

Programming Adapt9S12E128 and NC12 with using uBUG12

This document assumes that the Serial monitor is not erased and is present on the Adapt9S12E128 and NC12.

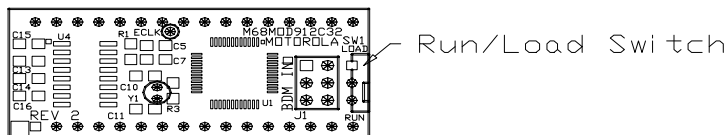
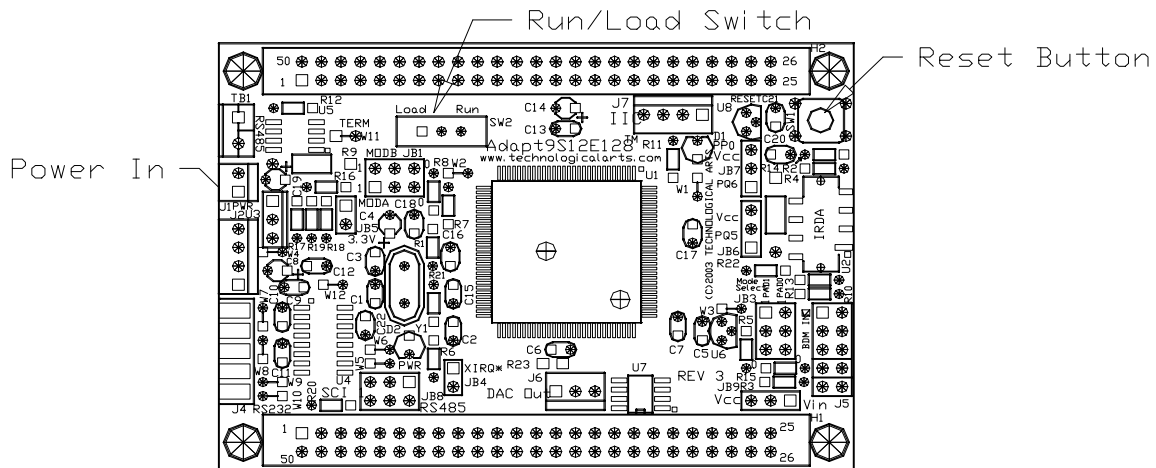
Download uBUG12 from Technological Arts which can be found at the link below <http://support.technologicalarts.ca/files/uBug12.zip>

or from the CD that came with the NC12 and E128 kit.

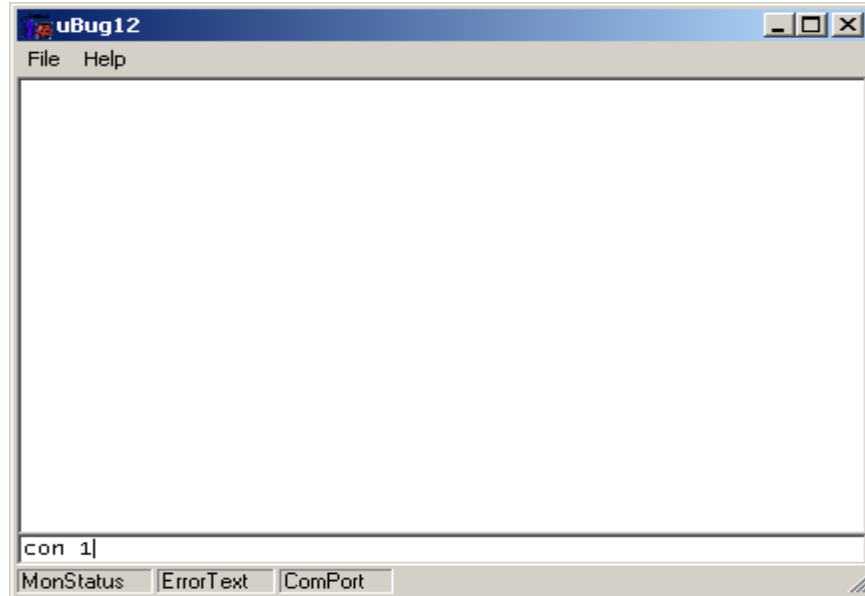
For windows 98 users the .NET framework must be installed before running uBUG12. The MS site link is

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

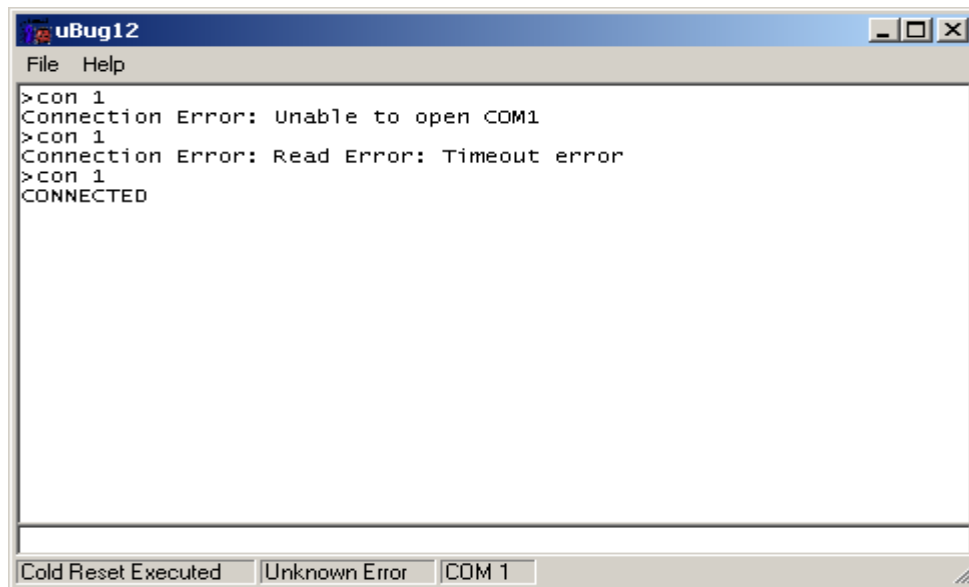
Switch the Run/Load switch to Load position and apply power to the board.



Double click on uBUG12 icon to execute program.



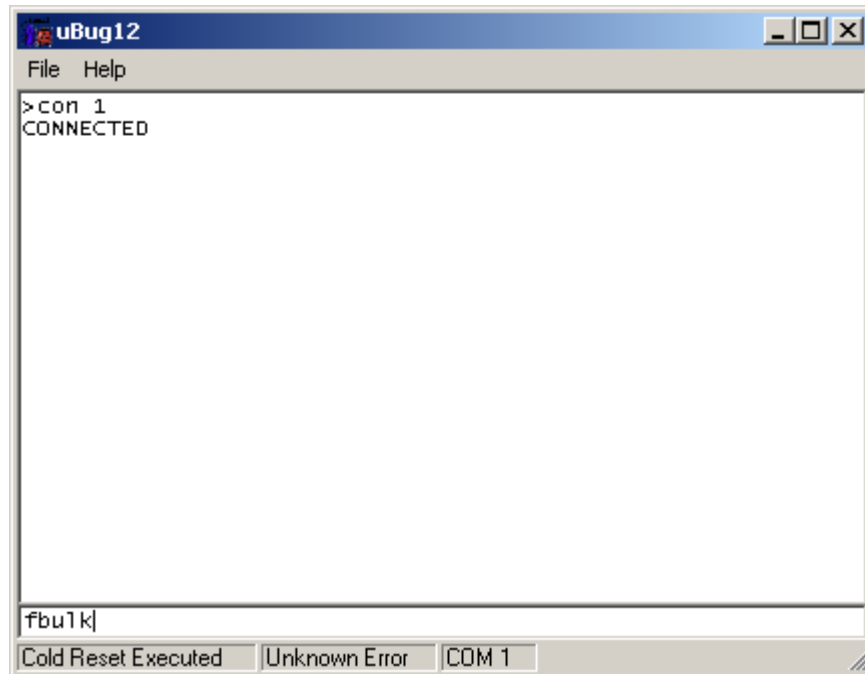
In the command bar type **con 1** for COM 1 or **con 2** for COM 2. A **CONNECTED** message will appear to indicate that a connection between PC and Adapt9S12E128 or NC12 that has been established.



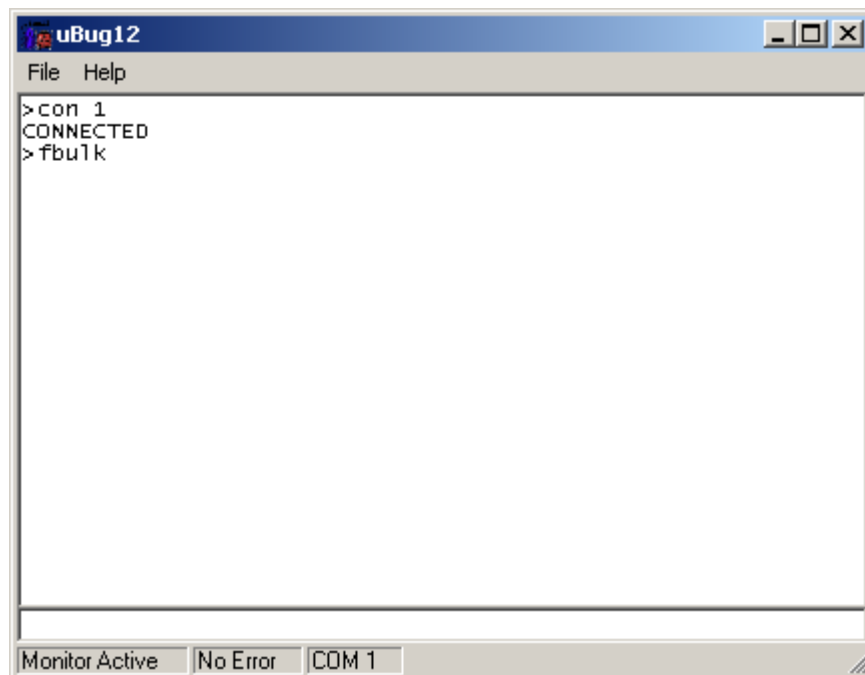
2 possible errors can occur:

Connection Error: Unable to open COM1 <- Another application is using the COM port

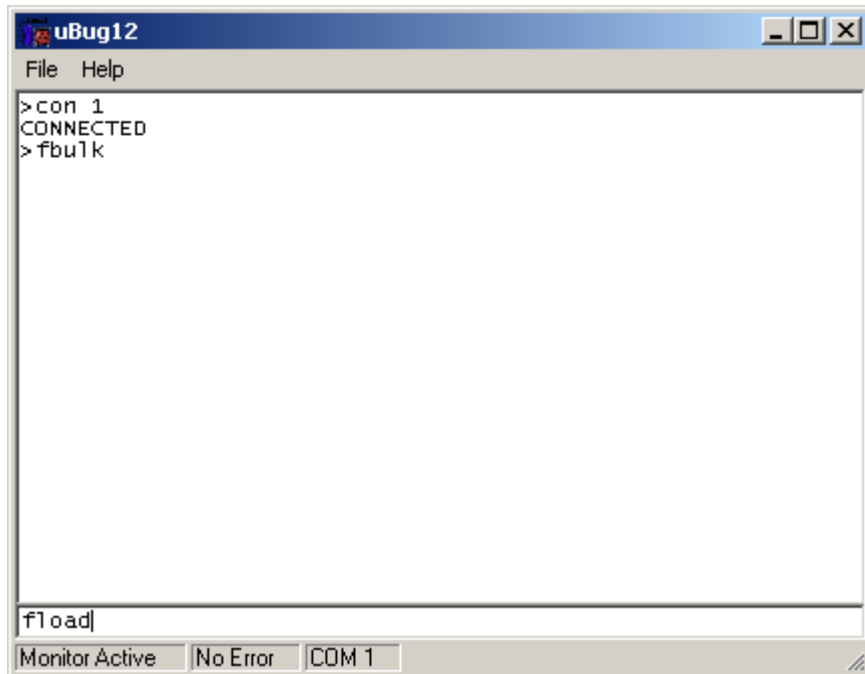
Connection Error: Read Error: Timeout error <- The MCU not currently in LOAD mode or the cable is disconnected from either PC or Docking Module



Make sure to erase the FLASH memory by typing the command **FBULK**.



To program, type the command **FLOAD ;b** for banked S19, S2, SX and formatted S19 (went thru SrecCVT program) records. For non-banked S2 the command is FLOAD.

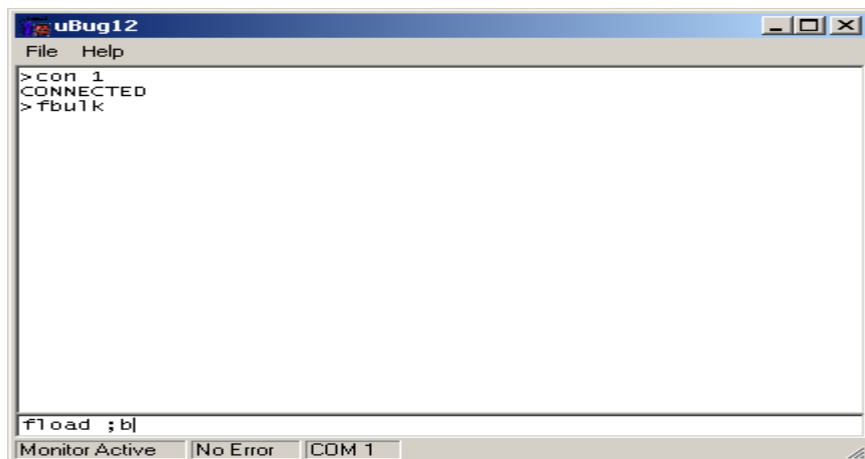


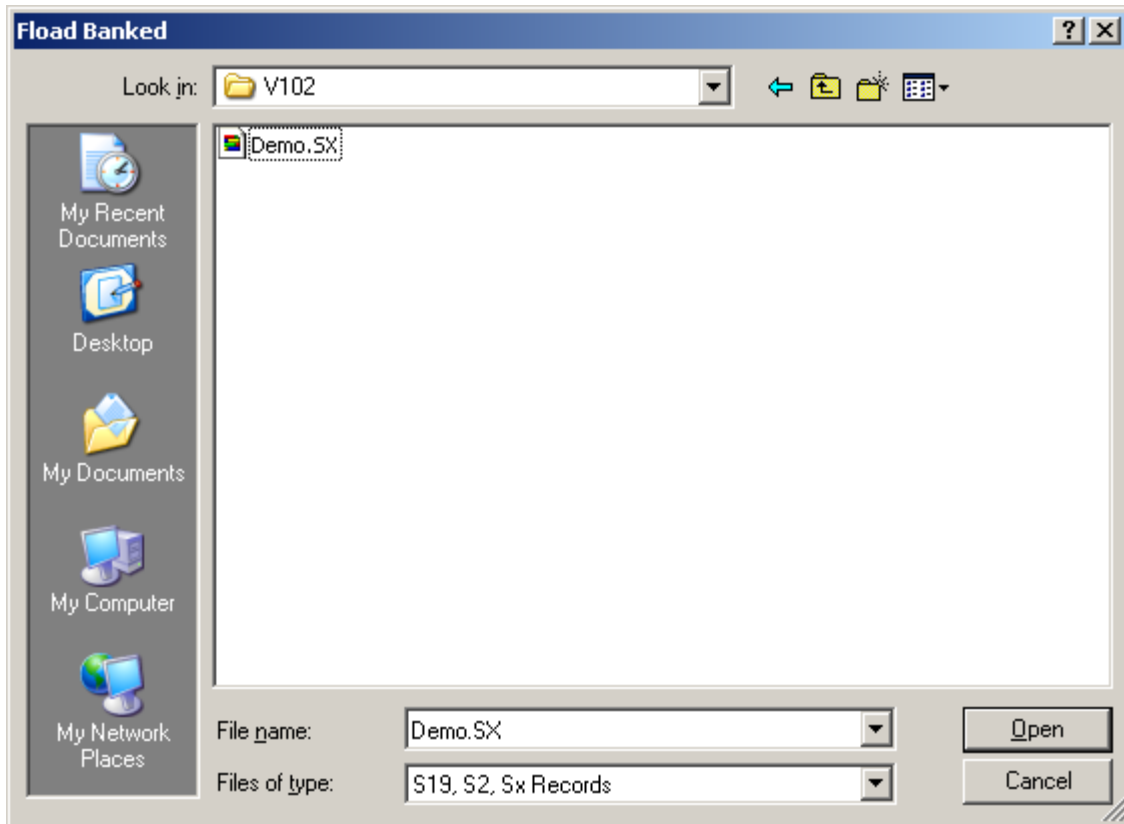
Once the command is invoked FLOAD explorer window will appear to help search and locate the S record to program the MCU with. Double click on the appropriate S record to initiate download to the MCU.

After successful programming slide the Run/Load switch to Run and press the reset button. The application is now running assuming the code is bug-free.

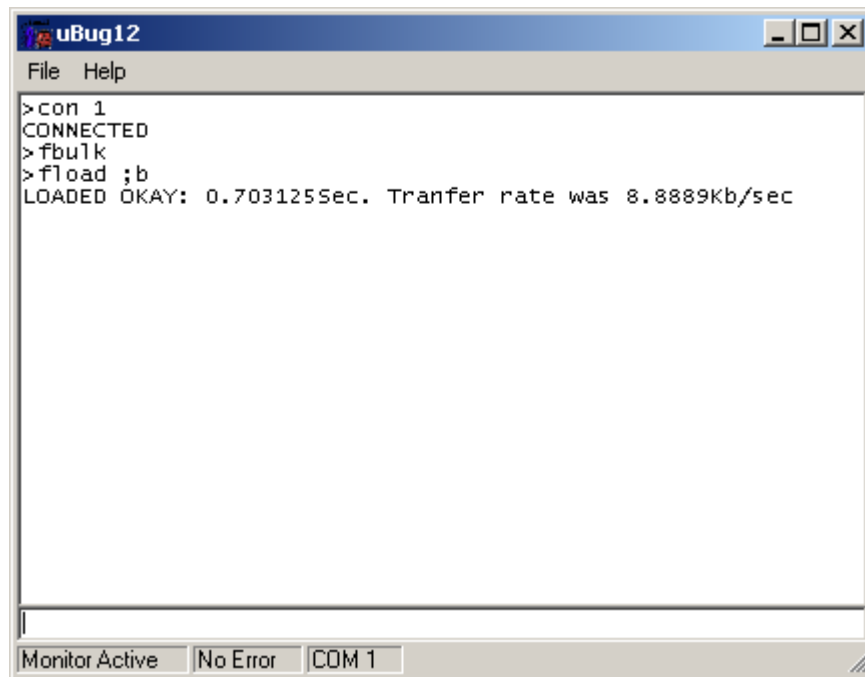
Uploading Banked S-record:

The command to upload banked S-record is **FLOAD ;B**. Note to include the **;b** option to let uBUG12 know that the S-record is banked. Users should be familiar with the differences between S19, SX, S2. See Appendix A for S-record explanation.



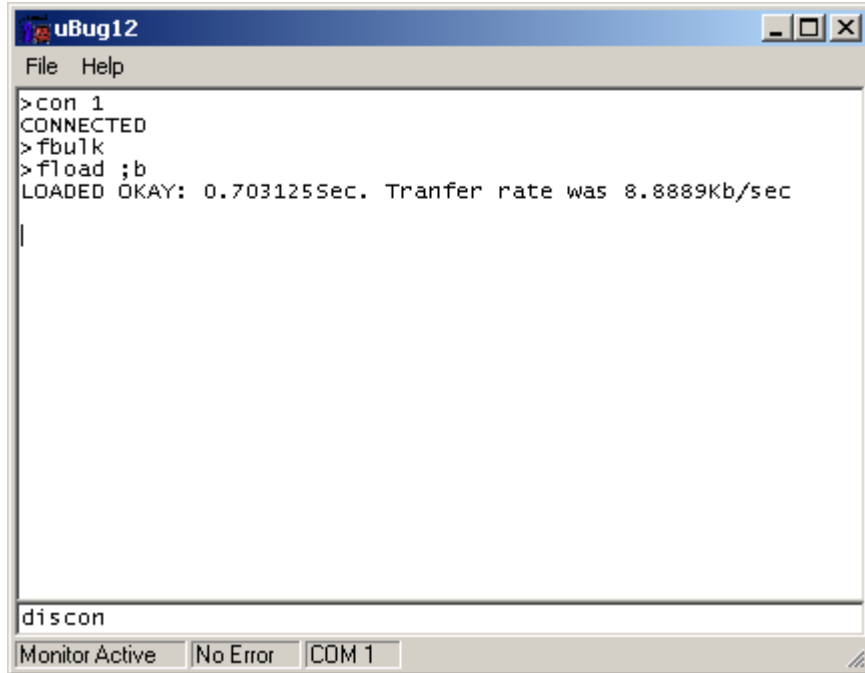


Double click on the file to initiate upload.

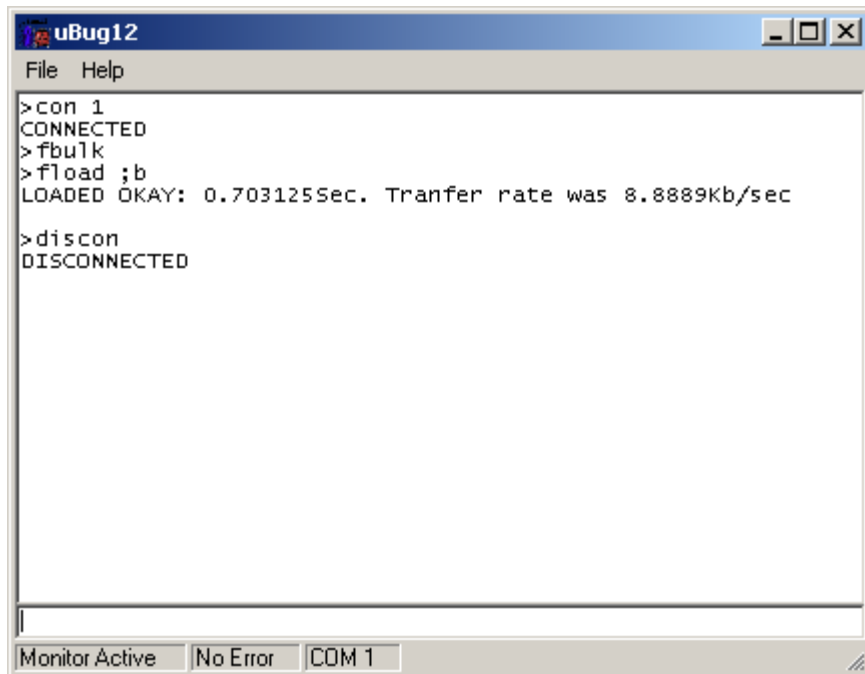


A good upload will show **LOADED OKAY** messages.

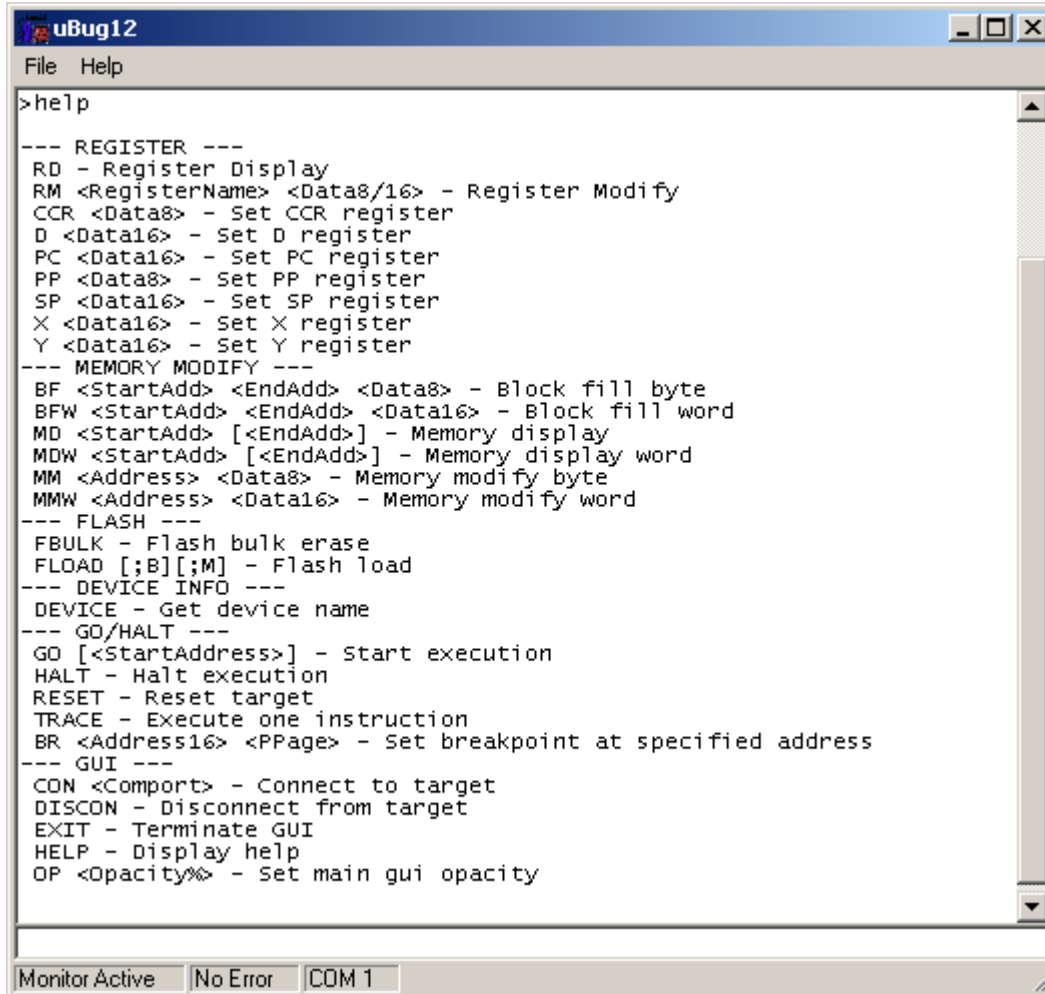
To disconnect uBUG12 GUI to serial port type the command is **discon**.



Disconnected message will appear to indicate that the serial is ready to be use by another application like **HyperTerm**.



Other uBUG12 commands are available by typing the *help* command.



```
uBug12
File Help
>help
--- REGISTER ---
RD - Register Display
RM <RegisterName> <Data8/16> - Register Modify
CCR <Data8> - Set CCR register
D <Data16> - Set D register
PC <Data16> - Set PC register
PP <Data8> - Set PP register
SP <Data16> - Set SP register
X <Data16> - Set X register
Y <Data16> - Set Y register
--- MEMORY MODIFY ---
BF <StartAdd> <EndAdd> <Data8> - Block fill byte
BFW <StartAdd> <EndAdd> <Data16> - Block fill word
MD <StartAdd> [<EndAdd>] - Memory display
MDW <StartAdd> [<EndAdd>] - Memory display word
MM <Address> <Data8> - Memory modify byte
MMW <Address> <Data16> - Memory modify word
--- FLASH ---
FBULK - Flash bulk erase
FLOAD [;B][;M] - Flash load
--- DEVICE INFO ---
DEVICE - Get device name
--- GO/HALT ---
GO [<StartAddress>] - Start execution
HALT - Halt execution
RESET - Reset target
TRACE - Execute one instruction
BR <Address16> <PPage> - Set breakpoint at specified address
--- GUI ---
CON <Comport> - Connect to target
DISCON - Disconnect from target
EXIT - Terminate GUI
HELP - Display help
OP <Opacity%> - Set main gui opacity

Monitor Active No Error CDM 1
```

The command help are self explanatory but one should try them out to be familiar with their usage and capability.

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

Note: For NC12 families

uBUG12 moves the internal RAM to \$3800 - \$3FFF. Make sure your code stack begins at \$4000 or at \$3F80 if you intend to use uBUG12 as limited de-bugger. 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
```

Note: For E128 families

uBUG12 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 de-bugger. 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 may not 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
```

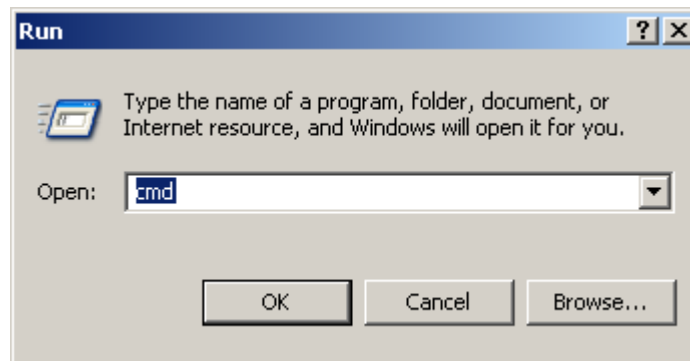

nop

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

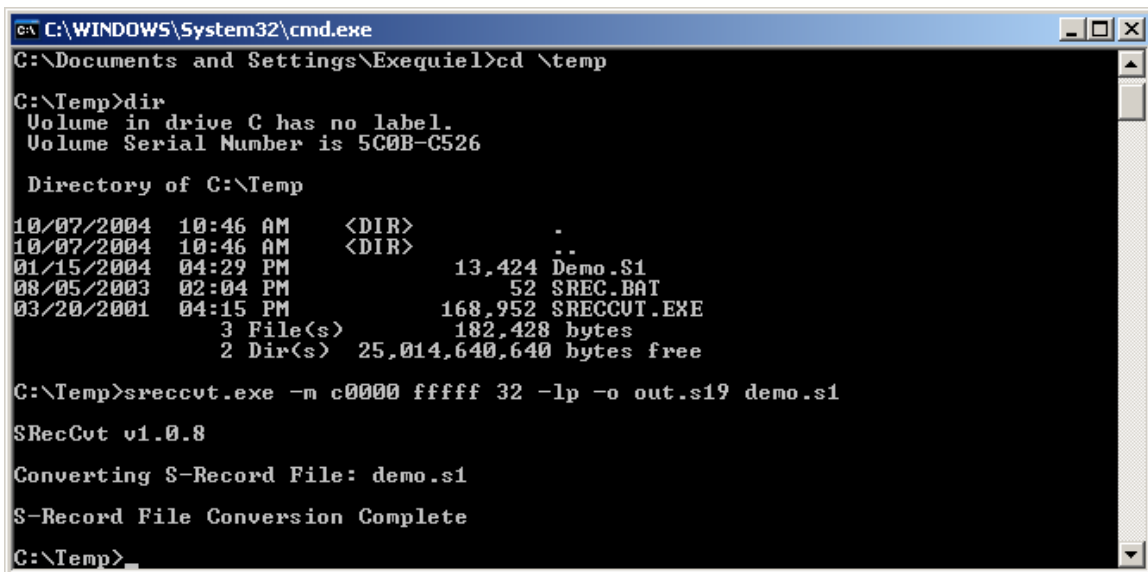
Using SrecCVT in DOS command line:

```
srecvct.exe -m C0000 FFFFF 32 -lp -o out.s19 yourfile.s19
```

Where **yourfile.s19** is to be converted and **out.s19** to be uploaded to FLASH. The examples below will show the sequence of events to achieve this. Select Start – Run then type CMD to run DOS



Make sure Srecvct.exe is at the same location as your file to be converted.

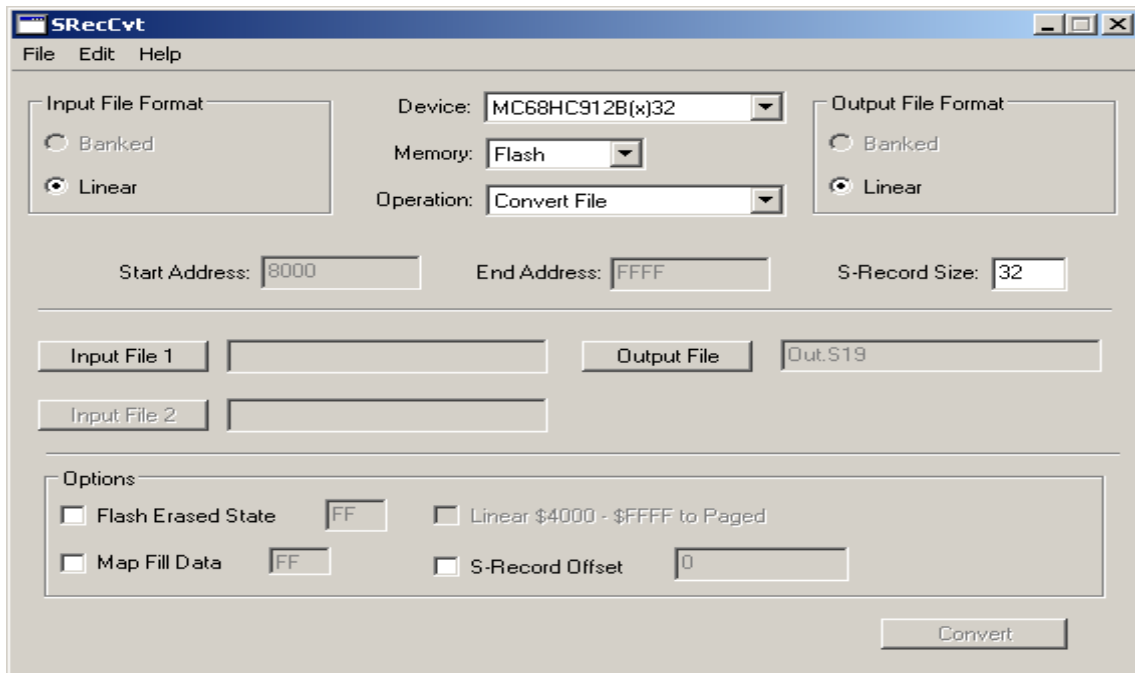
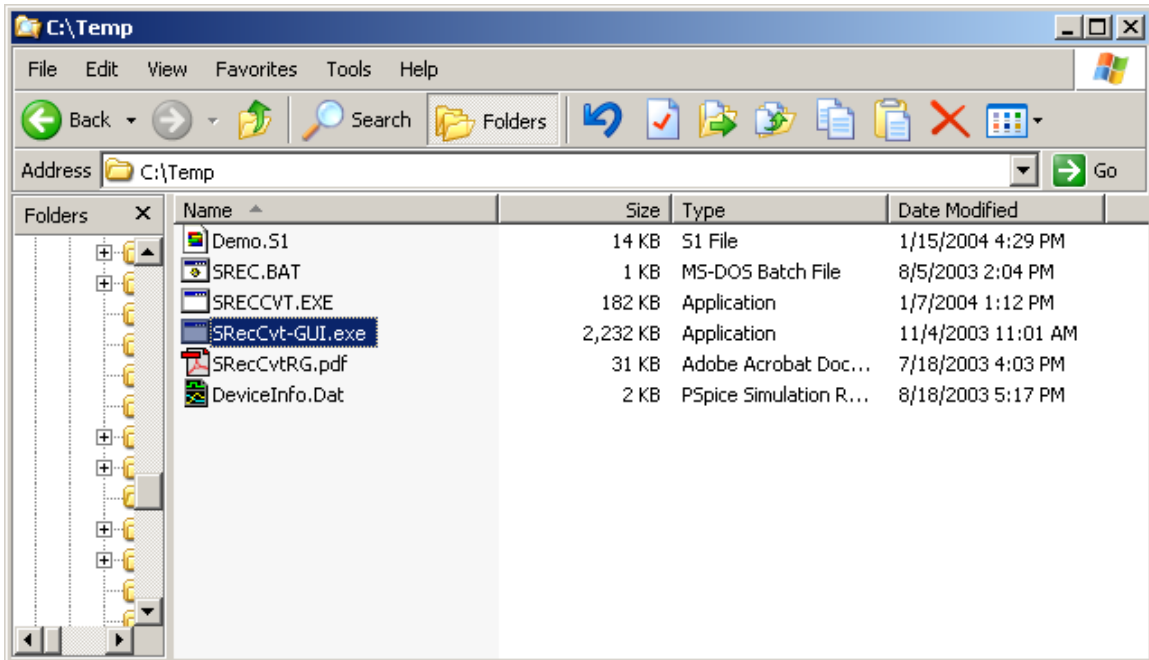


Here it shows a successful S-record conversion. And file Out.s19 is generated.

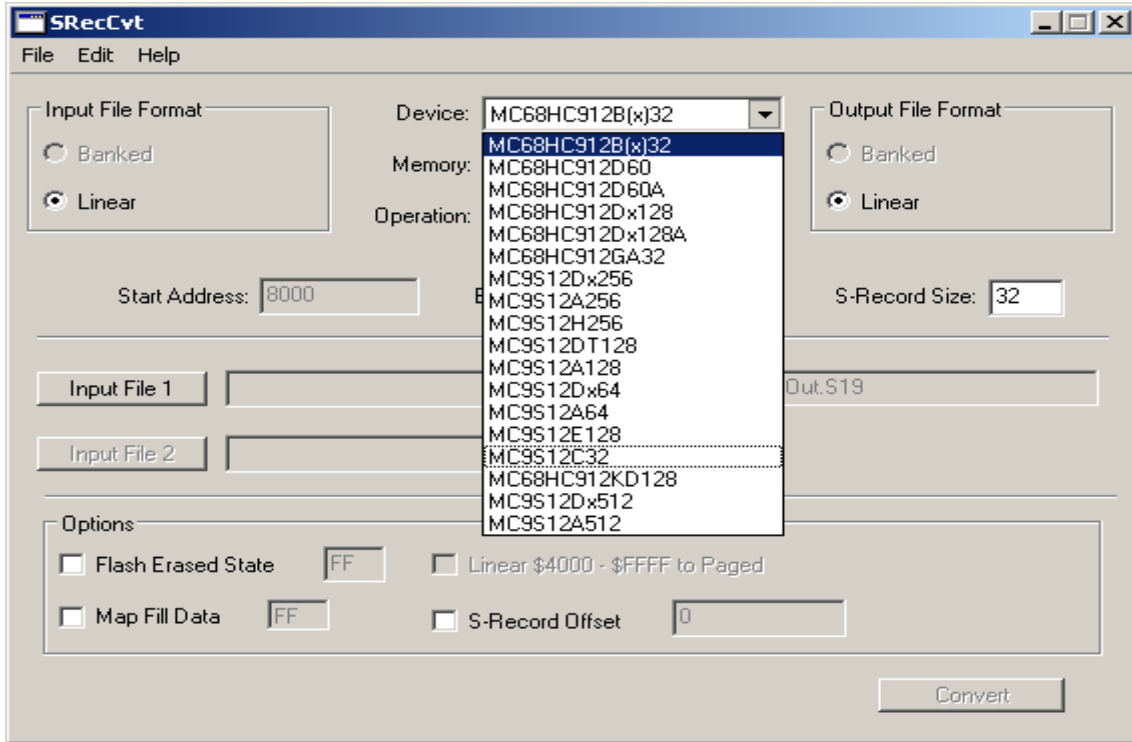
There is also GUI version of the SrecCVT program. One can download the program from www.freescale.com.

Using SrecCVT in Windows:

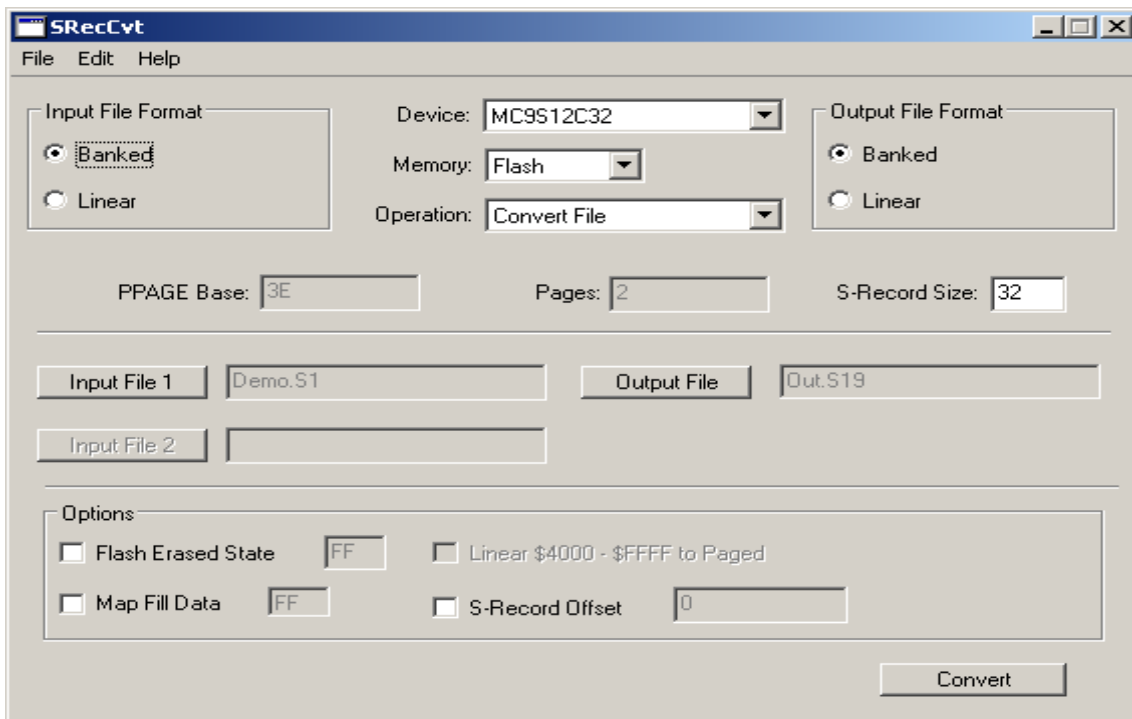
Double click on **SRecCvt-GUI.exe** to initiate program.



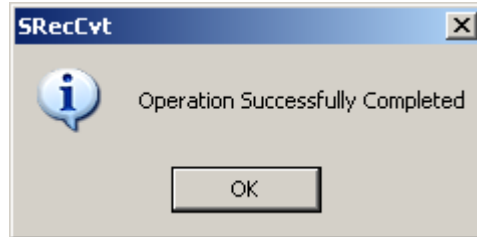
Choose or select the target device



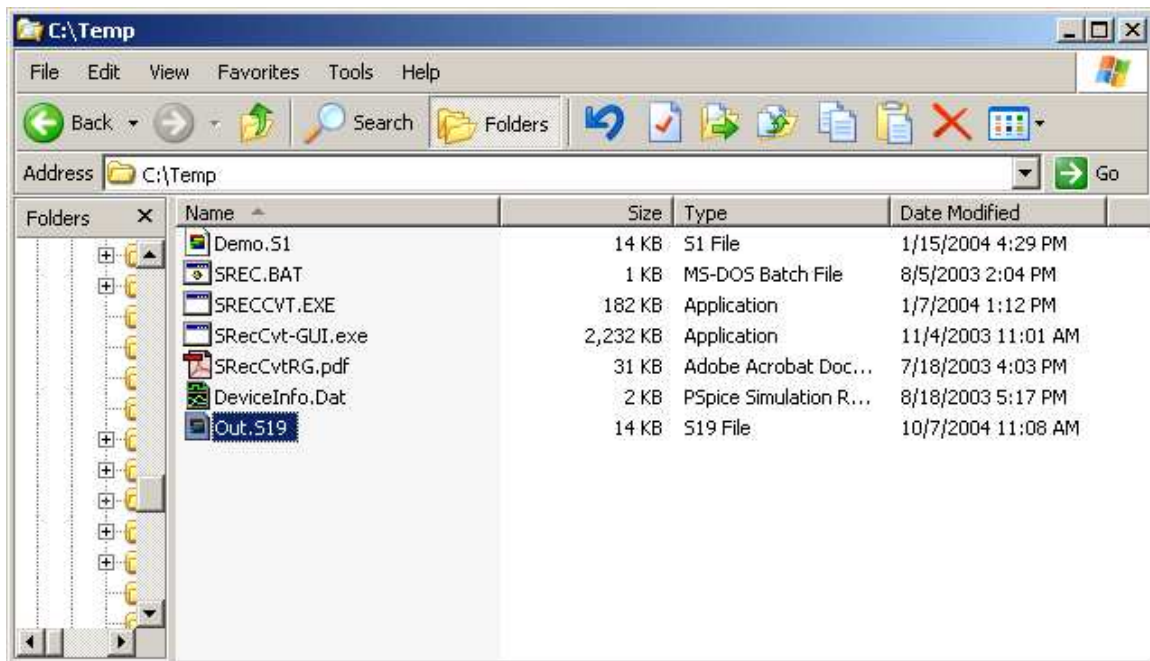
In this example MC9S12C32 is selected, Input file format is Banked, Output File Format is Banked. Input File name is Demo.s1 and the Output File name is Out.s19



Here it shows a successful conversion.



And file Out.s19 is generated.



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 interpreted 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 interpreted 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 interpreted 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 interpreted 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 interpreted 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 interpreted 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 interpreted 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; hexadecimal 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.