MOTOROLA DIGITAL SIGNAL PROCESSING DEVELOPMENT SOFTWARE

MOTOROLA DSP
LINKER/LIBRARIAN REFERENCE MANUAL

Motorola, Incorporated
Semiconductor Products Sector
DSP Division
6501 William Cannon Drive West
Austin, TX, 78735-8598

This manual documents the Linker and librarian as of version 6.0 of the software.

PREFACE

Notation

The notational conventions used in this manual are:

DIRECTIVE

All linker directives and options are shown in bold upper case to highlight them. However, the linker will recognize both upper and lower case for options and directives.

{ }

Contains a list of elements or directives, one of which must be selected. Each choice will be separated by a vertical bar. For example, {**R** I **L**} indicates that either **R** or **L** must be selected.

[ ]

Contains one or more optional elements. If more than one optional element is shown, the required element separators are indicated. All elements outside of the angle brackets (< >) must be specified as they appear. For example, the syntactical element [<number>,] requires the comma to be specified if the optional element <number> is selected.

< >

The element names are printed in lower case and contained in angle brackets. Some common elements used to describe linker options are:

<expr> or <expression>	A linker expression
<number>	A numeric constant
<string>	A string of ASCII characters enclosed in quotes
<delimiter>	A delimiter character
<option>	A linker option
<sym> or <symbol>	A linker symbol

Supporting Publications

DSP56000 Family Manual. Motorola, Inc. 1992.

DSP96002 User's Manual. Motorola, Inc. 1989.

DSP56100 Family Manual. Motorola, Inc. 1993.

DSP56300 Family Manual. Motorola, Inc. 1995.

DSP56800 Family Manual. Motorola, Inc. 1996.

Motorola DSP Simulator Reference Manual. Motorola, Inc. 1996.

Motorola DSP Assembler Reference Manual. Motorola, Inc. 1996.

TABLE OF CONTENTS

Chapter 5
MOTOROLA DSP S-RECORD CONVERSION UTILITY (SREC)

Chapter 6
MOTOROLA DSP COFF FILE DUMP UTILITY (COFDMP)

Appendix A
LINKER MESSAGES

Appendix B
LIBRARIAN MESSAGES

Appendix C
LINKER MAP FILE FORMAT

Chapter 1
MOTOROLA DSP LINKER

1.1 INTRODUCTION

The Motorola DSP Linker is a program that processes relocatable object files produced by the Motorola DSP assemblers, generating an absolute executable file which can be loaded directly into one of the Motorola DSP simulators, downloaded to an application development system, or converted to Motorola S-record format for PROM burning. A command line option provides for specification of a base address for each DSP memory space and logical location counter. In addition, a memory control file may be supplied to indicate absolute positioning of sections in DSP memory as well as physical mappings to internal and external memory. The Linker optionally generates a map file which shows memory assignment of sections by memory space and a sorted list of symbols with their load time values.

1.2 INSTALLING THE LINKER

The Linker is distributed on various media and in different formats depending on the host operating system environment. See Appendix G in the **Motorola DSP Assembler Reference Manual**, HOST-DEPENDENT INFORMATION, for details on installing and operating the Linker on your particular machine.

1.3 RUNNING THE LINKER

The general format of the command line to invoke the Linker is:

DSPLNK [options] <filenames>

where:

[options]

Any of the following command line options. These can be in any order, but must precede the list of source filenames. Some options can be given more

than once; the individual descriptions indicate which options may be specified multiple times. Option letters can be in either upper or lower case.

Command options that are used regularly may be placed in the environment variable **DSPLNKOPT**. If the variable is found in the environment the Linker adds the associated text to the existing command line prior to processing any options. See your host documentation for instructions on how to define environment variables.

Option arguments may immediately follow the option letter or may be separated from the option letter by blanks or tabs. However, an ambiguity arises if an option takes an optional argument. Consider the following command line:

DSPLNK -B MAIN IO

In this example it is not clear whether the file MAIN is an input file or is meant to be an argument to the **-B** option. If the ambiguity is not resolved the Linker will assume that MAIN is an input file and attempt to open it for reading. This may not be what the programmer intended.

There are several ways to avoid this ambiguity. If MAIN is supposed to be an argument to the **-B** option it can be placed immediately after the option letter:

DSPLNK -BMAIN IO

If there are other options on the command line besides those that take optional arguments the other options can be placed between the ambiguous option and the list of input file names:

DSPLNK -B MAIN **-V** IO

An alternative is to use two successive hyphens to indicate the end of the option list:

DSPLNK -B -- MAIN IO

In this latter case the Linker interprets MAIN as an input file name and uses the default naming conventions for the **-B** option.

1.4 LINKER OPTIONS

-A

Auto-align circular buffers. Any modulo or reverse-carry buffers defined in the object file input sections are relocated independently in order to optimize placement in memory. Code and data surrounding the buffer is packed to fill the space formerly occupied by the buffer and any corresponding alignment gaps.

Example: **DSPLNK -A** myprog.cln

Link the file MYPROG.CLN and optimally align any buffers encountered in the input.

-B[<objfil>]

This option specifies that an object file is to be created for Linker output. <objfil> can be any legal operating system filename, including an optional pathname. A hyphen also may be used as an argument to indicate that the object file should be sent to the standard output.

If a pathname is not specified, the file will be created in the current directory. If no filename is specified, or if the **-B** option is not present, the Linker will use the basename (filename without extension) of the first filename encountered in the input file list and append .CLD to the basename. If the **-I** option is present (see below) an explicit filename must be given. This is because if the Linker followed the default action it possibly could overwrite one of the existing input files. The **-B** option should be specified only once. **If the file named in the -B option already exists, it will be overwritten.**

Example: **DSPLNK -B**filter.cld main.cln fft.cln fio.cln

In this example, the files MAIN.CLN, FFT.CLN, and FIO.CLN are linked together to produce the absolute executable file FILTER.CLD.

-EA <errfil>
-EW <errfil>

These options allow the standard error output file to be reassigned on hosts that do not support error output redirection from the command line. <errfil> must be present as an argument, but can be any legal operating system filename, including an optional pathname.

The **-EA** option causes the standard error stream to be written to <errfil>; if <errfil> exists, the output stream is appended to the end of the file. The **-EW** option also writes the standard error stream to <errfil>; if <errfil> exists it is rewound (truncated to zero), and the output stream is written from the be-

ginning of the file. **Note that there must be white space separating either option from the filename argument.**

Example: **DSPLNK -EW** errors myprog.cln

Redirect the standard error output to the file ERRORS. If the file already exists, it will be overwritten.

-F<argfil>

Indicates that the Linker should read command line input from <argfil>. <argfil> can be any legal operating system filename, including an optional pathname. <argfil> is a text file containing further options, arguments, and filenames to be passed to the Linker. The arguments in the file need be separated only by some form of white space (blank, tab, newline). A semicolon (**;**) on a line following white space makes the rest of the line a comment.

The **-F** option was introduced to circumvent the problem of limited line lengths in some host system command interpreters. It may be used as often as desired, including within the argument file itself. Command options may also be supplied using the **DSPLNKOPT** environment variable. See the discussion of **DSPLNKOPT** under [options] at the beginning of this section.

Example: **DSPLNK -F**opts.cmd

Invoke the Linker and take command line options and input filenames from the command file OPTS.CMD.

-G

Send source file line number information to the object file. The generated line number information can be used by debuggers to provide source-level debugging.

Example: **DSPLNK -B -G** myprog.cln

Link the file MYPROG.CLN and send source file line number information to the resulting object file MYPROG.CLD.

-I

The Linker ordinarily produces an absolute executable file as output. When the **-I** option is given the Linker combines the input files into a single relocatable object file suitable for reprocessing by the Linker. No absolute addresses are assigned and no errors are issued for unresolved external references. Note that the **-B** option must be used when performing incre-

mental linking in order to give an explicit name to the output file. If the filename were allowed to default it could overwrite an existing input file.

Example: **DSPLNK -I -B**filter.cln main.cln fft.cln fio.cln

In this example, the files MAIN.CLN, FFT.CLN, and FIO.CLN are combined to produce the relocatable object file FILTER.CLN.

-L<library>

The Linker ordinarily processes a list of input files which each contain a single relocatable code module. If the **-L** option is encountered the Linker treats the following argument as a **library** file and **searches** the file for any outstanding unresolved references.

If a module is found in the library that resolves an outstanding external reference, the module is read from the library and included in the object file output. The Linker continues to search a library until all external references are resolved or no more references can be satisfied within the current library. The Linker searches a library only once, when it is encountered on the command line. Therefore, the position of the **-L** option on the command line is significant.

Example: **DSPLNK -B** filter main fir **-L**io

This example illustrates linking with a library. The files MAIN.CLN and FIR.CLN are combined with any needed modules in the library IO.LIB to create the file FILTER.CLD.

-M[<mapfil>]

This option indicates that a map file is to be created. <mapfil> can be any legal operating system filename, including an optional pathname. A hyphen also may be used as an argument to indicate that the map file should be sent to the standard output.

If a pathname is not specified, the file will be created in the current directory. If no filename is specified, the Linker will use the basename (filename without extension) of the first filename encountered in the input file list and append .MAP to the basename. If the **-M** option is not specified, then the Linker will not generate a map file. The **-M** option should be specified only once. **If the file named in the -M option already exists, it will be overwritten.**

Example: **DSPLNK -M** -- filter.cln gauss.cln

In this example, the files FILTER.CLN and GAUSS.CLN are linked together to produce a map file. Because no filename was given with the **-M** option, the output file will be named using the basename of the

first input file, in this case FILTER. The map file will be called FILTER.MAP.

-N

The Linker considers case significant in symbol names. When the **-N** option is given the Linker ignores case in symbol names; all symbols are mapped to lower case.

Example: **DSPLNK -N** filter.cln fft.cln fio.cln

In this example, the files FILTER.CLN, FFT.CLN, and FIO.CLN are linked to produce the absolute executable file FILTER.CLD. All symbol references are mapped to lower case.

-O<mem>[<ctr>][<map>]:<origin>

By default the Linker generates instructions and data for the output file beginning at absolute location zero for all DSP memory spaces. This option allows the programmer to redefine the start address for any memory space and associated location counter.

<mem> is one of the single-character memory space identifiers (X, Y, L, P). The letter may be upper or lower case. The optional <ctr> is a letter indicating the high (**H**) or low (**L**) location counters. If no counter is specified the default counter is used. <map> is also optional and signifies the desired physical mapping for all relocatable code in the given memory space. It may be **I** for internal memory, **E** for external memory, **R** for ROM, **A** for port A, and **B** for port B. If <map> is not supplied, then no explicit mapping is presumed.

The <origin> is a hexadecimal number signifying the new relocation address for the given memory space. The **-O** option may be specified as many times as needed on the command line. This option has no effect if incremental linking is being done (see the **-I** option).

Example: **DSPLNK -Ope:200** myprog -Lmylib

This will initialize the default P memory counter to hex 200 and map the program space to external memory.

-P<pathname>

When the Linker encounters input files, the current directory (or the directory given in the library specification) is first searched for the file. If it is not found and the **-P** option is specified, the Linker prefixes the filename (and optional pathname) of the file specification with <pathname> and searches the newly formed directory pathname for the file.

The pathname must be a legal operating system pathname. The **-P** option may be repeated as many times as desired. The directories will be searched in the order specified on the command line.

Example: **DSPLNK -P**\project\ testprog

This example uses IBM PC pathname conventions, and would cause the Linker to prefix any library files not found in the current directory with the \project\ pathname.

-Q

On some hosts the Linker displays a banner on the console when invoked. This option inhibits the banner display. It has no effect on hosts where the signon banner is not displayed by default.

Example: **DSPLNK -Q** myprog.cln

Link the file MYPROG.CLN but do not display the signon banner on the console.

-R[<ctlfil>]

This option indicates that a memory control file is to be read to determine the placement of sections in DSP memory and other Linker control functions. <ctlfil> can be any legal operating system filename, including an optional pathname.

If a pathname is not specified, an attempt will be made to open the file in the current directory. If no filename is specified, the Linker will use the basename (filename without extension) of the first filename encountered in the link input file list and append .CTL to the basename. If the **-R** option is not specified, then the Linker will not use a memory control file. The **-R** option should be specified only once.

Example: **DSPLNK -R**proj filter.cln gauss.cln

In this example, the files FILTER.CLN and GAUSS.CLN are linked together using the memory file PROJ.CTL.

-U<symbol>

Allows the declaration of an unresolved reference from the command line. <symbol> must be specified. This option is useful for creating an undefined external reference in order to force linking entirely from a library.

Example: **DSPLNK -U** start **-L**proj.lib

Declare the symbol START undefined so that it will be resolved by code within the library PROJ.LIB.

-V

This option causes the Linker to report linking progress (beginning of passes, opening and closing of input files) to the standard error output stream. This is useful to insure that link editing is proceeding normally.

Example: **DSPLNK -V** myprog.cln

Link the file MYPROG.CLN and send progress lines to the standard error output.

-X<opt>[,<opt>,...,<opt>]

The **-X** option provides for link time options that alter the standard operation of the Linker. The options are described below (* means default). All options may be preceded by **NO** to reverse their meaning. The **-X**<opt> sequence can be repeated for as many options as desired.

Option	Meaning
ABC*	Perform address bounds checking
AEC*	Check form of address expressions
ASC	Enable absolute section bounds checking
CSL	Cumulate section length data (see Chapter 3, SIZSYM Set Size Symbol)
ESO	Do not allocate memory below ordered sections
OVLP	Warn on section overlap
PSB*	Preserve sign bit in twos-complement negative operands
RO	Allow region overlap
RSC*	Enable relative section bounds checking
SVO	Preserve object file on errors
WEX	Add warning count to exit status

Example: **DSPLNK -X**WEX filter.cln fft.cln fio.cln

Have the Linker add the warning count to the exit status so that a project build will abort on warnings as well as errors.

-Z

The Linker strips source file line number and symbol information from the output file. Symbol information normally is retained for debugging purposes. This option has no effect if incremental linking is being done (see the **-I** option).

Example: **DSPLNK -Z** filter.cln fft.cln fio.cln

In this example, the files FILTER.CLN, FFT.CLN, and FIO.CLN are linked to produce the absolute object file FILTER.CLD. The output file will contain no symbol or line number information.

<filenames>

A list of operating system compatible filenames (including optional pathnames). If no extension is supplied for a given file, the Linker first will attempt to open the file using the filename as supplied. If that is not successful the Linker appends .CLN to the filename and attempts to open the file again. If no pathname is specified for a given file, the Linker will look for that file in the current directory. The list of files will be processed sequentially in the order given and all files will be used to generate the object file and map listing.

For more details on Linker operation in a particular machine environment see Appendix G, HOST-DEPENDENT INFORMATION, in the **Motorola DSP Assembler Reference Manual**.

Chapter 2
LINKER OPERATION

2.1 INTRODUCTION

Using a Linker allows the programmer to break up a large program into more manageable modules which may be assembled or compiled separately. These modules can then be link edited to produce an absolute module of the complete program. If a problem arises, only the module with the problem need be edited and reassembled. Then the programmer can relink the updated relocatable object module and the other previously created object modules to produce a new executable file.

2.2 RELOCATION AND LINKING

The input to the Linker is a set of relocatable object modules produced by the Motorola DSP assembler. The term **relocatable** means that the data in the module has not yet been assigned to absolute addresses in memory; instead, each different section is assembled as though it started at relative address 0 (an exception to this is absolute blocks, which do get assigned to absolute addresses at assembly time). When creating an absolute object module, it is the job of the Linker to read all the relocatable object modules which comprise a program and assign the relocatable blocks in each section to an absolute memory address. Then in the process of actually putting the code and data read from each object module into the proper location in the executable file, the Linker must fill in the correct addresses for such items as absolute addresses and references across sections. This is the process of relocation.

Along with relocation, the Linker performs resolution between modules, so that one module may reference symbols defined in a different module. At assembly time the module doing the referencing has no idea where the symbol it is referencing will be in the final absolute module. Therefore, the assembler sets up information in the relocatable object module which indicates that an external symbol is referenced in this module and where the symbol is referenced. In the relocatable object module where the symbol is defined there is information indicating that this is the module in which the symbol is defined, along with the value of the symbol in the module. When the modules are presented as input to the Linker, the correct value of the symbol can be inserted wherever it is referenced.

If an external reference is made to a symbol for which there is no corresponding record in the input, the Linker flags it as an unresolved external reference. No final values are as-

signed to these references, and the resulting output file is unusable. A list of unresolved references is sent to the Linker's standard output and to the optional link map file.

References in the input file may be specified as either absolute or relative expressions. An absolute expression is one which consists only of absolute terms, or is the difference between two relative terms. A relative expression consists of one relative term along with absolute terms and/or the result of two relative terms with opposing signs. Expressions in the input file are a modified notation as supported by the assembler. See Appendix E in the **Motorola DSP Assembler Reference Manual** for more information on the format of relocatable object file expressions.

2.3 LINKER PASSES

The Linker makes two partial passes over the input data. During the first pass, it collects section, symbol, and external reference information from each input file given on the command line. If the input file is a library, the Linker checks to see if there are any external references outstanding. If there are, the Linker opens the library file and searches each module in the library until all external references are resolved or no more references can be satisfied within that library. If there are no outstanding unresolved references, the Linker skips the library. At the end of the first pass a list of unresolved external references is sent to the standard output as well as to the map file if one exists. References to unresolved symbols may be fixed up using the **SYMBOL** directive of the memory control file, discussed in Chapter 3.

Prior to the second pass, the Linker scans its internal tables and performs fixups on section start addresses and symbol values. This includes setting the base relocation address for any memory spaces and counters as given by the **-O** option on the command line or the **BASE** directive in the memory control file. If a memory control file was specified on the command line it is opened and read to determine placement of any named sections in memory.

Blocks of code and data are arranged in memory first by region then by location counter assignment. Absolute sections are located first, followed by ordered sections, and then any remaining sections are placed in memory. It is possible that addresses assigned to a section using a numbered location counter might overlap addresses of another section using a different counter. This design is intentional so that counters may be used as a logical connection to the physical mapping of separate memories (e.g. internal and external RAM), or as a means for supporting load and runtime counters for overlays.

During the second pass, the Linker processes the data records, evaluating data fields as expressions and writing the modified values to the output file. Errors are reported during either pass, and the Linker may abort depending on the severity of the error. Linker errors are routed to standard output so they may be redirected to a file if necessary. An output file produced with errors should not be used in any case. The number of errors is returned as an exit status when the Linker returns control to the host operating system.

2.4 LINKING WITH REGIONS AND SECTIONS

The basic relocatable entity is the **section**. Sections are created by the assembler **SECTION** directive. A section may contain code or data from any DSP memory space, and the addresses of the code or data may be relocatable or absolute. Sections that are not absolute at assembly time can be located either directly using the Linker memory control file **SECTION** directive, or indirectly via the Linker command line **-O** option or the control file **BASE** directive. Note that sections do not have to be relocatable to be linked; an absolute section can be linked with any arbitrary module that contains or satisfies a reference to that section.

Sections may be grouped for relocation into **regions**. A region is a defined area of memory where sections are located. Regions make it possible to specify varying base addresses for different groups of sections and set boundaries for section growth. Whereas sections represent blocks of code or data which are positioned within a given memory map, regions designate an area of a specified size where sections can be placed.

There is always at least one default region. Other regions are defined using the Linker **REGION** directive. A region always has a size and a start and end address. If these are not given, the Linker uses default values (e.g. if no end address is supplied for a region, the highest target address is used).

Regions should not overlap. One exception is that regions may overlap for setting overlay base addresses. Another exception is the default region, which allows explicit regions to supersede it for relocation blocks. After all sections in explicit regions are located, the Linker relocates remaining sections around the previously assigned blocks within the default region. This behavior can be altered with the Linker **RO** option (see section 1.3).

2.5 LINKING WITH CIRCULAR BUFFERS

A **circular buffer** is a fixed area of memory manipulated via special-purpose DSP addressing modes. Because of the way buffers are accessed, they must be suitably aligned on an address boundary. When the programmer declares a modulo or reverse-carry buffer, the assembler aligns the buffer block at an address corresponding to the size of the buffer. The alignment may create a padding gap comprising the locations skipped to properly position the buffer block in memory.

The Linker processes buffer blocks in one of two ways. By default, it keeps track of the largest buffer in any section and aligns the entire section based on the size of the largest buffer block. This insures that any smaller buffers contained in the section will remain aligned after relocation, but it can introduce additional padding gaps because of the section alignment.

If the **-A** option is given on the Linker command line, or if one of the buffer alignment directives is specified in the memory control file, the Linker *auto-aligns* buffers. It extracts the buffer blocks in a given section, locates them in memory before any other relocatable blocks, and repositions the section code and data to fill in gaps left from padding and re-

located buffers. The Linker sorts the buffers, placing the largest blocks in memory first in order to make more eligible addresses available for subsequent smaller blocks.

Auto-alignment works only with relocatable buffers; the Linker will not attempt to realign any absolute block. Also, a buffer defined inside a relocatable overlay cannot be auto-aligned because the assumption is that the overlay block will move, invalidating any optimal placement of the buffer. If a buffer is declared using an open-ended alignment directive such as **BADDR**, the Linker will not auto-align any buffers within that section associated with the current memory space and counter. This is because the Linker has no knowledge of how far the open-ended block extends, and since alignment works only at the section level, the Linker must abandon auto-alignment of all buffers in the section. See Chapter 4 in the **Motorola DSP Assembler Reference Manual** for more information on circular buffers.

2.6 LINKING WITH OVERLAYS

An **overlay** is a segment of code or data that is loaded at one address, but is moved and executed or used at another location. A good example is a user program burned into PROM and transferred into internal RAM by the DSP bootstrap program. The Linker handles overlays by recognizing overlay blocks, reconciling overlay block addresses with previously relocated sections, and altering values for symbols associated with overlay blocks.

Processing of any overlays is postponed until after all absolute and otherwise relocatable sections have been placed into the memory map. This is done so that any overlays based on an explicit address (e.g. a relocatable expression) will be properly located. If overlays exist that were not explicitly based (default overlays), the Linker attempts to base them from previous explicit blocks. If there are no explicit blocks, the Linker will base the default overlays from the enclosing section or region base address. In all cases for default overlays, the blocks will be located as if they were contiguous; that is, default overlays will not overlap one another.

After all overlay blocks are processed the Linker resolves overlay symbols. Overlay symbols are those labels defined inside an overlay within the source file. The overlay block information is retained for reporting to the link map file. See Chapter 4 in the **Motorola DSP Assembler Reference Manual** for more information on overlays.

Chapter 3
LINKER DIRECTIVES

3.1 MEMORY CONTROL FILE

A memory control file is simply a text file containing Linker directives. It optionally contains module identification, a global starting load address for linking purposes, and ordering, sizing, or placement information for any named sections. Section addresses may be for any memory space and any logical location counter. The memory control file also can specify physical memory mappings (internal, external) associated with any memory space or counter. In addition, global unresolved symbols may be assigned values in the memory control file.

3.2 LINKER DIRECTIVE DESCRIPTIONS

Linker directives are commands which control the operation of the Linker with respect to section relocation, buffer alignment, symbol definition, and map file format. Linker directives are listed below:

BALIGN	**MAP**	**SBALIGN**	**SIZSYM**
BASE	**MEMORY**	**SECSIZE**	**START**
IDENT	**REGION**	**SECTION**	**SYMBOL**
INCLUDE	**RESERVE**	**SET**	

Several of the directives use the notation **mem** or **memx** to indicate the contents of a field. The definitions of **mem** and **memx** are as follows:

mem	=	<attr>:<exp>
memx	=	<attr>:<exp>..<exp>
attr	=	<scm> | <sme>
scm	=	<spc>[<ctr>][<map>]
sme	=	<spc>[<map>][(exp)]
spc	=	X | Y | L | P
ctr	=	L | H
map	=	I | E | R | A | B
exp	=	expression

The **spc** field indicates one of the DSP memory spaces (X, Y, L, P). The **ctr** field specifies either **L**ow or **H**igh location counters; if none is given the default counter is used. Alternatively, an expression in parentheses may be provided to indicate an arbitrary counter designation. The **map** field indicates **I**nternal memory, **E**xternal memory, **R**OM, port **A**, or port **B**; this field may be omitted, in which case no explicit mapping is done.

BALIGN
Auto-align Circular Buffers

BALIGN <mem>[,...,<mem>]

The **BALIGN** directive auto-aligns circular buffers within a particular region. All relocatable buffers found in any section within the region are relocated independently for optimal placement in memory. Code and data around the buffer is made contiguous in order to fill in previously occupied space. The <mem> argument indicates where in memory the alignment should begin.

Example:

```
BALIGN      XE:$200,YE:$200    ; Realign X and Y external buffers
```

BASE
Set Region Base Address

BASE <mem>[,...,<mem>]

The **BASE** directive indicates where to begin the location counter for the given memory region. This will be the base link address for all specified memory areas and all linked code and data within the region except for sections relocated absolutely via a memory file **SECTION** directive. Code and data not explicitly relocated will originate from this address. The **BASE** directive is analogous to the Linker **-O** command line option.

Example:

```
BASE        XE:$200,YE:$200,PI:$200 ; Set memory base addresses
```

IDENT
Object Module Identification

IDENT <module name> <version> <revision> [;<comment>]

The **IDENT** directive functions similarly to the assembler **IDENT** directive by identifying the name, version number, and revision number of the absolute or incrementally linked output module. The information is sent to the resulting output file. The <module name> adheres to the rules for assembly language labels, so that it must begin with an alphabetic character and consist only of alphanumeric characters or the underscore up to a length of 255. The version number and revision number must be absolute expressions. If a comment follows the version and revision numbers it will be copied into the output file as well.

Example:

```
IDENT   MYMODULE   1   2   ; MYMODULE, version 1, revision 2
```

INCLUDE
Include Directive File

INCLUDE <filename>

The **INCLUDE** directive provides for insertion of separate files containing other Linker control directives. File inclusion can be convenient for always including a set of master directives in several different configuration files. <filename> must be in quotes.

Example:

```
INCLUDE         'main.ctl'      ; Include master control file
```

MAP PAGE
Map File Format Control

MAP **PAGE** <exp1>[,<exp2>[,<exp3>[,<exp4>[,<exp5>]]]]

The **MAP PAGE** modifier works similarly to the assembler **PAGE** directive, and causes the .MAP file to be printed on the page according to the parameters supplied. If no **MAP PAGE** appears in the memory control file, the Linker produces a map file with a column width of 80, a physical page length of 66 lines, and no blank lines at top and bottom.

Example:

```
MAP     PAGE    132,,3,3
```

The above **MAP PAGE** directive indicates a column width of 132, a physical page length of 66 lines (default), with three blank lines at the top and bottom of each page.

MAP OPT
Map File Contents Control

MAP **OPT** <option>[,<option>,...,<option>]

The **MAP OPT** modifier determines the content of the output in the Linker map file. The following **MAP OPT** options are available:

GLOBMAP	- produce global map by memory space
NOCONST	- do not list symbols without a memory space attribute
NOGLOBSYM	- omit symbols from global map
NOLOCAL	- do not list non-global symbols (e.g. symbols which are local to a section)
NOSECADDR	- do not list sections by address
NOSECNAME	- do not list sections by name
NOSYMNAME	- do not list symbols by name
NOSYMVAL	- do not list symbols by value
NOUNUSED	- do not list unused memory blocks

If no **MAP OPT** is found in the memory control file, the Linker will list all symbols and sections by name, address, and value.

Example:

```
MAP     OPT     NOCONST,NOSYMVAL
```

The above **MAP OPT** directive specifies no constants in the map listing and no symbols by value.

MEMORY
Set Region High Memory Address

MEMORY <mem>[,...,<mem>]

The **MEMORY** directive establishes a maximum high memory address for locating code and data in the given memory region. If the Linker attempts to relocate a block beyond the address specified in the **MEMORY** directive, an error will occur. This directive is useful for reflecting the true physical memory limits of the target system.

Example:

```
MEMORY          PE:$1FFF    ; External program memory ends at hex 1FFF
```

REGION
Establish Memory Region

```
REGION      <region>      [<mem>[,...,<mem>]]
   .
   .
   .
ENDR
```

The **REGION** directive defines a region of memory in which to locate sections. The region name identifies the region. The optional <mem> parameter gives an absolute region size. The **REGION** directive is used in conjunction with existing control directives to specify a bounds for placing sections in memory. For example, a **BASE** directive used within a **REGION/ENDR** pair defines the base address for *that region only*. Likewise a **MEMORY** directive within a **REGION** scope indicates the high address for the enclosing region.

Example:

```
REGION      INTERNAL_ROM    X:$256,Y:$256
BASE        X:0,Y:0         ; Base for INTERNAL_ROM region only
SECTION     BUFFERS
ENDR
```

RESERVE
Reserve Memory Block

RESERVE <memx>[,...,<memx>]

The **RESERVE** directive sets aside a block of memory which the Linker will not use for relocation. The expression field in the <mem> parameter takes the form of a range **n..m**, where **n** is the low reserve address and **m** is the high reserve address. This directive can be used to protect ROM locations, system code, or uninitialized buffer areas.

Example:

```
RESERVE     PI:$0..$1FF     ; Protect interrupt vectors
```

SBALIGN
Auto-align Section Buffers

SBALIGN <section> <mem>[,...,<mem>]

The **SBALIGN** directive auto-aligns circular buffers within a named section. All relocatable buffers found in the section are relocated independently for optimal placement in memory. Code and data around the buffer is made contiguous in order to fill in previously occupied space. The <mem> argument indicates where in memory the alignment should begin.

Example:

```
SBALIGN     MYSEC     XE:$200     ; Realign X external buffers
```

SECSIZE
Pad Section Length

SECSIZE <section> [<mem>[,...,<mem>]]

The **SECSIZE** directive provides a mechanism for padding a section to a particular length despite its code or data content. The value field in the **mem** parameter is an expression which can either be an absolute size expressed as an integer, or a floating point value representing a percentage to pad.

Example:

```
SECSIZE     PADSEC    X:$1000,Y:$1000    ; X and Y absolute size
SECSIZE     PADSEC    P:150.0            ; Increase size by one half
```

SECTION
Set Section Base Address

SECTION <section> [<mem>[,...,<mem>]]

The **SECTION** directive either assigns a section of code or data to an absolute location in DSP memory, or implies an ordering if no address specification is present. The addresses serve as the base for the corresponding memory spaces and counters in the named section. Any memory areas not indicated in the **SECTION** directive are relocated relative to the global starting load address given by the **-O** command line option or the memory file **BASE** directive. If there is no **-O** option or **BASE** directive, unassigned areas are placed in memory relative to location zero.

If the **SECTION** directive appears with only a section name and no address, it means that the Linker should locate this section in memory before handling any other default sections. Thus given a set of sections A, B, C, and D, if B and C were listed in **SECTION** directives without a corresponding address, the Linker would place B and C in memory before A and D. This provides a means for ordering sections in memory.

Example:

```
SECTION     ABS   X:$2000,Y:$2000   ; X and Y absolute base
SECTION     ORD                     ; Ordered section
```

SET
Set Symbol Value

SET <symbol> { <mem> | <expression> }

The **SET** directive is a synonym for the **SYMBOL** directive, described below. It is useful for sharing counter declaration files between the assembler and Linker since the syntax is compatible.

Example:

```
SET     PCTR    5       ; Set P memory counter number
```

SIZSYM
Set Size Symbol

SIZSYM <symbol> <attr>:[<section>]

The **SIZSYM** directive makes it possible to declare an arbitrary symbol to hold section length data. This is useful when the programmer needs a cumulative section length for overlay handling. It is analogous to the **SYMBOL** directive, in that the symbol so defined may be employed to resolve an external reference in a source file.

If no section name is supplied the Linker returns the length of all sections for the memory space and counter specified. Ordinarily the Linker retains length data only for relocatable sections; use **CSL** (see **-X** option) to cumulate length data for absolute and buffer sections as well. Note that **SIZSYM** symbols are valued after the memory control file is read, so that attempting to reference the symbol value within the memory control file itself may cause erroneous results.

Example:

```
SIZSYM      XLEN      X:      ; Assign length of X memory to XLEN
```

START
Establish Start Address

START <expression>

The **START** directive gives an alternative start address to which the program will jump at runtime. This value is ordinarily given by the assembler **END** directive. The expression may consist of an absolute value or a global symbol whose value will be adjusted during link processing.

Example:

```
START   BEGIN                   ; Jump to location BEGIN after loading
```

SYMBOL
Set Symbol Value

SYMBOL <symbol> { <mem> | <expression> }

The **SYMBOL** directive allows the programmer to specify a value for an otherwise unresolved reference. The named symbol must not have been defined during link processing. The symbol is stored as an absolute global symbol. The symbol value may be either integer or floating point. If the value is an address it may contain a memory space reference and optionally a counter and mapping designation.

Example:

```
SYMBOL      TARGET    X:$200      ; Set TARGET to hex 200
```

3.3 Memory Control File Example

Figure 1 shows the contents of an example memory control file. The **IDENT** directive identifies the object module and gives it explicit version and revision numbers. The comment is also preserved in the output file. The **START** directive gives a starting address of FILTER for the program, overriding any previous settings done with the assembler **END** directive.

The **BASE** directive indicates that the X and Y low memory counters are to be mapped into internal DSP memory, with a starting address of 100 hexadecimal. Any data associated with the X or Y low memory counters, and not relocated due to a subsequent memory file **SECTION** directive, will be assigned addresses relative to this starting location. The **BASE** directive also shows that X and Y high memory counters have been assigned starting address 2000 hexadecimal in external DSP memory, and that linking to external program memory begins at location 200 hexadecimal. Note that any memory specifications given by the **-O** command line option override the values supplied by the memory file **BASE** directive for the default region.

The **RESERVE** directives set aside a part of the low internal X and Y data memory, even though the base address is lower than the reserved area. The Linker will locate data around the reserved portions as if they had been previously allocated.

The **REGION** directive defines a sized region of memory for modulo buffers in internal ROM. The corresponding **MEMORY** directive indicates that there are only 256 words of memory in each data space for this region (the default base is zero).

The example **SECTION** directives are similar to the format of the **BASE** directive, except that the particular section is named so that the individual section counters may be modified. For the section named INPUT, the program low memory counter is initialized to 100 hex and mapped to external memory. The program memory for the FILTER section uses the default location counter and sets the initial value to 400 hex, mapped to external memory. Finally, the OUTPUT section is set to 800 hex, using the high memory P space counter mapped to external memory.

Two unresolved symbols are given values with the **SYMBOL** directive. The symbol XDATA is assigned to external high X memory with a value of 2000 hexadecimal. The symbol YDATA is assigned to external high Y memory with a value of 2000 hexadecimal. Both symbols will be stored as absolute global entities.

The **MAP** directives control the formatting and content of the link map file. The first directive sets the page width to 132, with three blank lines at top and bottom. The second directive disables the reporting of sections by address and symbols by value.

```
ident       filter   2       1       ; Filter module
start       filter

base        xli:$100,xhe:$2000,yli:$100,yhe:$2000,pe:$200
reserve     xli:$200..$3ff,yli:$200..$3ff

region      internal_rom
memory      x:256,y:256
section     buffers
endr

section     input   ple:$100
section     filter  pe:$400
section     output phe:$800

symbol      xdata  xhe:$2000
symbol      ydata  yhe:$2000

map         page        132,,3,3
map         opt         nosecaddr,nosymval
```

Figure 1. DSP Linker Memory Control File Example

Chapter 4
Motorola DSP Librarian

4.1 INTRODUCTION

The Motorola DSP Librarian is a stand-alone utility that allows separate files to be grouped together into a single file for linking or archival storage. After a library is created, files may be added, deleted, replaced, or extracted from the library. The library contents may also be listed, indicating the module name (base name of the input file path), size in bytes, and the date and time the module was entered into the library.

4.2 INSTALLING THE LIBRARIAN

The librarian is distributed on various media and in different formats depending on the host operating system environment. See Appendix G in the **Motorola DSP Assembler Reference Manual**, HOST-DEPENDENT INFORMATION, for details on installing and operating the librarian on your particular machine.

4.3 RUNNING THE LIBRARIAN

The general format of the command line to invoke the librarian is:

DSPLIB [options] [<library>] [<files>]

where:

[options]

Any **one** of the following command line options. The single option must precede the library name. Option letters may be specified in either upper or lower case. If no option is supplied, the librarian operates as if the **update** (**-U**) option were given.

4.4 LIBRARIAN OPTIONS

-A

This option adds the modules in the file list to the named library. The library file must exist, and the modules must not already be in the library.

Example: **DSPLIB -A** fftlib fft16.cln fft512.cln ditfft.cln

In this example, the files FFT16.CLN, FFT512.CLN, and DITFFT.CLN are added to the existing library FFTLIB.LIB.

-C

Create a new library file and add any specified modules to it. If the library file already exists, an error is issued.

Example: **DSPLIB -C** fftlib fft16.cln fft512.cln ditfft.cln

In this example, a new library file FFTLIB.LIB is created and the files FFT16.CLN, FFT512.CLN, and DITFFT.CLN are added to the library.

-D

Delete the named modules from the library. If the module is not in the library, an error is issued.

Example: **DSPLIB -D** fftlib fft16.cln

In this example, the module FFT16.CLN is removed from the library FFTLIB.LIB.

-EA<argfil>
-EW<argfil>

These options allow the standard error output file to be reassigned on hosts that do not support error output redirection from the command line. <errfil> must be present as an argument, but can be any legal operating system filename, including an optional pathname.

The **-EA** option causes the standard error stream to be written to <errfil>; if <errfil> exists, the output stream is appended to the end of the file. The **-EW** option also writes the standard error stream to <errfil>; if <errfil> exists it is rewound (truncated to zero), and the output stream is written from the beginning of the file.

Example: **DSPLIB -EW**errors **-A** fftlib fft16.cln fft512.cln ditfft.cln

Redirect the standard error output to the file ERRORS. If the file already exists, it will be overwritten.

-F<argfil>

Indicates that the librarian should read command line input from <argfil>. <argfil> can be any legal operating system filename, including an optional pathname. <argfil> is a text file containing module names to be passed to the librarian. The arguments in the file need be separated only by some form of white space (blank, tab, newline). A semicolon (**;**) on a line following white space makes the rest of the line a comment.

The **-F** option was introduced to circumvent the problem of limited line lengths in some host system command interpreters.

Example: **DSPLIB -F**opts.cmd

Invoke the librarian and take command line filenames from the command file OPTS.CMD.

-L

List library contents. This option lists the module name as contained in the library header, the module size (less library overhead), and the date and time the file was stored into the library. The listing output is routed to standard output so that it may be redirected to a file if desired.

Example: **DSPLIB -L** fftlib > fftlib.lst

This example lists the contents of the library FFTLIB.LIB. The output is redirected to the file FFTLIB.LST.

-Q

On some hosts the librarian displays a banner on the console when invoked. This option inhibits the banner display. It has no effect on hosts where the signon banner is not displayed by default.

Example: **DSPLIB -AQ** mylib.clb myprog.cln

Add the file MYPROG.CLN to the library MYLIB.CLB, but do not display the signon banner.

-R

This option replaces the named modules in the given library. The modules must already be present in the library file.

Example: **DSPLIB -R** fftlib fft512.cln ditfft.cln

This example replaces the files FFT512.CLN and DITFFT.CLN in the library FFTLIB.LIB.

-U

This option updates the specified modules if they exist in the library; otherwise it adds them to the end of the library file.

Example: **DSPLIB -U** fftlib ditfft.cln

In this example, the file DITFFT.CLN is updated in the library FFTLIB.LIB.

-V

Display the librarian version number and copyright notice on standard output.

Example: **DSPLIB -V**

This example displays the current librarian version number and copyright notice.

-X

Extract named modules from the library. The resulting files are given the name of the modules as stored in the library module header.

Example: **DSPLIB -X** fftlib fft16.cln fft612.cln

This example extracts the files FFT16.CLN and FFT512.CLN from the library FFTLIB.LIB. The files are placed in the current directory.

<library>

An operating system compatible filename (including optional pathname) specifying the library file to create or access. If no extension is supplied, the librarian will automatically append .LIB to the filename. If no pathname is specified, the librarian will look for the library in the current directory.

The librarian also has an interactive mode, where commands can be entered repeatedly without reloading the librarian program for each operation. If the librarian is invoked without arguments, it prompts for a command string. The interactive commands correspond to those given above, and the syntax is similar to that of the command line. Because interactive input is taken from the standard input channel of the host environment, it is possible to create a batch of librarian commands and feed them to the program for execution via redirection. Enter **help** or **?** at the prompt for more information on the librarian interactive mode.

<files>

A list of operating system compatible filenames separated by blanks. If no pathname is specified for a given file, the librarian will look for that file in the

current directory. For input operations the filenames may also contain an optional pathname; the path is stripped when the file is written to the library. For output operations only the filename should be used to refer to library modules. The list of files will be processed sequentially in the order given.

4.5 LIBRARY PROCESSING

A library file may contain several relocatable object modules, each of which contains one or more global symbol definitions. Rather than being normal input to the Linker, a library file is searched. This means that for each relocatable object module in the library, a check is made to determine whether any globally defined symbols in the library module match any externally referenced symbols encountered in previous input modules. If so, the relocatable object module from the library is included in the executable file. If not, the search continues with the next module in the library file.

Chapter 5
Motorola DSP S-record Conversion Utility (SREC)

5.1 INTRODUCTION

The Motorola DSP S-Record Conversion Utility SREC converts Motorola DSP COFF format files into Motorola S-record files. The S-record format was devised for the purpose of encoding programs or data files in a printable form for transportation between computer systems. Motorola S-record format is recognized by many PROM programming systems.

5.2 INSTALLING SREC

SREC is distributed on various media and in different formats depending on the host operating system environment. See Appendix G in the **Motorola DSP Assembler Reference Manual**, HOST-DEPENDENT INFORMATION, for details on installing and operating SREC on your particular machine.

5.3 RUNNING SREC

The general format of the command line to invoke SREC is:

SREC [options] <files>

where:

[options]

Any of the following command line options. The options must precede the file names. Option letters may be specified in either upper or lower case.

5.4 SREC OPTIONS

-A<alen>

Use <alen> as the S-record address length. A value of 2 indicates a two-byte address and will generate S1 records. A value of 3 indicates a three-byte address and will generate S2 records. A value of 4 indicates a four-byte address and will generate S3 records. This option overrides any S-record address length implied by the processor type. Address truncation

may occur for targets with address ranges greater than what <alen> can accommodate.

Example: **SREC -A**4 prog

The file PROG.CLD is translated to S-records using the S3-S7 record format.

-B

Use byte addressing when transferring load addresses to S-record addresses. This means that object file start addresses are multiplied by the number of bytes per target DSP word and subsequent S1/S3 record addresses are computed based on the data byte count.

Example: **SREC -B** prog

In this example, the file PROG.CLD is translated to S-record format using byte addressing. The load addresses will be multiplied by the number of bytes per DSP word. A separate output file will be produced for each DSP memory space (X, Y, L, P, or E) represented in the input file.

-L

Use double-word addressing when transferring load addresses from L space to S-record addresses. This means that object file records for L space data are moved unchanged and subsequent S1/S3 record addresses are computed based on the data word count divided by 2. This option should always be used when the object file contains sections in L memory space.

Example: **SREC -L** filter.cld

Convert the file FILTER.CLD into separate S-record files for each memory space in the object file. Convert L space load addresses to long addresses in the S-record file.

-M

Split each DSP word into bytes and store the bytes in parallel S-records. The **-M** and **-S** options are mutually exclusive.

Example: **SREC -M** main

For each memory space in the file MAIN.CLD create multiple S-record files that correspond to each byte in the target DSP word. For example, if MAIN.CLD contained only references to P memory and the target DSP is the DSP56000, then SREC would produce the files MAIN.P0, MAIN.P1, and MAIN.P2.

-O<mem>:<offset>

Add <offset> to S-record addresses in <mem> memory space. <mem> is one of the valid memory space specifiers: X, Y, L, P, or E. <offset> must be given in hexadecimal.

Example: **SREC -O**P:100 prog

The file PROG.CLD is translated to S-record format with the value 100 hexadecimal added to all P memory addresses.

-P<procno>

Assume <procno> object file format. This makes a difference in the type of S-type data records produced. <procno> is one of the Motorola DSP processor numbers, e.g. 56000, 96000, etc. This option overrides the object file machine ID. It is useful for handling object files from programs that do not generate target machine information.

Example: **SREC -P**56000 prog

The file PROG.CLD is translated to S-record format with the assumption that the target processor is in the DSP56000 family of processors.

-Q

On some hosts SREC displays a banner on the console when invoked. This option inhibits the banner display. It has no effect on hosts where the signon banner is not displayed by default.

Example: **SREC -Q** myprog.cld

Translate the file MYPROG.CLD to S-record format but do not display the signon banner on the console.

-R

Write bytes high to low, rather than low to high. This option has no effect when used with the **-M** option.

Example: **SREC -R** prog

The file PROG.CLD is translated to S-record format with bytes written high to low. A separate output file will be produced for each DSP memory space (X, Y, L, P, or E) represented in the input file.

-S

Write data to a single file, putting memory space information into the address field of the S0 header record. The **-M** and **-S** options are mutually exclusive.

Example: **SREC -S** filter

This example writes the S-record output to a single file called FILTER.S and stores the memory space information in the address field of the S0 header record. An S0 record is emitted whenever the memory space changes in the object file.

-T<tlen>

Use <tlen> as the target word length. A value of 2 indicates a two-byte word length. A value of 3 indicates a three-byte word length. A value of 4 indicates a four-byte word length. This option overrides any target word length implied by the processor type, and therefore may lead to value padding or truncation.

Example: **SREC -T**4 prog

The file PROG.CLD is translated to S-records using four-byte data words.

-U

Write words high to low, rather than low to high when processing L memory data records. This option has no effect when used with the **-X** option.

Example: **SREC -U** prog

The file PROG.CLD is translated to S-record format with L memory words written high to low. A separate output file will be produced for each DSP memory space (X, Y, L, P or E) represented in the input file.

-W

Use word addressing when transferring load addresses to S-record addresses. This means that object file start addresses are moved unchanged and subsequent S1/S3 record addresses are computed based on the data word count.

Example: **SREC -W** main

In this example, the file MAIN.CLD is translated to S-record format using word addressing. A separate output file will be produced for

each DSP memory space (X, Y, L, P, or E) represented in the input file.

-X

Split L memory input words into respective X and Y data records. This option has no effect when used with the **-U** option.

Example: **SREC -X** prog

The file PROG.CLD is translated to S-record format with L memory words translated to equivalent X and Y data values. A separate output file will be produced for each DSP memory space (X, Y, L, P, or E) represented in the input file.

<files>

A list of operating system compatible filenames separated by blanks. If no pathname is specified for a given file, SREC will look for that file in the current directory. If the special character '**-**' is used as a filename SREC will read from the standard input stream. The list of files will be processed sequentially in the order given.

5.5 SREC PROCESSING

SREC takes as input a Motorola DSP absolute object file and produces byte-wide Motorola S-record files suitable for PROM burning. The Motorola DSP COFF file header records are mapped into S0 and S7/S8/S9 records respectively. DSP COFF section raw data are mapped into S1, S2, or S3-type records depending on the magnitude of the address value or on the type of the target processor.

Since Motorola DSPs use different word sizes, the words must be split into bytes and stored in a suitable format. The program keeps track of the input address magnitude to determine the appropriate S-record format to generate. If the **-A** or **-P** option is selected, SREC uses a format corresponding to the address size or processor type specified. For example, if the programmer entered a **-P**96000 option, SREC would always produce S3/S7 records regardless of the input address size.

In the default mode of operation the program writes the low, middle, and high bytes of each word consecutively to the current S1/S2/S3 record being written. For example, given the DSP56000 raw data record below:

```
0008F8 300000 340000 094E3E
|      |      |      |
|      |      |      fourth word
|      |      third word
|      second word
first word
```

SREC would create the following S1 record:

```
    byte count field
    |   address  field                          checksum field
    |   |                                       |
S10D0000F8080000003000003 43E4E09F9
          |      |      |      |
          |      |      |      fourth word
          |      |      third word
          |      second word
          first word
```

Output records are written to a file named according to the following convention:

<basename>.<M>

where <*basename*> is the filename of the input object file without extension and <*M* > is the memory space specifier (X, Y, L, or P) for this set of data words. Note that a separate file is created for each memory space encountered in the input file; thus the maximum number of output files in the default mode is 4.

When the **-M** option is specified, SREC splits each DSP source word into bytes and stores the bytes in parallel S1/S2/S3 records. For example, the following DSP56000 raw data:

```
0008F8 300000 340000 094E3E
|      |      |      |
|      |      |      fourth word
|      |      third word
|      second word
first word
```

would be converted by SREC into the three S1 records below:

```
    byte count field
    |   address  field
    |   |
S1070000F800003EC2   --  low  byte
S10700000800004EA2   --  mid  byte
S1070000003034098B   --  high byte
          |   | |   | |
          |   | |   | checksum field
          |   | |   fourth word
          |   | third word
          |   second word
          first word
```

The three records corresponding to the high, middle, and low bytes of each data word are written to separate files. The files are named according to the following convention:

<basename>.<M><#>

where <*basename*> is the filename of the input object file without extension, <*M*> is the memory space specifier (X, Y, L, P, or E) for this set of data words, and <*#*> is one of the digits 0, 1, or 2 corresponding to low, middle, and high bytes, respectively.

Note that a separate set of byte-wide files is created for each memory space encountered in the input file. Thus the number of output files generated is (number of memory spaces in input * size of DSP word).

The **-S** option writes all information to a single file, storing the memory space information in the address field of the S0 header record. The values stored in the address field and their correspondence to the DSP memory spaces are as follows:

Value	DSP Memory Space
1	X
2	Y
3	L
4	P
5	E

When the memory space changes in the object file section record, a new S0 header record is generated. The resulting output file is named **<basename>.S**, where <base-

name> is the filename of the input object file without extension. The **-M** and **-S** options are mutually exclusive.

Address fields in DSP section records are copied as is to the appropriate S1, S2, or S3 record. Subsequent S1, S2, or S3 record addresses are byte incremented until a new section is encountered or end-of-file is reached. In some cases the starting S1/S2/S3 record address must be adjusted for byte addressing by multiplying the section start address by the number of bytes in a DSP word. When the **-B** option is given, any section address fields are adjusted to begin on a byte-multiple address. If the **-W** option is specified (the default) byte-incrementing is not done when generating S-record addresses, e.g. the S-record addresses are word-oriented rather than byte-oriented. The **-B** and **-W** options have no effect when used in conjunction with the **-M** mode, since in that case byte and word address mappings are 1:1.

Section records for L space memory contain words which are loaded into adjacent X and Y memory locations. In these cases performing the default strict word addressing may be inappropriate. The **-L** option may be given to indicate that double-word addressing should be used to generate subsequent S1/S2/S3 addresses after the initial load address. In addition the **-L** option should be used when doing byte addressing since the initial load addresses must be adjusted to account for double-word addressing in the object file. In general, it is a good idea to use the **-L** option whenever the input object file contains sections which refer to L memory space.

5.6 S-RECORD FILE FORMAT

An S-record file consists of a sequence of specially formatted ASCII character strings. These character strings are made up of several fields which identify the record type, record length, memory address, code or data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number, the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

5.6.1 S-Record Content

An S-record consists of 5 distinct fields: the **TYPE** field, the **RECORD LENGTH**, **ADDRESS** field, **CODE/DATA**, and the **CHECKSUM** field.

Field	Printable Characters	Contents
Type	2	S-record type: S0, S1, etc.
Record length	2	The count of the character pairs in the record, excluding the type and record length.
Address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/data	0-2n	From 0 to *n* bytes of executable code, memory loadable data, or descriptive information.
Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

5.6.2 S-Record Types

There are ten possible Motorola S-record types. The following sections discuss the S-record types used by the Motorola DSP SREC utility.

5.6.2.1 S0 Record

The S0 record is the header record (sometimes called the 'sign-on' record) for a block of Motorola S-records. The address field is always 4 printable characters representing a 2-byte address. It is normally zero, but when the SREC **-S** option is used, the program generates a code corresponding to the DSP memory space of the subsequent S-record data block:

Value	DSP Memory Space
1	X
2	Y
3	L
4	P
5	E

With the **-S** option whenever a memory change occurs a new S0 header record is produced. In this case if a data block were to be located in X memory, the address field would contain '30303031' (note that such use of the S0 address field is a DSP-specific extension to the standard S0 record format). The code/data field may contain any descriptive information identifying the following block of S-records. This is followed by the normal two-character checksum.

5.6.2.2 S1, S2, S3 Records

Each data record begins with the start characters S1, S2, or S3 followed by a byte count. These record types vary only by the length of their respective address fields. An S1 record has a 2-byte address field represented by 4 hexadecimal characters. An S2 record has a 3-byte address field represented by 6 hexadecimal characters. An S3 record has a 4-byte address field represented by 8 hexadecimal characters. Data bytes follow the address field and are represented by hexadecimal character pairs. A two-character checksum terminates the data record. The SREC program guarantees that the number of bytes in an S1, S2, or S3 data record is an integral multiple of the word size of the target DSP.

5.6.2.3 S7, S8, S9 Records

These are end-of-file records and may appear only once in the S-record file. Each trailer record begins with the start characters S7, S8, or S9 followed by a byte count. As with the data records these record types vary only by the length of their respective address fields. An S7 record has a 4-byte address field represented by 8 hexadecimal characters. An S8 record has a 3-byte address field represented by 6 hexadecimal characters. An S9 record has a 2-byte address field represented by 4 hexadecimal characters. The address field in the trailer record is used by the SREC program to store the end address given in the object file optional header record. A two-character checksum immediately follows the address field.

Chapter 6
MOTOROLA DSP COFF FILE DUMP UTILITY (COFDMP)

6.1 INTRODUCTION

The Motorola DSP COFF file dump program COFDMP is a stand-alone utility that reads an absolute or relocatable Common Object File Format (COFF) file and produces a formatted display of the object file contents. The entire file or only selected portions may be processed depending on command line options. The program also can generate either codes or symbolic references to entities such as symbol type and storage class.

6.2 INSTALLING COFDMP

The COFDMP program is distributed on various media and in different formats depending on the host operating system environment. See Appendix G in the **Motorola DSP Assembler Reference Manual**, HOST-DEPENDENT INFORMATION, for details on installing and operating COFDMP on your particular machine.

6.3 RUNNING COFDMP

The general format of the command line to invoke COFDMP is:

COFDMP [options] <files>

where:

[options]

Any of the following command line options. The options must precede the file name. Option letters may be specified in either upper or lower case. If no option is supplied the entire input file is dumped.

6.4 COFDMP OPTIONS

-C

Dump the string table of the specified file. This information may not be present if the object file has been stripped.

Example: **COFDMP -C** fft16.cld

In this example, the symbol table is listed from the absolute object file FFT16.CLD.

-D<dmpfil>

This option specifies that a dump file is to be created for cofdmp output. <dmpfil> can be any legal operating system filename, including an optional pathname.

If the **-D** option is not specified, then the program will route dump output to the standard output (usually the console or terminal screen) by default. The **-D** option should be specified only once. **If the file named in the -D option already exists, it will be overwritten.**

Example: **COFDMP -D** filter.dmp filter.cld

In this example, the absolute load file FILTER.CLD will be dumped to the output file FILTER.DMP.

-F

Dump the file header of the specified file.

Example: **COFDMP -F** fft16.cln

In this example, the file header is listed from the relocatable object file FFT16.CLN.

-H

Dump the section headers of the specified file.

Example: **COFDMP -H** fft16.cld

In this example, the section headers are listed from the absolute object file FFT16.CLD.

-L

Dump the source file line number information from the specified file. This information may not be present if the object file has been stripped.

Example: **COFDMP -L** fft16.cln

In this example, the source file line number information is listed from the relocatable object file FFT16.CLN.

-O

Dump the optional header of the specified file.

Example: **COFDMP -O** fft16.cln

In this example, the optional header is listed from the relocatable object file FFT16.CLN.

-Q

On some hosts COFDMP displays a banner on the console when invoked. This option inhibits the banner display. It has no effect on hosts where the signon banner is not displayed by default.

Example: **COFDMP -Q** myprog.cld

Dump the file MYPROG.CLD but do not display the signon banner on the console.

-R

Dump section relocation entries from the specified file. Only relocatable object files will contain this information.

Example: **COFDMP -R** fft16.cln

In this example, the section relocation information is listed from the relocatable object file FFT16.CLN.

-S

Dump the section raw data from the specified file.

Example: **COFDMP -S** fft16.cld

In this example, the section raw data is listed from the absolute object file FFT16.CLD.

-T

Dump the symbol table from the specified file. This information may not be present if the object file has been stripped.

Example: **COFDMP -T** fft16.cln

In this example, the symbol table is listed from the relocatable object file FFT16.CLN.

-V

Dump the specified file symbolically, using names for bit flags, symbol types, and storage classes.

Example: **COFDMP -V** fft16.cld

In this example, the entire contents of the absolute object file FFT16.CLD is dumped symbolically.

<files>

A list of operating system compatible filenames. If no pathname is specified for a file, the program will look for that file in the current directory. An explicit filename must be provided; there is no default extension for the input file.

6.5 COFDMP PROCESSING

The COFDMP program reads the input file and writes a formatted dump of the file contents to the standard output (unless the **-D** command line option is given). I/O redirection may be used to send the output to a file if the host operating system supports it.

The program currently will not dump individual modules from library files; they must be extracted and then dumped. Also note that if the file has been stripped some information may no longer be available in the file, such as relocation information, line number entries, and symbol and string table data.

The input file must be a Motorola DSP COFF object file, either absolute or relocatable. See Appendix E in the **Motorola DSP Assembler Reference Manual**, MOTOROLA DSP OBJECT FILE FORMAT, for more information on Motorola DSP COFF object files.

Appendix A
LINKER MESSAGES

A.1 INTRODUCTION

Linker messages are grouped into four categories:

Command Line Errors

These errors indicate invalid command line options, missing filenames, file open errors, or other invocation errors. Command line errors generally cause the Linker to stop processing.

Warnings

Warnings notify the programmer of suspect constructs but do not otherwise affect the object file output.

Errors

These errors indicate problems with object file format, size of address fields, and syntax. In these cases the resulting object code is generally not valid.

Fatal

Fatal errors signify serious problems encountered during the link process such as lack of memory, file not found, or other internal errors. The Linker halts immediately.

The Linker also will provide information on the file name, module ID, and section location of the error, if it can be ascertained. Messages are always routed to standard output.

A.2 COMMAND LINE ERRORS

Align not valid with incremental link - ignored

Both the **-I** and **-A** options were given on the command line. The **-A** option auto-aligns circular buffers in the output file, but an incrementally linked file (**-I**) must retain original buffer placement for future linking. If both are specified, the Linker does not auto-align.

Cannot open command file
Cannot open library file
Cannot open map file
Cannot open memory control file
Cannot open object file

The file associated with a **-F**, **-L**, **-M**, **-R**, or **-B** command line option was not found or could not be opened.

Default object file not allowed in incremental link

When performing an incremental link using the **-I** option the **-B** option must be used in order to name the output object file. The default naming convention cannot be used because it might overwrite one of the input files.

Duplicate map file specified - ignored
Duplicate memory control file specified - ignored
Duplicate object file specified - ignored

More than one **-M**, **-R**, or **-B** option was encountered on the command line.

Illegal command line option

The option specified on the command line was not recognized by the Linker.

Illegal command line -E option

Neither of the qualifiers **-A** or **-W** were provided with the **-E** option.

Illegal command line -X option argument

The argument given with the **-X** command line option was not recognized by the Linker.

Invalid syntax for command line -E option

There must be whitespace between the **-E** option and its filename argument.

Missing argument for command line -E option

The **-E** command line option must have a filename argument.

Missing command line option argument

The expected arguments following a command line specifier were missing.

Missing object filename

There must be at least one object filename specified on the command line.

No modules linked - empty object file

An object file header record was never found in the input.

Object file name same as executable file name
Object file name same as map file name

One of the object files appeared to the Linker to have the same name as the specified executable or map file. The Linker aborts rather than potentially writing over an input object file.

Options for both debug and strip specified - strip ignored

Both the **-G** and **-Z** options were given on the command line. The **-G** option takes precedence.

Seek failure

An attempt to seek randomly to verify a library file has failed.

Strip not valid with incremental link - ignored

Both the **-I** and **-Z** options were given on the command line. The **-I** option takes precedence.

A.3 WARNINGS

Actual length of section greater than specified size

The length of a section is greater than the absolute size given in a memory control file **SECSIZE** directive.

Duplicate global symbol

A global symbol in one object file was also defined by the same name in a different module.

Duplicate XDEF symbol

An external symbol in one object file was also defined by the same name in a different module.

EMI 8-bit memory value truncated
EMI 12-bit memory value truncated
EMI 16-bit memory value truncated
EMI 20-bit memory value truncated

The value in a data directive was too large for the current EMI memory configuration.

Empty bit mask field

The first operand of a **BFxxx**-type instruction was zero.

Expression value outside fractional domain

The expected fractional value was not within the range -1.0 <= *m* < 1.

Incompatible debug format

The input object file debug information is not compatible with the current version of the Linker.

Memory model mismatch

An object file produced by the C compiler is incompatible with other files in the input stream. This can result when C source files compiled using X memory for data storage and others compiled using Y memory for data storage are linked. The Linker cannot reconcile the memory differences.

No modules linked - empty object file

An object file header record was never found in the input.

Object file major version number greater than Linker major version number
Object file minor version number greater than Linker minor version number

The input file version number generated by the assembler does not match the current version number of the Linker.

Options for both debug and strip specified - strip ignored

Both the **-G** and **-Z** options were given on the command line. The **-G** option takes precedence.

Overlay buffer not aligned

A circular buffer inside a relocatable overlay will be misaligned when the overlay is transferred at runtime. The Linker makes no attempt to realign buffers located inside relative overlay blocks.

Remapping region
Remapping section

A **REGION** or **SECTION** directive set an existing memory space to a different mapping attribute.

Section already set as absolute

A section listed as ordered in a memory control file **SECTION** record was found to be already located absolutely.

String truncated in expression evaluation

Only the first four characters of a string constant are used during expression evaluation.

Strip not valid with incremental link - ignored

The **-I** and **-Z** command line options are mutually exclusive. The **-I** option takes precedence.

A.4 ERRORS

Arithmetic exception

An internal floating point exception occurred while evaluating an expression. The result of the evaluation is probably not valid.

Autoaligned buffer not allowed in overlay

A circular buffer inside a relocatable overlay cannot be auto-aligned because the overlay block will be moved, voiding any benefit of optimal buffer placement.

Binary constant expected

A character other than ASCII '0' or '1' either followed the binary constant delimiter (%) or appeared in an expression where a binary value was expected by default.

Bit mask cannot span more than eight bits

If the first operand of a **BFxxx**-type instruction was shifted one bit to the right until the low-order bit was a 1, the resulting value must not exceed $FF hexadecimal.

Buffer block too large

The runtime location counter overflowed while the Linker was attempting to allocate storage for a data buffer. The Linker automatically advances the program counter to the next valid base address given the size of the modulo or reverse carry buffer.

Buffer out of order

The buffer sequence numbers in the input stream are out of phase.

Cannot nest regions

A **REGION** directive in the memory control file may not appear between another **REGION/ENDR** pair.

Cannot open include file

The file specified in a memory control file **INCLUDE** directive was not found or could not be opened.

Decimal constant expected

A character other than ASCII '0' through '9' either followed the decimal constant delimiter (`) or appeared in an expression where a decimal value was expected by default.

Divide by zero

The expression evaluator detected a divide by zero.

Duplicate global symbol

Two identically named global symbols have been found.

Duplicate local symbol

Two identically named symbols have been found which are local to the same section.

Duplicate region assignment for section

A memory control file **SECTION** directive with the same name and memory attributes was assigned to more than one region.

Duplicate special symbol

The Linker reserves a few symbol names to deal with certain floating point entities such as Infinity and Not-a-Number. These symbols are Inf, Nan, Tiny, and Huge.

ENDR without corresponding REGION directive

An **ENDR** directive in the memory control file did not have a matching **REGION** directive.

Expression cannot have a negative value

The **MAP PAGE** directive does not allow negative expression arguments.

Expression contains forward references

The expression representing a location counter number contains a term which the expression evaluation logic cannot resolved (e.g. an undefined symbol).

Expression involves incompatible memory spaces

The memory space attribute is regarded by the Linker as a type, in the same sense that high level languages use type for variables. Symbols may have memory space attributes of **X**, **Y**, **L**, **P**(rogram), **E**(MI), or **N**(one); only **N** is fully compatible with all other attributes. In this case, two operands were evaluated with different memory space attributes, neither of which was **N**.

Expression result must be absolute

Certain directives and some Linker usage require absolute values as arguments or operands.

Expression result must be integer

The expression refers to an address; therefore the result must be an integer within the address range of the target DSP.

External reference not allowed in expression

The expression contained an undefined symbol which the expression evaluation logic cannot resolve.

Extra characters beyond expression

The expression evaluator found extra characters after the end of a valid expression. Unbalanced parentheses can cause this error.

Extra characters in function argument or missing ')' for function

Mismatched parentheses or wrong number of parameters in a function invocation.

Floating point constant expected

A character other than ASCII '0' through '9', 'e' or 'E', or '.' appeared in an expression where a floating point value was expected by default.

Floating point not allowed in relative expression

Relative expressions are generally used for address computation, therefore a floating point value would not be appropriate.

Hex constant expected

A character other than ASCII '0' through '9', 'a' through 'f', or 'A' through 'F' either followed the hexadecimal constant delimiter (**$**) or appeared in an expression where a hexadecimal value was expected by default.

Illegal memory counter specified

The memory counter specifier must be **H** for the high counter, **L** for the low counter, or absent for the default counter.

Illegal memory map character

The memory map indicator must be **I** for internal memory, **E** for external memory, **R** for ROM, **A** for port A, **B** for port B, or absent for no explicit mapping.

Illegal operator for floating point element

Bitwise operators are invalid for floating point values.

Illegal option

A bad argument was provided with the **-X** command line option.

Invalid address expression

The memory space attributes of the expression operands are incompatible.

Invalid address relocation field

Either a new record began or end-of-file was reached when the Linker was reading the address specification in a memory file **BASE** or **SECTION** record.

Invalid EMI memory designation

The EMI memory type does not fall within an appropriate range of values.

Invalid function name

The Linker could not match an internal function name.

Invalid include file name

The filename associated with a memory control file **INCLUDE** directive is missing or malformed.

Invalid MAP option
Invalid MAP option field
Invalid MAP page field
Invalid MAP record field

One of the options or fields in a memory control file **MAP** record was not recognized by the Linker.

Invalid memory space field

The memory space specifier for a **SIZSYM** directive was not found.

Invalid module name field

The module name field associated with a memory control file **IDENT** directive is missing or malformed.

Invalid overlay base address

An overlay base address was given that specified an address within another overlay segment.

Invalid page length specified

The minimum page length allowed by the **MAP PAGE** directive is 10 lines per page. The maximum is 255.

Invalid page width specified

The minimum page width allowed by the **MAP PAGE** directive is 1 column per line. The maximum is 255.

Invalid region name field

The region name associated with a memory control file **REGION** directive is missing or malformed.

Invalid relative expression

The terms of a relative expression may only participate in addition and subtraction operations and must have opposing signs.

Invalid relocation type field

An invalid record type was encountered in the memory control file.

Invalid reserve range syntax

The syntax for the memory control file **RESERVE** is such that the range is given as two values from the same memory space and mapping, separated by two periods in succession (**..**).

Invalid revision number field

The revision number field in a memory control file **IDENT** record is not valid. The value must be a decimal integer number.

Invalid section name field

The section name associated with a memory control file **SECTION** directive is missing or malformed.

Invalid shift amount

A shift expression must evaluate to within the range 0 <= *n* <= 32.

Invalid source line number

The source line number in a relocation expression must be an integer.

Invalid start address expression

The end address expression in the optional header record is malformed.

Invalid start address field

The start address field in a memory control file **START** record is not valid. The contents must be a positive numeric value.

Invalid symbol

Symbols are limited to 512 characters. The first character must be alphabetic or the underscore character (A-Z, a-z, _). The remaining characters must be alphanumeric, including the underscore character (A-Z, a-z, 0-9, _).

Invalid symbol memory mapping

The memory type for a symbol in the symbol table does match a valid Linker memory configuration.

Invalid symbol name field

Either a new record began or end-of-file was reached when the Linker was reading the name specification in a memory file **SYMBOL** record.

Invalid symbol value field

Either a new record began or end-of-file was reached when the Linker was reading the value specification in a memory file **SYMBOL** record.

Invalid version number field

The version number field in a memory control file **IDENT** record is not valid. The value must be a decimal integer number.

Left margin exceeds page width

The blank left margin value in the **MAP PAGE** directive exceeds the default or specified page width parameter.

Missing '(' for function

Parentheses are not balanced in an internal function call.

Missing ')' in expression

Parentheses are not balanced in an expression.

Missing ']' in expression

Square brackets are not balanced in an expression.

Missing '}' in expression

Curly braces are not balanced in an expression.

Missing expression

An expression was expected by the expression evaluator.

Missing filename

The filename associated with a memory control file **INCLUDE** directive is missing.

Missing option

The argument associated with the **-X** command line option is missing.

Missing quote in string

A single or double quote character was expected by the string parsing routines.

Missing string after concatenation operator

The string concatenation operator (**++**) must be followed by another quoted string.

No previous function declaration

An end-of-function symbol record was encountered without a corresponding function type symbol record.

Operation not allowed with address term

Only addition and subtraction are allowed in expressions with address terms.

Overlay address involves incompatible memory spaces

The memory space attribute is regarded by the Linker as a type, in the same sense that high level languages use type for variables. Symbols may have memory space attributes of **X**, **Y**, **L**, **P**(rogram), **E**(MI), or **N**(one); only **N** is fully compatible with all other attributes. In this case, the runtime overlay address was found to be incompatible with the memory space used as the overlay origin.

Overlay out of order

The overlay sequence numbers in the input stream are out of phase.

Page length too small for specified top and bottom margins

The sum of the top and bottom margins specified in the **MAP PAGE** directive is greater than the page length - 10.

Page length too small to allow default bottom margin

The bottom margin exceeds the page length specified in the **MAP PAGE** directive.

Region high address lower than base address

A memory control file **BASE** directive had a greater value than a corresponding **MEMORY** directive for a given region.

Region size/address mismatch

The size field in a memory control file **REGION** directive does not correlate with the computed size derived from **BASE** and **MEMORY** directives for the same region.

Region without associated ENDR directive

A **REGION** directive in the memory control file did not have a matching **ENDR** directive.

Relative expression must be integer

A relative expression must evaluate to an integer value.

Relative terms from different sections not allowed

Two relative terms from different sections may not participate in an arithmetic operation since the result might not be meaningful. This error may be disabled with the Linker **XC** option.

Section not found

The section named in a **SECTION**, **SBALIGN**, or **SECSIZE** directive could not be found by the Linker.

Section padding percentage too small

The percentage of padding in a memory control file **SECSIZE** record must be greater than 100.0.

Specified address greater than maximum memory address

The directive address is greater than the argument given in a previously encountered **MEMORY** directive for this region.

Specified size greater than maximum memory address

The size given in a memory control file **SECSIZE** directive is greater than the argument given in a previously encountered **MEMORY** directive for this region.

Symbol already defined

A symbol assumed to be unresolved and named in a memory file **SYMBOL** record has already been defined.

Symbol name too long

Symbols are limited to 512 characters. The first character must be alphabetic or the underscore character (A-Z, a-z, _). The remaining characters must be alphanumeric, including the underscore character (A-Z, a-z, 0-9, _).

Symbol tag mismatch

A matching tag reference could not be found for a tagged symbol table entry.

Syntax error - expected ':'
Syntax error - expected '):'

The Linker was expecting the end of either a memory space designator or a location counter expression while parsing a memory attribute string.

Syntax error - expected comma

The Linker was expecting a comma while parsing the arguments of a function.

Syntax error - expected quote

The Linker was expecting the start of a quoted string.

Syntax error in expression

The syntax in a memory file **BASE** or **SECTION** record is not correct (possibly missing a colon before the address specification).

Unresolved overlay base address

The symbol used as the runtime overlay address was never resolved during the fix-up phase.

A.5 FATAL ERRORS

Arithmetic exception

An internal floating point exception occurred while evaluating an expression.

Cannot determine file size

The size of the input link module could not be determined.

Cannot find GLOBAL section

The memory control file processing logic could not locate a GLOBAL section record.

Cannot find section record

The Linker was unable to locate a previously accessed section record. This is a serious internal error that should be reported to Motorola.

Cannot open library file
Cannot open object file

The Linker attempted to open a library or object file for reprocessing on the second pass and the open failed.

Cannot read file header from library module
Cannot read file header from object module

An I/O error occurred which prevented the Linker from reading the file headers in a library or object file.

Cannot read line number entries from object module
Cannot read module string table size
Cannot read object module section headers
Cannot read object module string table
Cannot read object module symbol entries
Cannot read raw data from object module
Cannot read relocation entries from object module

An I/O error occurred which prevented the Linker from reading entities in the input object file.

Cannot read optional header from library module
Cannot read optional header from object module

An I/O error occurred which prevented the Linker from reading the optional headers in a library or object file.

Cannot seek to library module symbol table

An I/O error occurred which prevented the Linker from positioning correctly when attempting to read a library module symbol table.

Cannot seek to object module line number entries
Cannot seek to object module raw data
Cannot seek to object module relocation entries
Cannot seek to object module section headers
Cannot seek to object module symbol table

An I/O error occurred which prevented the Linker from positioning correctly in the input object file.

Cannot seek to start of line number entries
Cannot seek to start of object data
Cannot seek to start of object file
Cannot seek to start of object module
Cannot seek to start of section headers
Cannot seek to start of string table
Cannot seek to start of symbol table

An I/O error occurred which prevented the Linker from positioning correctly in the output object file.

Cannot set current section

The current module section map has been corrupted. This is a serious internal error that should be reported to Motorola.

Cannot set section counter

The current module counter map has been corrupted. This is a serious internal error that should be reported to Motorola.

Cannot write left margin to map file
Cannot write new line to map file
Cannot write new page to map file
Cannot write page header to map file
Cannot write string to map file

An I/O error occurred which prevented the Linker from writing data to the output map file.

Cannot write .text/.data headers to object file
Cannot write file header to object file
Cannot write line number entries to object file
Cannot write optional header to object file
Cannot write padding to object file

Cannot write raw data to object module
Cannot write relocation entries to object module
Cannot write section headers to object module
Cannot write string table to object file
Cannot write symbols to object file

An I/O error occurred which prevented the Linker from writing data to the output object file.

Compare select failure

The comparison indicator passed to the evaluator selection logic was not valid. This is a serious internal error that should be reported to Motorola.

Current relocation map not available

A valid relocation map could not be accessed. This is a serious internal error that should be reported to Motorola.

Current relocation section not available

A valid relocation section could not be accessed. This is a serious internal error that should be reported to Motorola.

Current section not available

A valid current section could not be accessed. This is a serious internal error that should be reported to Motorola.

Duplicate section entry

An attempt was made to place two sections with the same name and memory attributes into a single region.

Expression operator failure

Expression operator lookup has failed. This is a serious internal error that should be reported to Motorola.

Expression stack underflow

The internal expression evaluation list is out of sequence. This is a serious internal error that should be reported to Motorola.

Fatal segmentation or protection fault; contact Motorola

A program error has caused the Linker to access an invalid host system address. This generally indicates a bug in the Linker software.

File contains no relocation information

An absolute object file was specified as input to the Linker.

Invalid data block type

Internal section type information has been corrupted.

Invalid global section

A section with a sequence number of zero did not have the proper global section name.

Invalid library module header

An I/O error occurred when attempting to read a library module header.

Invalid library module header format

The module header for a file contained in a library has been corrupted.

Invalid object file for target processor

Incompatible target processor object files were specified as input to the Linker.

Invalid operand bit size

The bit size operand in an object file expression was not recognized.

Invalid section number

A new section encountered by the Linker does not have a unique section number.

Invalid section number data

The section number in a section symbol record is out of range.

Map option select failure

The value returned from the mapping selection logic was not valid. This is a serious internal error that should be reported to Motorola.

No current counter map

A valid counter map was not available. This is a serious internal error that should be reported to Motorola.

Offset failure

An attempt to save the object file offset has failed.

Option select error

The value returned from the option selection logic was not valid. This is a serious internal error that should be reported to Motorola.

Out of memory - link aborted

There is not enough internal memory to perform dynamic allocation. Since the Linker stores all working information in memory, including symbol and section information, there is the possibility that memory will be exhausted if many symbols or sections are defined in a single Linker run.

Relocation type select failure

A value returned from the memory control file function select logic is bad. This is a serious internal error that should be reported to Motorola.

Section map lookup failure

A valid section map could not be accessed. This is a serious internal error that should be reported to Motorola.

Section nesting error

The section nesting count is out of phase.

Seek failure

An attempt to seek randomly in the object file has failed.

Symbol map lookup failure

A valid symbol map could not be accessed. This is a serious internal error that should be reported to Motorola.

Appendix B
LIBRARIAN MESSAGES

B.1 INTRODUCTION

Librarian messages are grouped into three categories:

Command Line Errors

These errors indicate invalid command line options, missing filenames, file open errors, or other invocation errors. Command line errors generally cause the librarian to stop processing.

Warnings

Warnings indicate that a file cannot be open, or that a module already exists or does not exist in the library. The librarian continues processing.

Fatal Errors

Fatal errors signify serious problems encountered during library processing such as lack of memory, file not found, or other internal errors. The librarian halts immediately.

B.2 COMMAND LINE ERRORS

argument missing

A necessary argument, such as a module name, was missing from the librarian command line.

unknown option

The given command line option is not recognized. The librarian continues as if the **-U** option had been given.

B.3 WARNINGS

<module> already in library

In an add operation a module with the same name as the one specified already exists in the library.

<module> not in library

The named module was not found in the specified library. This error can occur for example during a replace operation.

ambiguous command

The interactive command issued at the librarian prompt was not unique to the set of characters entered.

cannot open module file

The named module file could not be open. Either the file does not exist or there was an I/O error.

command requires library name

All interactive commands require that the library name follow the command name on the input line.

duplicate module name

The same module name was entered twice on the command line.

invalid command

The librarian did not recognize an interactive command.

B.4 FATAL ERRORS

add requires explicit module names

At least one module name must be given for an add operation.

cannot allocate argument vector
cannot allocate copy buffer
cannot allocate input buffer
cannot allocate module structure
cannot allocate module vector
cannot allocate output buffer

The librarian did not have enough memory to allocate internal data structures.

cannot create temporary file name

The librarian was unable to create a temporary file name for the library scratch file.

cannot open command file

The named command line option file does not exist, or there was an I/O error.

cannot open library file

The named library file does not exist, or there was an I/O error.

cannot open temporary file

An I/O error prevented the librarian from opening the temporary library scratch file.

cannot read module header

The librarian could not read the module header in the specified library. Either an I/O error occurred, or the library file is empty.

cannot rename <file>

An error occurred while attempting to rename a library file.

cannot save command file arguments

The librarian could not obtain enough memory to store the module names given in the command line option file.

cannot stat module

The librarian could not obtain date and time information for the named module.

cannot write end of module marker

An I/O error occurred preventing the librarian from writing the end of module marker to the library file.

cannot write header to library file

An I/O error occurred preventing the librarian from writing the module header to the library file.

delete requires explicit module names

At least one module name must be given for a delete operation.

error reading file

An I/O error occurred while reading a file.

error writing file

An I/O error occurred while writing a file.

fatal errors - <library> not altered

This is an informative message indicating that the named library was not changed because of previous fatal errors.

file I/O error

An I/O error occurred while either reading or writing a file.

improper module header format

A module header in the library file has been corrupted, or the specified file is not a library file.

library file already exists

In a create operation the named library file already exists.

missing command filename

The required argument on a **-F** command line option was missing.

out of memory - librarian aborted

There is not enough internal memory to perform dynamic allocation. Since the librarian stores all working information in memory, there is the possibility that memory will be exhausted if many modules are processed in a single library operation.

Appendix C
LINKER MAP FILE FORMAT

C.1 INTRODUCTION

The Linker optionally produces a memory map listing file when the command line **-M** is specified. See Chapter 1, Running the Linker for more information on command line and map listing options. If the **-M** command line option is given, the map listing goes to the file named as the option argument; if no argument is specified, the map listing file takes the name of the first object file on the command line and changes the extension to .MAP (see Chapter 1).

C.2 MAP FILE COMMENTARY

Figure C-1 is a Linker-generated map listing of a sample application. The listing illustrates a selection of the format and reporting features provided by the Linker. The following section highlights some of those features.

At the top of every map listing page is a banner which identifies the Linker and lists its version number, the date and time of linking, the current input file name, and the page number. The map file page length, width, and margins can be controlled by the memory control file **MAP PAGE** record (see Chapter 3, MAP PAGE Map File Format Control).

The first titled grouping in the report is a list of sections sorted by starting address. This list of sections is subdivided by DSP memory and ordered by counter such that all X default (counter 0) memory references are grouped together, followed by X low (counter 1) memory, and so forth. Each line gives the starting address, ending address, and length of every uniquely-named section in the Linker input stream if that section contained code or data for the current memory space. The length reflects the total of all section fragments assimilated from separate input files. As a result there is only one line for each section even if the section appears in different files. If sections are located such that they overlap in memory the Linker will flag the overlap in the map file to the right of the section name. The link map also shows any unused memory areas between allocated blocks. These lines may be disabled using the **MAP OPT NOUNUSED** directive (see Chapter 3, MAP OPT Map File Contents Control).

A section name may be repeated for a given memory space if that section contains buffers, overlays, or absolute blocks. In this case the start, end, and size of the block is reported and the type of block is placed to the right of the section name. On page 1 of the

example listing section SECT1 contains a modulo buffer of length 32 starting at address 20 hexadecimal in X memory. On the same page the section SECT2 has an overlay segment of length 7 that is loaded at address 12 hexadecimal in P memory. The listing of sections by address can be turned off with the memory control file **MAP OPT NOSECADDR** directive (see Chapter 3, MAP OPT Map File Contents Control).

The next titled grouping on the map report (page 2) is a list of sections sorted by name. The name of the section is given along with the start, end, and length of blocks in each DSP memory space. As in the section by address listing special blocks such as buffers or overlays are shown on a line by themselves. The section by name report can be disabled by using the memory control file **MAP OPT NOSECNAME** directive (see Chapter 3, MAP OPT Map File Contents Control).

After the section-oriented reports there appears on page 3 of the map file a symbol listing ordered by name. Each line starts with the symbol name (truncated to 16 characters), followed by the symbol type (integer or floating point), the memory space if any and value, the name of the section in which the symbol is defined, and the symbol attributes. A symbol can be absolute or relative (REL), local, XDEFed (EXTERN), or global, and possibly associated with a buffer or overlay. This portion of the map listing may be omitted through the memory control file **MAP OPT NOSYMNAME** directive (see Chapter 3, MAP OPT Map File Contents Control).

The last page of the listing shows a symbol listing sorted by value. The listing by value can be turned off with the memory control file **MAP OPT NOSYMVAL** directive (see Chapter 3, MAP OPT Map File Contents Control).

The final report group lists the unresolved externals found during the link phase. This consists of all the symbol references for which there was no corresponding definition found in the link input. The Linker indicates the symbol name and the module in which the reference was made.

Motorola DSP Linker Version 5.0 92/06/25 15:44:20 sample.map Page 1

Section Link Map by Address

X Memory (0 - default)

Start	End	Length	Section	
0000	0040	65	sect1	
0020	003F	32	sect1	Mod
0041	005F	31	UNUSED	
0060	0082	35	sect1a	
0060	007F	32	sect1a	Mod
0083	0089	7	sect2	
0089	FFFF	65399	UNUSED	

P Memory (0 - default)

Start	End	Length	Section	
0000	0006	7	sect1	Abs
0007	000D	7	sect1	
000E	0011	4	sect1a	
0012	0018	7	sect2	Ovl
0012	0017	6	sect2	
0019	02FF	743	UNUSED	
0300	0302	3	sect1a	Abs
0303	FFFF	64765	UNUSED	

Figure C-1 Linker Map Format

Motorola DSP Linker Version 5.0 92/06/25 15:44:20 sample.map Page 2

Section Link Map by Name

Section		Memory	Start	End	Length
GLOBAL		None			
sect1		X default	0000	0040	65
	Mod	X default	0020	003F	32
	Abs	P default	0000	0006	7
		P default	0007	000D	7
sect1a		X default	0060	0082	35
	Mod	X default	0060	007F	32
		Y default	0000	0020	33
		P default	000E	0011	4
	Abs	P default	0300	0302	3
sect2		X default	0083	0089	7
		Y default	0021	0031	17
	Ovl	P default	0012	0018	7
		P default	0012	0017	6

Figure C-1 Linker Map Format (continued)

Motorola DSP Linker Version 5.0 92/06/25 15:44:20 sample.map Page 3

Symbol Listing by Name

Name	Type	Value	Section	Attributes
next	int	P:000007	sect1	REL GLOBAL
ovlab1	int	P:000012	sect2	REL OVERLAY
ovlab2	int	P:000016	sect2	REL EXTERN OVERLAY
sect1_ds	int	X:000000	sect1	REL GLOBAL
sect1a123	int	X:000080	sect1a	REL
sect1a_ds	int	Y:000000	sect1a	REL
sect1abuf	int	X:000060	sect1a	REL BUFFER
sect1buf	int	X:000020	sect1	REL GLOBAL BUFFER
sect2_ds	int	Y:000021	sect2	REL
sect2_ov	int	P:000012	sect2	REL
sym1	int	X:000040	sect1	REL GLOBAL
sym2	int	Y:000020	sect1a	REL EXTERN
sym3	int	Y:000031	sect2	REL

Figure C-1 Linker Map Format (continued)

Motorola DSP Linker Version 5.0 92/06/25 15:44:20 sample.map Page 4

Symbol Listing by Value

Value	Name	Value	Name	Value	Name
000000	sect1_ds	000000	sect1a_ds	000007	next
000012	ovlab1	000012	sect2_ov	000016	ovlab2
000020	sect1buf	000020	sym2	000021	sect2_ds
000031	sym3	000040	sym1	000060	sect1abuf
000080	sect1a123				

Unresolved Externals:

ovlab2 (sample.lnk)
undef1 (sample.lnk)
undef2 (sample.lnk)

Figure C-1 Linker Map Format (continued)

INDEX