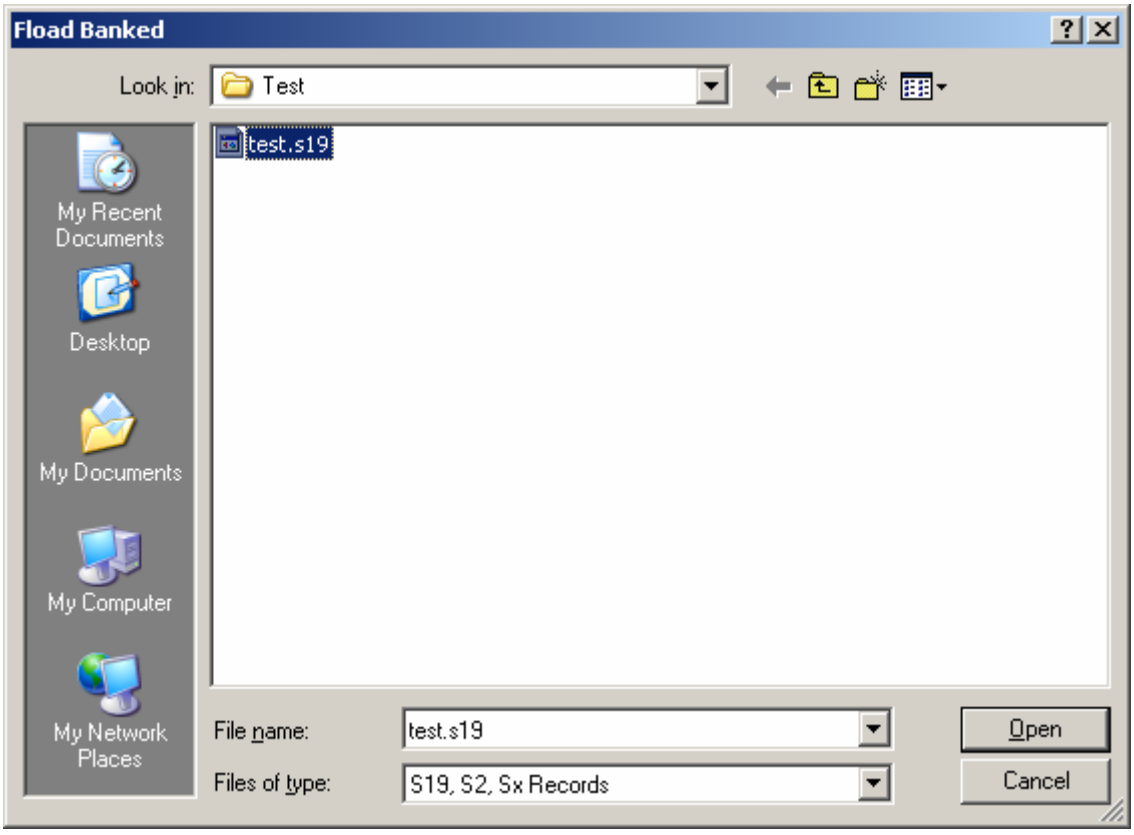


To load your program, type the command ***FLOAD ;B*** for banked S19, S2, SX and formatted S19 (i.e. went thru SrecCVT program) records. To load a file containing non-banked S2 records, the command is ***FLOAD***.

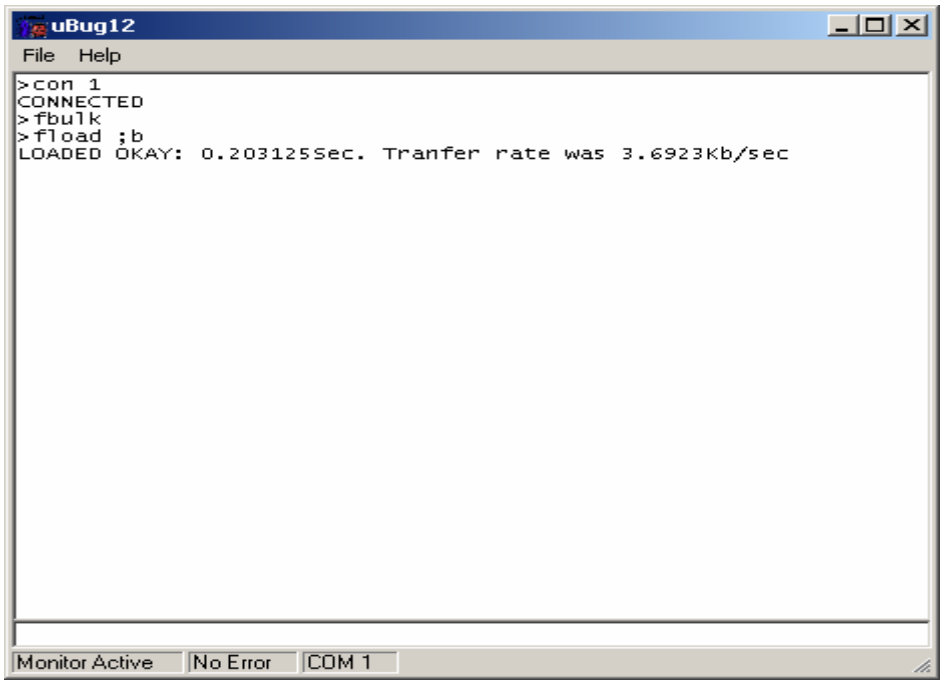
Uploading Banked S-record:

The command to upload banked S-records is ***FLOAD ;B***. It is important to include the ***;B*** option to let uBUG12 know that the S-record is banked. Make sure you become familiar with the differences between S19, SX, S2. See Appendix A for an explanation of S-records.





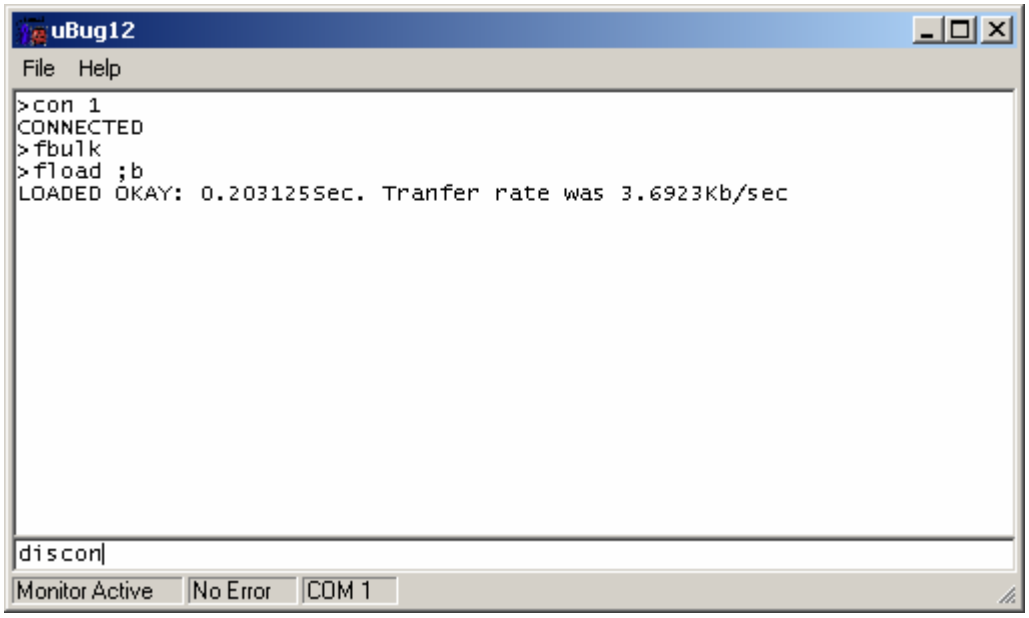
Double click on the file to initiate upload.



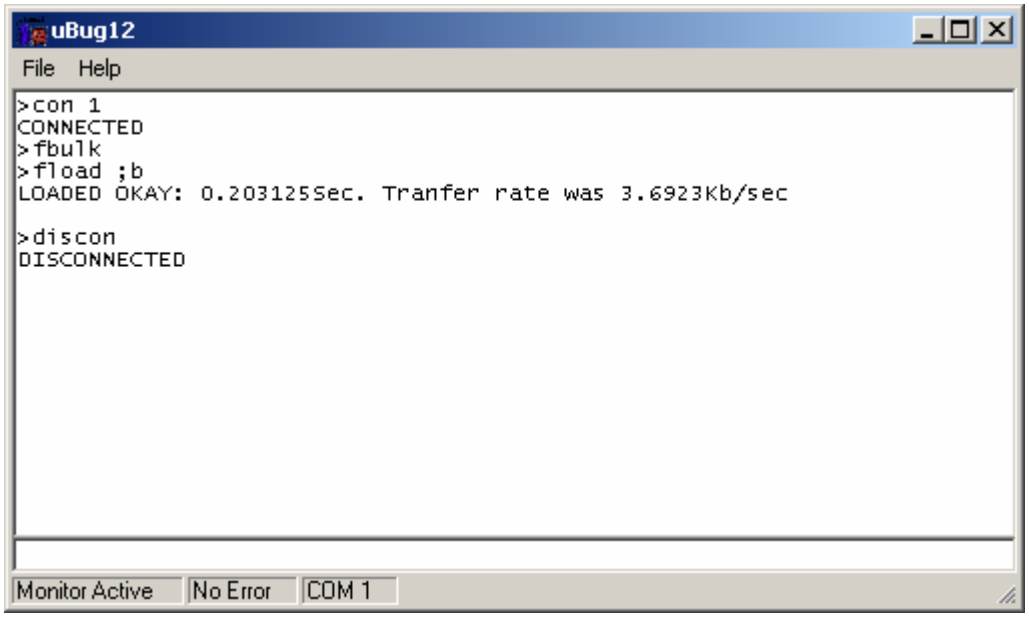
A good upload will show ***LOADED OKAY*** messages.

After successful programming, slide the Run/Load switch to Run and press the reset button. The application will begin blinking the LED connected to port pin PP0.

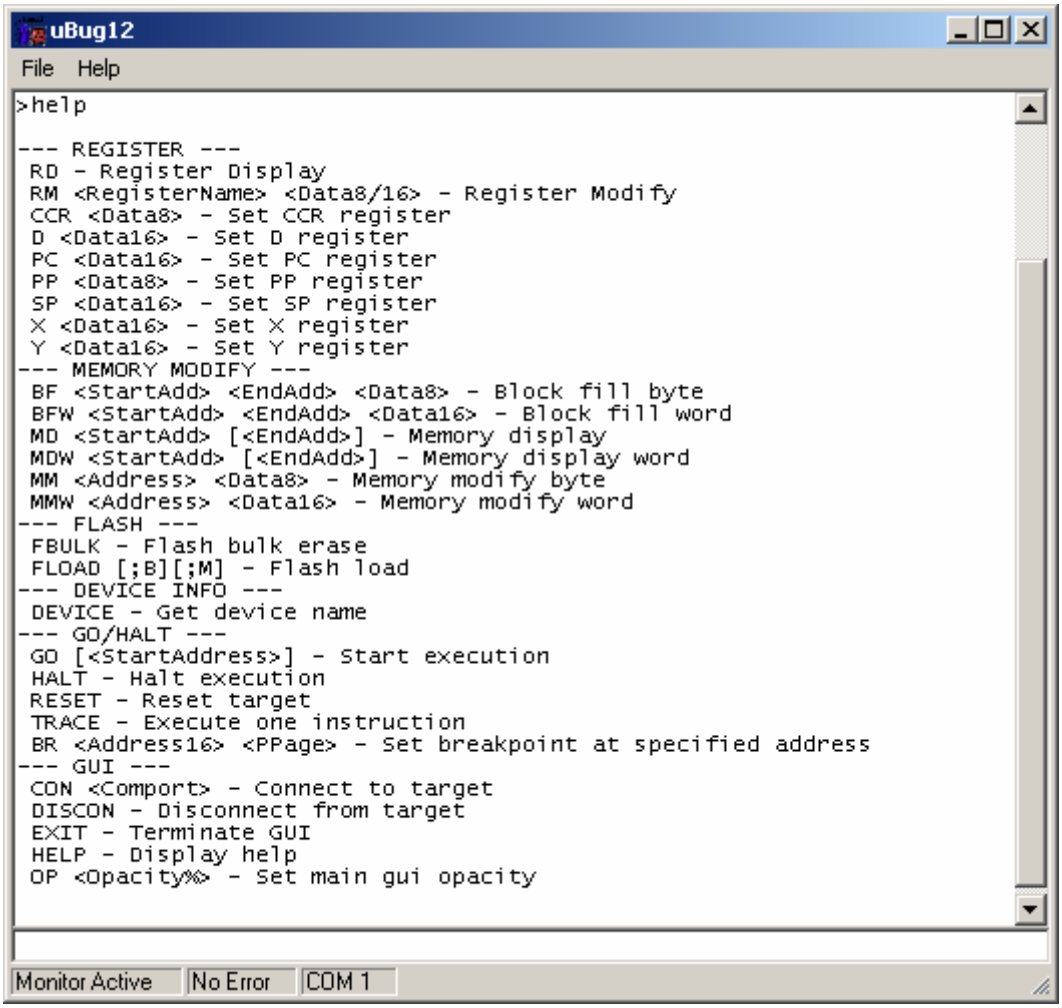
To disconnect uBUG12 from the serial port type the command ***discon***.



A Disconnected message will appear to indicate that the serial port is available for use by another application (eg. ***HyperTerm*** or ***Tera Term***).



A list of other uBUG12 commands can be viewed by typing the ***help*** command.



The commands are pretty well self explanatory but you should try them out to be familiar with their usage and capability.

Note that the Serial Monitor resides at 0xF800 – 0xFFFF. Therefore SerialMon will automatically re-locate the vector addresses below $F800.

Note: For NC12 families

SerialMon moves the internal RAM to 0x3800 – 0x3FFF. Make sure your code stack begins at 0x4000 (or at 0x3F80 if you intend to use uBUG12 as limited de-bugger). To implement this, insert the following into your code.

```
STACK           equ   $3F80        ;Stack at below Ubug12

 movb   #$00,INITRG              ;set registers at $0000
 movb   #$39,INITRM              ;move and set ram to end at $3fff
```

Note: For E128 families

SerialMon moves the internal RAM to $2000 - $3FFF. Make sure your code stack begins at $4000 or at $3F80 if you intend to use uBUG12 as limited debugger. To make sure this is done you can add the code below to your code.

```
STACK           equ   $3F80        ;Stack at below Ubug12

 movb   #$00,INITRG              ;set registers at $0000
 movb   #$39,INITRM              ;move and set ram to end at $3fff
```

The last thing to note (E128 and C32) is the uBUG12 enabled the PLL during Load mode. In Run mode the PLL is NOT enabled as the user maynot want this feature enabled.

The code below shows how to enable the PLL.

```
OscFreq         equ     8000            ;Enter Osc speed
initSYNR        equ     $02             ; mult by synr + 1 = 3 (24MHz)
initREFDV       equ     $00             ;
PLLSEL          equ     %10000000       ;PLL select bit
LOCK            equ     %00001000       ;lock status bit
PLLON           equ     %01000000       ;phase lock loop on bit


; Initialize clock generator and PLL
 bclr    CLKSEL,PLLSEL           ;disengage PLL to system
 bset    PLLCTL,PLLON            ;turn on PLL

 movb    #initSYNR,SYNR          ;set PLL multiplier
 movb    #initREFDV,REFDV        ;set PLL divider

 nop
 nop
 nop
 nop

 brclr   CRGFLG,LOCK,*+0         ;while (!(crg.crgflg.bit.lock==1))
 bset    CLKSEL,PLLSEL           ;engage PLL to system
```

Examining the content of the MCU:

Below is the S-record (test.s19) that was programmed into the Adapt9S12E128.

```
S10E4000CF3FFF16405587CE20008EF6
S110400B200027056A000820F6CE405ACD9B
S111401820008E405A2706180A307020F51634
```

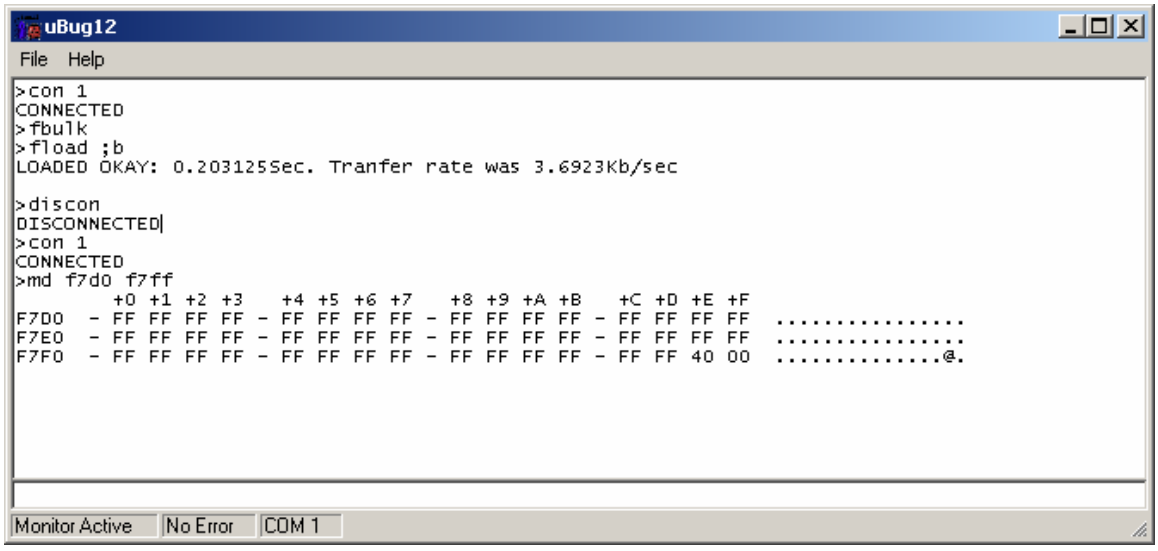
S1074026402A20FE0A
S110402A34B7751B9EC6FF7B025AC6FF7B90
S110403702584A8000382010C6FF7B025852
S10D40444A8000387902584A8000CF
S10A404E3820EEB757303DA6
S21138800034B7751B9ECC00006C1EEC1EC3FA
S21238800D00016C1EEC1E8C03E82DF2B75730BF
S20538801B0A1D
S111FFD0FFFFFFFFFFFFFFFFFFFFFFFFFFFF2D
S111FFDEFFFFFFFFFFFFFFFFFFFFFFFFFFFF1F
S111FFECFFFFFFFFFFFFFFFFFFFFFFFFFFFF11
S109FFFAFFFFFFFF4000C1
S10840551D0016073DEB
S9034000BC

These are the area of interest where the S-record is programmed to. Let us start with the interrupt vector area. As stated previously, Serial Monitor re-locates the vector address at below $F800.

S111FFD0FFFFFFFFFFFFFFFFFFFFFFFFFFFF2D
S111FFDEFFFFFFFFFFFFFFFFFFFFFFFFFFFF1F
S111FFECFFFFFFFFFFFFFFFFFFFFFFFFFFFF11
S109FFFAFFFFFFFF4000C1

Use uBUG12 to memory dump from $F7D0 to $F7FF by the command ***md f7d0 f7ff.***

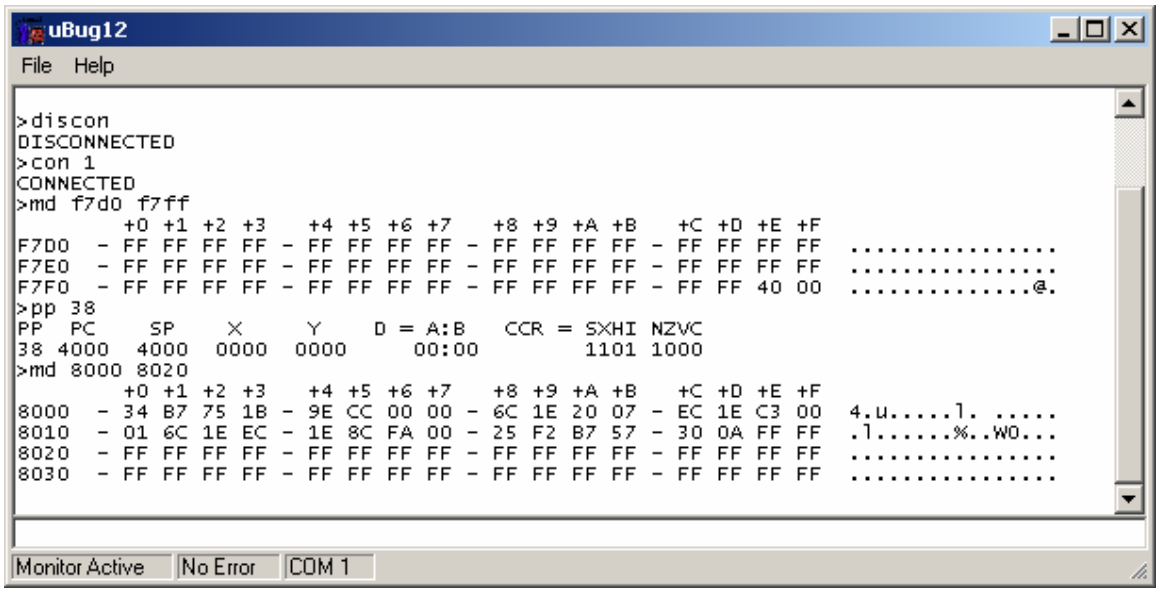
Note the content of the memory address at $F7FE:$F7FF is $4000, the RESET vector.



Use uBUG12 to memory dump address $8000 to $8020 by the command ***md 8000 8020.*** Please note the PPAGE = 38 the first available PPAGE as defined by address range 0xE0000 to 0xF7FFF.

S211 **38** 8000 ***34B7751B9ECC00006C1EEC1EC3*** FA
S212 **38** 800D ***00016C1EEC1E8C03E82DF2B75730*** BF
S205 **38** 801B ***0A*** 1D

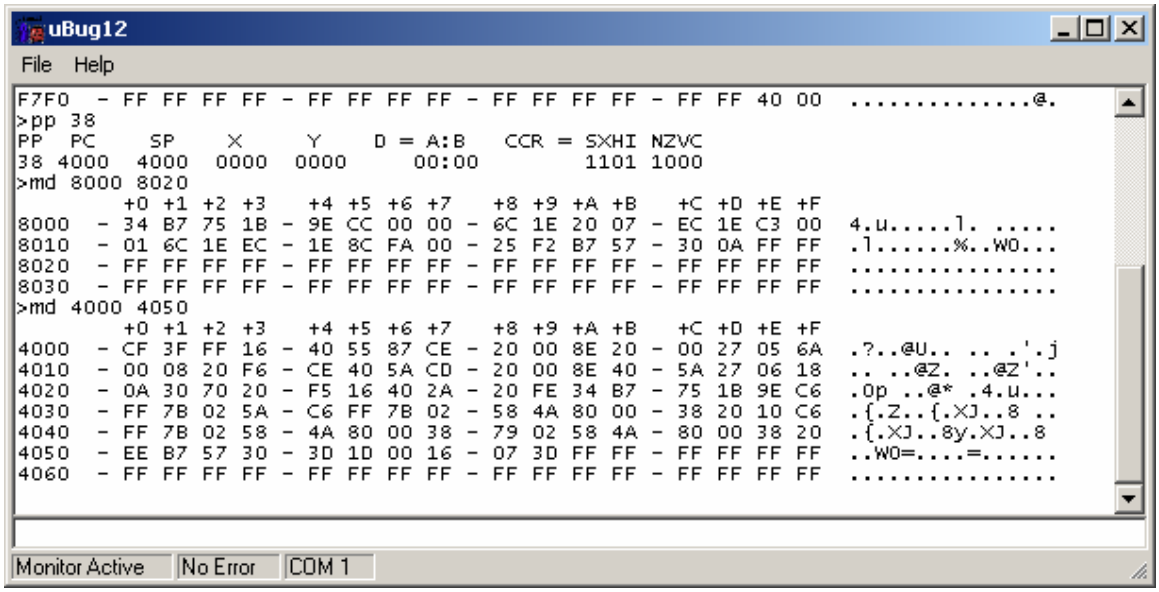
Change the PPAGE (PP) to $38 using the command ***pp 38*** then a memory dump of the content from $8000 to $8020 as shown. Note the content are the same.



The content of address beginning at $4000 to $4050

S10E **4000** ***CF3FFF16405587CE20008E*** F6
S110 **400B** ***200027056A000820F6CE405ACD*** 9B
S111 **4018** ***20008E405A2706180A307020F516*** 34
S107 **4026** ***402A20FE*** 0A
S110 **402A** ***34B7751B9EC6FF7B025AC6FF7B*** 90
S110 **4037** ***02584A8000382010C6FF7B0258*** 52
S10D **4044** ***4A8000387902584A8000*** CF
S10A **404E** ***3820EEB757303D*** A6

Note that the memory dump is the same as the S-record.



This concludes the use of ICC12 from erasing and programming FLASH with using uBUG12.

HCS12E128.H

```
#ifndef __HCS12E128_H
#define __HCS12E128_H

/* macros, generic #define etc.
 */
#include <hc12def.h>

/* base address of register block, change this if you relocate the register
 * block. This is for S12E128 !
 */

#define _IO_BASE        0
#define _ADDR(off)      (unsigned char volatile *)(_IO_BASE + off)
#define _P(off)         *(unsigned char volatile *)(_IO_BASE + off)
#define _LP(off)        *(unsigned short volatile *)(_IO_BASE + off)

#define PORTA   _P(0x00)
#define PORTB   _P(0x01)
#define DDRA    _P(0x02)
#define DDRB    _P(0x03)

//#define Reserved      _P(0x04)
//#define Reserved      _P(0x05)
//#define Reserved      _P(0x06)
//#define Reserved      _P(0x07)

#define PORTE   _P(0x08)
#define DDRE    _P(0x09)
#define PEAR    _P(0x0A)
#define MODE    _P(0x0B)
#define PUCR    _P(0x0C)
#define RDRIV   _P(0x0D)
#define EBICTL  _P(0x0E)

//#define reserved      _P(0x0F)

#define INITRM  _P(0x10)
#define INITRG  _P(0x11)
#define INITEE  _P(0x12)
#define MISC    _P(0x13)
#define MTST0   _P(0x14)
#define ITCR    _P(0x15)
#define ITEST   _P(0x16)
#define MIST1   _P(0x17)
```

```
//#define Reserved      _P(0x18)
#define VREGCTRL      _P(0x19)

#define PARTIDH  _P(0x1A)
#define PARTIDL  _P(0x1B)
#define PARTID   _LP(0x1A)

#define MEMSIZ0  _P(0x1C)
#define MEMSIZ1  _P(0x1D)
#define INTCR    _P(0x1E)
#define HPRIO    _P(0x1F)

#define BDGC1      _P(0x20)
#define DBGCS      _P(0x21)
#define DBGTBH      _P(0x22)
#define DBGTBL      _P(0x23)
#define DBGCNT      _P(0x24)
#define DBGCCX      _P(0x25)
#define DBGCCH      _P(0x26)
#define DBGCCL      _P(0x27)

#define BKPCT0   _P(0x28)
#define BKPCT1   _P(0x29)
#define BKP0X    _P(0x2A)
#define BKP0H    _P(0x2B)
#define BKP0L    _P(0x2C)
#define BKP0     _LP(0x2B)

#define BKP1X    _P(0x2D)
#define BKP1H    _P(0x2E)
#define BKP1L    _P(0x2F)
#define BKP1     _LP(0x2E)

#define PPAGE     _P(0x30)

//#define RESERVED      _P(0x31)
//#define RESERVED       _P(0x32)
//#define RESERVED      _P(0x33)

#define SYNR     _P(0x34)
#define REFDV    _P(0x35)
#define CTFLG    _P(0x36)
#define CRGFLG   _P(0x37)
#define CRGINT   _P(0x38)
#define CLKSEL   _P(0x39)
```

```
#define PLLCTL   _P(0x3A)
#define RTICTL   _P(0x3B)
#define COPCTL   _P(0x3C)
#define FORBYP   _P(0x3D)
#define CTCTL    _P(0x3E)
#define ARMCOP   _P(0x3F)

#define T0IOS    _P(0x40)
#define T0CFORC  _P(0x41)
#define T0OC7M   _P(0x42)
#define T0OC7D   _P(0x43)

#define T0CNT    _LP(0x44)

#define T0SCR1   _P(0x46)
#define T0TOV    _P(0x47)
#define T0CTL1   _P(0x48)
#define T0CTL2   _P(0x49)
#define T0CTL3   _P(0x4A)
#define T0CTL4   _P(0x4B)
#define T0IE     _P(0x4C)
#define T0SCR2   _P(0x4D)
#define T0FLG1   _P(0x4E)
#define T0FLG2   _P(0x4F)

//#define RESERVED     _LP(0x50)
//#define RESERVED     _LP(0x52)
//#define RESERVED     _LP(0x54)
//#define RESERVED     _LP(0x56)

#define T0C4     _LP(0x58)
#define T0C5     _LP(0x5A)
#define T0C6     _LP(0x5C)
#define T0C7     _LP(0x5E)

#define P0ACTL    _P(0x60)
#define P0AFLG    _P(0x61)
#define P0ACNTH    _P(0x62)
#define P0ACNTL    _P(0x63)
#define P0ACNT      _LP(0x62)

//#define Reserved    _P(0x64 - 7F)

#define ATD0CTL0  _P(0x80)
#define ATD0CTL1  _P(0x81)
#define ATD0CTL2  _P(0x82)
```

```
#define ATD0CTL3  _P(0x83)
#define ATD0CTL4  _P(0x84)
#define ATD0CTL5  _P(0x85)

#define ATD0STAT0   _P(0x86)
//#define Reserved   _P(0x87)
#define ATDTEST0   _P(0x88)
#define ATDTEST1   _P(0x89)
#define ATD0STAT1     _P(0x8A)
#define ATD1STAT1      _P(0x8B)
#define ATDDIEN0      _P(0x8C)
#define ATDDIEN1   _P(0x8D)

#define PORTAD0    _P(0x8E)

#define PORTAD1   _P(0x8F)

#define ATDDR0H  _P(0x90)
#define ATDDR0L  _P(0x91)
#define ATDDR0   _LP(0x90)

#define ATDDR1H  _P(0x92)
#define ATDDR1L  _P(0x93)
#define ATDDR1   _LP(0x92)

#define ATDDR2H _P(0x94)
#define ATDDR2L _P(0x95)
#define ATDDR2  _LP(0x94)

#define ATDDR3H _P(0x96)
#define ATDDR3L _P(0x97)
#define ATDDR3  _LP(0x96)

#define ATDDR4H _P(0x98)
#define ATDDR4L _P(0x99)
#define ATDDR4  _LP(0x98)

#define ATDDR5H _P(0x9A)
#define ATDDR5L _P(0x9B)
#define ATDDR5  _LP(0x9A)

#define ATDDR6H _P(0x9C)
#define ATDDR6L _P(0x9D)
#define ATDDR6  _LP(0x9C)

#define ATDDR7H _P(0x9E)
```

```
#define ATDDR7L _P(0x9F)
#define ATDDR7  _LP(0x9E)

#define ATDDR8H _P(0xA0)
#define ATDDR8L _P(0xA1)
#define ATDDR8  _LP(0xA0)

#define ATDDR9H _P(0xA2)
#define ATDDR9L _P(0xA3)
#define ATDDR9  _LP(0xA2)

#define ATDDR10H _P(0xA4)
#define ATDDR10L _P(0xA5)
#define ATDDR10  _LP(0xA4)

#define ATDDR11H _P(0xA6)
#define ATDDR11L _P(0xA7)
#define ATDDR11  _LP(0xA6)

#define ATDDR12H _P(0xA8)
#define ATDDR12L _P(0xA9)
#define ATDDR12  _LP(0xA8)

#define ATDDR13H _P(0xAA)
#define ATDDR13L _P(0xAB)
#define ATDDR13  _LP(0xAA)

#define ATDDR14H _P(0xAC)
#define ATDDR14L _P(0xAD)
#define ATDDR14  _LP(0xAC)

#define ATDDR15H _P(0xAE)
#define ATDDR15L _P(0xAF)
#define ATDDR15  _LP(0xAE)


#define SCI0BDH  _P(0xC8)
#define SCI0BDL  _P(0xC9)
#define SCI0BD   _LP(0xC8)

#define SC0CR1   _P(0xCA)
#define SCI0CR1  _P(0xCA)
#define SCI0CR2  _P(0xCB)

#define SCI0SR1  _P(0xCC)
#define SC0SR1     SCI0SR1              /* compatability */
```

```
#define SC0SR2   _P(0xCD)
#define SCI0SR2  _P(0xCD)

#define SCI0DRH  _P(0xCE)
#define SCI0DRL  _P(0xCF)
#define SC0DRL     SCI0DRL              /* compatability */
#define SCI0DR   _LP(0xCE)

#define SCI1BDH  _P(0xD0)
#define SCI1BDL  _P(0xD1)
#define SCI1BD   _LP(0xD0)

#define SCI1CR1   _P(0xD2)
#define SCI1CR2   _P(0xD3)
#define SCI1SR1   _P(0xD4)
#define SCI1SR2   _P(0xD5)

#define SCI1DRH   _P(0xD6)
#define SCI1DRL   _P(0xD7)
#define SCI1DR    _LP(0xD6)

#define SPICR1   _P(0xD8)
#define SPICR2   _P(0xD9)
#define SPIBR    _P(0xDA)
#define SPISR    _P(0xDB)

//#define Reserved     _P(0xDC)

#define SPIDR    _P(0xDD)

//#define Reserved     _P(0xDE)
//#define Reserved     _P(0xDF)

#define IBAD    _P(0xE0)
#define IBFD    _P(0xE1)
#define IBCR    _P(0xE2)
#define IBSR    _P(0xE3)
#define IICDR   _P(0xE4)

//#define Reserved     _P(0xE5)
//#define Reserved     _P(0xE6)
//#define Reserved     _P(0xE7)

#define SCI2BDH  _P(0xE8)
#define SCI2BDL  _P(0xE9)
#define SCI2BD   _LP(0xE8)
```

```
#define SCI2CR1   _P(0xEA)
#define SCI2CR2   _P(0xEB)
#define SCI2SR1   _P(0xEC)
#define SCI2SR2   _P(0xED)

#define SCI2DRH   _P(0xEE)
#define SCI2DRL   _P(0xEF)
#define SCI2DR    _LP(0xEE)


#define DAC0C0  _P(0xF0)
#define DAC0C1  _P(0xF1)
#define DAC0DL  _P(0xF2)
#define DAC0DR  _P(0xF3)


#define DAC1C0  _P(0xF4)
#define DAC1C1  _P(0xF5)
#define DAC1DL  _P(0xF6)
#define DAC1DR  _P(0xF7)

#define FCLKDIV _P(0x0100)
#define FSEC    _P(0x0101)
#define C0BTR0  _P(0x0102)

//#define Reserved for Factory Test _P(0x0102)

#define FCNFG  _P(0x0103)
#define FPROT  _P(0x0104)
#define FSTAT  _P(0x0105)
#define FCMD   _P(0x0106)

//#define Reserved _P(0x0107 - 013F)

#define T1IOS     _P(0x140)
#define T1CFORC   _P(0x141)
#define T1OC7M    _P(0x142)
#define T1OC7D    _P(0x143)

#define T1CNT     _LP(0x144)

#define T1SCR1    _P(0x146)
#define T1TOV     _P(0x147)
#define T1CTL1    _P(0x148)
#define T1CTL2    _P(0x149)
```

```
#define T1CTL3    _P(0x14A)
#define T1CTL4    _P(0x14B)
#define T1IE      _P(0x14C)
#define T1SCR2    _P(0x14D)
#define T1FLG1    _P(0x14E)
#define T1FLG2    _P(0x14F)

//#define RESERVED      _LP(0x50)
//#define RESERVED      _LP(0x52)
//#define RESERVED      _LP(0x54)
//#define RESERVED      _LP(0x56)

#define T1C4      _LP(0x158)
#define T1C5      _LP(0x15A)
#define T1C6      _LP(0x15C)
#define T1C7      _LP(0x15E)

#define P1ACTL     _P(0x160)
#define P1AFLG     _P(0x161)
#define P1ACNTH     _P(0x162)
#define P1ACNTL     _P(0x163)
#define P1ACNT       _LP(0x162)

//#define RESERVED 0x164-0x17F
#define T2IOS     _P(0x180)
#define T2CFORC   _P(0x181)
#define T2OC7M    _P(0x182)
#define T2OC7D    _P(0x183)

#define T2CNT     _LP(0x184)

#define T2SCR1    _P(0x186)
#define T2TOV     _P(0x187)
#define T2CTL1    _P(0x188)
#define T2CTL2    _P(0x189)
#define T2CTL3    _P(0x18A)
#define T2CTL4    _P(0x18B)
#define T2IE      _P(0x18C)
#define T2SCR2    _P(0x18D)
#define T2FLG1    _P(0x18E)
#define T2FLG2    _P(0x18F)

//#define RESERVED      _LP(0x50)
//#define RESERVED      _LP(0x52)
//#define RESERVED      _LP(0x54)
//#define RESERVED      _LP(0x56)
```

```
#define T2C4      _LP(0x198)
#define T2C5      _LP(0x19A)
#define T2C6      _LP(0x19C)
#define T2C7      _LP(0x19E)

#define P2ACTL     _P(0x1A0)
#define P2AFLG     _P(0x1A1)
#define P2ACNTH    _P(0x1A2)
#define P2ACNTL    _P(0x1A3)
#define P2ACNT       _LP(0x1A2)

//#define RESERVED 0x1B0 - 0x1DF

#define PWME          _P(0x1E0)
#define PWMPOL      _P(0x1E1)
#define PWMCLK      _P(0x1E2)
#define PWMPRCLK   _P(0x1E3)
#define PWMCAE      _P(0x1E4)
#define PWMCTL      _P(0x1E5)
#define PWMTST      _P(0x1E6)
#define PWMPRSC    _P(0x1E7)
#define PWMSCLA    _P(0x1E8)
#define PWMSCLB    _P(0x1E9)
#define PWMSCNTA   _P(0x1EA)
#define PWMSCNTB   _P(0x1EB)
#define PWMCNT0    _P(0x1EC)
#define PWMCNT1    _P(0x1ED)
#define PWMCNT2    _P(0x1EE)
#define PWMCNT3    _P(0x1EF)
#define PWMCNT4    _P(0x1F0)
#define PWMCNT5    _P(0x1F1)
#define PWMPER0    _P(0x1F2)
#define PWMPER1    _P(0x1F3)
#define PWMPER2    _P(0x1F4)
#define PWMPER3    _P(0x1F5)
#define PWMPER4    _P(0x1F6)
#define PWMPER5    _P(0x1F7)
#define PWMDTY0    _P(0x1F8)
#define PWMDTY1    _P(0x1F9)
#define PWMDTY2    _P(0x1FA)
#define PWMDTY3    _P(0x1FB)
#define PWMDTY4    _P(0x1FC)
#define PWMDTY5    _P(0x1FD)

//#define RESERVED 0x1FE - 0x1FF
```

```
#define PMFCFG0     _P(0x200)
#define PMFCFG1     _P(0x201)
#define PMFCFG2     _P(0x202)
#define PMFCFG3     _P(0x203)
#define PMFFCTL     _P(0x204)
#define PMFFPIN     _P(0x205)
#define PMFFSAT     _P(0x206)
#define PMFQSMP     _P(0x207)
#define PMFDMPA     _P(0x208)
#define PMFDMPB     _P(0x209)
#define PMFDMPC     _P(0x20A)
#define PMFOUTC     _P(0x20C)
#define PMFOUTB     _P(0x20D)
#define PMFDTMS     _P(0x20E)
#define PMFCCTL     _P(0x20F)
#define PMFVAL0     _LP(0x210)
#define PMFVAL1     _LP(0x212)
#define PMFVAL2     _LP(0x214)
#define PMFVAL3     _LP(0x216)
#define PMFVAL4     _LP(0x218)
#define PMFVAL5     _LP(0x21A)
#define PMFENCA     _P(0x220)
#define PMFFQCA     _P(0x221)
#define PMFCNTA     _LP(0x222)
#define PMFMODA     _LP(0x224)
#define PMFDTMA     _LP(Ox226)
#define PMFENCB     _LP(Ox228)
#define PMFCNTB     _LP(Ox22A)
#define PMFMODB     _LP(Ox22C)
#define PMFENCC     _P(Ox230)
#define PMFFQCC     _P(Ox231)
#define PMFCNTC     _LP(Ox232)
#define PMFMODC     _LP(Ox234)
#define PMFDTMC     _LP(Ox236)

//#define RESERVED _P(0x0238 - 0x023C)

#define PTIM   _P(0x0251)
#define DRRM        _P(0x0252)
#define RDDM        _P(0x0253)
#define PERM        _P(0x0254)
#define PPSM        _P(0x0255)
#define WOMM        _P(0x0256)
//#define RESERVED                _P(0x0257)
#define PTP         _P(0x0258)
```

```
#define PTIP    _P(0x0259)
#define DDRP            _P(0x025A)
#define RDRP            _P(0x025B)
#define PERP            _P(0x025C)
#define PPSP            _P(0x025D)
//#define RESERVED          _P(0x025E)
//#define RESERVED          _P(0X025F)
#define PTQ             _P(0x0260)
#define PTIQ    _P(0x0261)
#define DDRQ            _P(0x0262)
#define RDRQ            _P(0x0263)
#define PERQ            _P(0x0264)
#define PPSQ            _P(0x0265)
//#define RESERVED          _P(0x0266)
//#define RESERVED          _P(0X0267)
#define PTU             _P(0x0268)
#define PTIU    _P(0x0269)
#define DDRU            _P(0x026A)
#define RDRU            _P(0x026B)
#define PERU            _P(0x026C)
#define PPSU            _P(0x026D)
#define MODRR           _P(0x026E)
//#define RESERVED          _P(0X026F)

#define PTAD                    _LP(0x0270)
#define PTIAD                   _LP(0x0272)
#define DDRAD                   _LP(0x0274)
#define RDRAD                   _LP(0x0276)
#define PERAD                   _LP(0x0278)
#define PPSAD                   _LP(0x027A)
#define PIEAD                   _LP(0x027C)
#define PIFAD                   _LP(0x027E)
//#define RESERVED          _P(0X0280 - 0X02FF)

//#define Unimplemented _P(0x0300 - 0x03FF)
#endif
```

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.