HI-TECH C® for PIC10/12/16
User's Guide

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

ISBN: 978-1-60932-739-2

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

6.6 HEXMATE

The HEXMATE utility is a program designed to manipulate Intel HEX files. HEXMATE is a post-link stage utility which is automatically invoked by the compiler driver, and that provides the facility to:

- Calculate and store variable-length checksum values
- Fill unused memory locations with known data sequences
- Merge multiple Intel HEX files into one output file
- Convert INHX32 files to other INHX formats (e.g. INHX8M)
- Detect specific or partial opcode sequences within a HEX file
- Find/replace specific or partial opcode sequences
- Provide a map of addresses used in a HEX file
- Change or fix the length of data records in a HEX file.
- Validate checksums within Intel HEX files.

Typical applications for HEXMATE might include:

- Merging a bootloader or debug module into a main application at build time
- Calculating a checksum over a range of program memory and storing its value in program memory or EEPROM
- Filling unused memory locations with an instruction to send the PC to a known location if it gets lost.
- Storage of a serial number at a fixed address.
- Storage of a string (e.g. time stamp) at a fixed address.
- Store initial values at a particular memory address (e.g. initialize EEPROM)
- Detecting usage of a buggy/restricted instruction
- Adjusting HEX file to meet requirements of particular bootloaders

6.6.1 HEXMATE Command Line Options

HEXMATE is automatically called by the command line driver, PICC. This is primarily to merge in HEX files with the output generated by the source files, however there are some PICC options which directly map to HEXMATE options, and so other functionality can be requested without having to run HEXMATE explicitly on the command line. For other functionality, the following details the options available when running this application.

If HEXMATE is to be run directly, its usage is:

HEXMATE [specs,]*file1.HEX* [[specs,]*file2.HEX* ... [specs,]*fileN.HEX*] [*options*]

Where *file1.HEX* through to *fileN.HEX* form a list of input Intel HEX files to merge using HEXMATE. If only one HEX file is specified, then no merging takes place, but other functionality is specified by additional options. Table 6-8 lists the command line options that HEXMATE accepts.

TABLE 6-8: HEXMATE COMMAND-LINE OPTIONS

Option	Effect
-ADDRESSING	Set address fields in all HEXMATE options to use word addressing or other
-BREAK	Break continuous data so that a new record begins at a set address
-CK	Calculate and store a checksum value
-FILL	Program unused locations with a known value
-FIND	Search and notify if a particular code sequence is detected

TABLE 6-8: HEXMATE COMMAND-LINE OPTIONS (CONTINUED)

Option	Effect
-FIND...,DELETE	Remove the code sequence if it is detected (use with caution)
-FIND...,REPLACE	Replace the code sequence with a new code sequence
-FORMAT	Specify maximum data record length or select INHX variant
-HELP	Show all options or display help message for specific option
-LOGFILE	Save HEXMATE analysis of output and various results to a file
-O*file*	Specify the name of the output file
-SERIAL	Store a serial number or code sequence at a fixed address
-SIZE	Report the number of bytes of data contained in the resultant HEX image.
-STRING	Store an ASCII string at a fixed address
-STRPACK	Store an ASCII string at a fixed address using string packing
-W	Adjust warning sensitivity
+	Prefix to any option to overwrite other data in its address range if necessary

The input parameters to HEXMATE are now discussed in greater detail. Note that any integral values supplied to the HEXMATE options should be entered as hexadecimal values without leading 0x or trailing h characters. Note also that any address fields specified in these options are to be entered as byte addresses, unless specified otherwise in the -ADDRESSING option.

6.6.1.1 SPECIFICATIONS,FILENAME.HEX

Intel HEX files that can be processed by HEXMATE should be in either INHX32 or INHX8M format. Additional specifications can be applied to each HEX file to put restrictions or conditions on how this file should be processed.

If any specifications are used they must precede the filename. The list of specifications will then be separated from the filename by a *comma*.

A *range restriction* can be applied with the specification r*Start-End*. A range restriction will cause only the address data falling within this range to be used. For example:

```
r100-1FF,myfile.hex
```

will use myfile.hex as input, but only process data which is addressed within the range 100h-1FFh (inclusive) from that file.

An address shift can be applied with the specification *sOffset*. If an address shift is used, data read from this HEX file will be shifted (by the offset specified) to a new address when generating the output. The offset can be either positive or negative. For example:

```
r100-1FFs2000,myfile.HEX
```

will shift the block of data from 100h-1FFh to the new address range 2100h-21FFh.

Be careful when shifting sections of executable code. Program code should only be shifted if it is position independent.

6.6.1.2 + PREFIX

When the + operator precedes an argument or input file, the data obtained from that source will be forced into the output file and will overwrite another other data existing at that address range. For example:

```
+input.HEX +-STRING@1000="My string"
```

Ordinarily, HEXMATE will issue an error if two sources try to store differing data at the same location. Using the + operator informs HEXMATE that if more than one data source tries to store data to the same address, the one specified with a + prefix will take priority.

6.6.1.3 -ADDRESSING

By default, all address arguments in HEXMATE options expect that values will be entered as byte addresses. In some device architectures the native addressing format may be something other than byte addressing. In these cases it would be much simpler to be able to enter address-components in the device's native format. To facilitate this, the -ADDRESSING option is used.

This option takes exactly one parameter which configures the number of bytes contained per address location. If for example a device's program memory naturally used a 16-bit (2 byte) word-addressing format, the option -ADDRESSING=2 will configure HEXMATE to interpret all command line address fields as word addresses. The affect of this setting is global and all HEXMATE options will now interpret addresses according to this setting. This option will allow specification of addressing modes from one byte per address to four bytes per address.

6.6.1.4 -BREAK

This option takes a *comma*-separated list of addresses. If any of these addresses are encountered in the HEX file, the current data record will conclude and a new data record will recommence from the nominated address. This can be useful to use new data records to force a distinction between functionally different areas of program space. Some HEX file readers depend on this.

6.6.1.5 -CK

The -CK option is for calculating a checksum. The usage of this option is:

```
-CK=start-end@destination [+offset][wWidth][tCode][gAlogithm]
```

where:

- *start* and *end* specify the address range over which the checksum will be calculated.
- *destination* is the address where the checksum result will be stored. This value cannot be within the range of calculation.
- *offset* is an optional initial value to add to the checksum result.
- *Width* is optional and specifies the byte-width of the checksum result. Results can be calculated for byte-widths of 1 to 4 bytes. If a positive width is requested, the result will be stored in big-endian byte order. A negative width will cause the result to be stored in little-endian byte order. If the width is left unspecified, the result will be 2 bytes wide and stored in little-endian byte order.
- *Code* is a hexadecimal code that will trail each byte in the checksum result. This can allow each byte of the checksum result to be embedded within an instruction.
- *Algorithm* is an integer to select which HEXMATE algorithm to use to calculate the checksum result. A list of selectable algorithms are given in Table 6-9. If unspecified, the default checksum algorithm used is 8 bit addition (1).

A typical example of the use of the checksum option is:

```
-CK=0-1FFF@2FFE+2100w2
```

This will calculate a checksum over the range 0-1FFFh and program the checksum result at address 2FFEh. The checksum value will be offset by 2100h. The result will be two bytes wide.

TABLE 6-9: HEXMATE CHECKSUM ALGORITHM SELECTION

Selector	Algorithm description
-4	Subtraction of 32 bit values from initial value
-3	Subtraction of 24 bit values from initial value
-2	Subtraction of 16 bit values from initial value
-1	Subtraction of 8 bit values from initial value
1	Addition of 8 bit values from initial value
2	Addition of 16 bit values from initial value
3	Addition of 24 bit values from initial value
4	Addition of 32 bit values from initial value
7	Fletcher's checksum (8 bit)
8	Fletcher's checksum (16 bit)

6.6.1.6 -FILL

The -FILL option is used for filling unused memory locations with a known value. The usage of this option is:

-FILL=Code@*Start-End*[,*data*]

where:

- *Code* is the opcode that will be assigned to unused locations in memory. Multi-byte codes should be entered in little endian order.
- *Start* and *End* specify the address range over which this fill will apply.
- The data flag will specify that only records within the range that contain data will be filled. The default is to fill all records in the range.

For example:

-FILL=3412@0-1FFF,data

will program opcode 1234h in all unused addresses from program memory address 0 to 1FFFh (Note the endianism).

This option accepts whole bytes of hexadecimal data from 1 to 8 bytes in length.

If the data flag has been specified, HEXMATE will only perform ROM filling to records that actually contain data. This means that these records will be padded out to the default data record length or the width specified in the -FORMAT option. Records will also begin on addresses which are multiples of the data record length used. The default data record length is 16 bytes. This facility is particularly useful or is a requirement for some bootloaders that expect that all data records will be of a particular length and address alignment.

6.6.1.7 -FIND

This option is used to detect and log occurrences of an opcode or partial code sequence. The usage of this option is:

-FIND=Findcode [m*Mask*]@*Start-End* [/*Align*][w][t"*Title*"]

where:

- *Findcode* is the hexadecimal code sequence to search for and is entered in little endian byte order.
- *Mask* is optional. It specifies a bit mask applied over the *Findcode* value to allow a less restrictive search. It is entered in little endian byte order.

- *Start* and *End* limit the address range to search.
- *Align* is optional. It specifies that a code sequence can only match if it begins on an address which is a multiple of this value.
- *w*, if present, will cause HEXMATE to issue a warning whenever the code sequence is detected.
- *Title* is optional. It allows a title to be given to this code sequence. Defining a title will make log-reports and messages more descriptive and more readable. A title will not affect the actual search results.

Here are some examples.

The option -FIND=3412@0-7FFF/2w will detect the code sequence 1234h when aligned on a 2 (two) byte address boundary, between 0h and 7FFFh. w indicates that a warning will be issued each time this sequence is found.

In this next example, -FIND=3412M0F00@0-7FFF/2wt"ADDXY", the option is the same as in last example but the code sequence being matched is masked with 000Fh, so HEXMATE will search for any of the opcodes 123xh, where x is any digit. If a byte-mask is used, is must be of equal byte-width to the opcode it is applied to. Any messaging or reports generated by HEXMATE will refer to this opcode by the name, *ADDXY* as this was the title defined for this search.

If HEXMATE is generating a log file, it will contain the results of all searches. -FIND accepts whole bytes of HEX data from 1 to 8 bytes in length. Optionally, -FIND can be used in conjunction with REPLACE or DELETE (as described below).

6.6.1.8 -FIND...,DELETE

If the DELETE form of the -FIND option is used, any matching sequences will be removed. This function should be used with extreme caution and is not normally recommended for removal of executable code.

6.6.1.9 -FIND...,REPLACE

If the REPLACE form of the -FIND option is used, any matching sequences will be replaced, or partially replaced, with new codes. The usage for this sub-option is:

-FIND...,REPLACE=*Code* [m*Mask*]

where:

- *Code* is a little endian hexadecimal code to replace the sequences that match the -FIND criteria.
- *Mask* is an optional bit mask to specify which bits within *Code* will replace the code sequence that has been matched. This may be useful if, for example, it is only necessary to modify 4 bits within a 16-bit instruction. The remaining 12 bits can masked and be left unchanged.

6.6.1.10 -FORMAT

The -FORMAT option can be used to specify a particular variant of INHX format or adjust maximum record length. The usage of this option is:

-FORMAT=*Type* [,*Length*]

where:

- *Type* specifies a particular INHX format to generate.
- *Length* is optional and sets the maximum number of bytes per data record. A valid length is between 1 and 16, with 16 being the default.

Consider the case of a bootloader trying to download an INHX32 file which fails because it cannot process the extended address records which are part of the INHX32 standard. You know that this bootloader can only program data addressed within the

range 0 to 64k, and that any data in the HEX file outside of this range can be safely disregarded. In this case, by generating the HEX file in INHX8M format the operation might succeed. The HEXMATE option to do this would be -FORMAT=INHX8M.

Now consider if the same bootloader also required every data record to contain eight bytes of data, no more, no less. This is possible by combining the -FORMAT with -FILL options. Appropriate use of -FILL can ensure that there are no gaps in the data for the address range being programmed. This will satisfy the minimum data length requirement. To set the maximum length of data records to eight bytes, just modify the previous option to become -FORMAT=INHX8M,8.

The possible types that are supported by this option are listed in Table 6-10. Note that INHX032 is not an actual INHX format. Selection of this type generates an INHX32 file but will also initialize the upper address information to zero. This is a requirement of some device programmers.

TABLE 6-10: INHX TYPES USED IN -FORMAT OPTION

Type	Description
INHX8M	Cannot program addresses beyond 64K
INHX32	Can program addresses beyond 64K with extended linear address records
INHX032	INHX32 with initialization of upper address to zero

6.6.1.11 -HELP

Using -HELP will list all HEXMATE options. By entering another HEXMATE option as a parameter of -HELP will show a detailed help message for the given option. For example:

```
-HELP=string
```

will show additional help for the -STRING HEXMATE option.

6.6.1.12 -LOGFILE

The -LOGFILE option saves HEX file statistics to the named file. For example:

```
-LOGFILE=output.log
```

will analyze the HEX file that HEXMATE is generating and save a report to a file named output.log.

6.6.1.13 -MASK

Use this option to logically AND a memory range with a particular bitmask. This is used to ensure that the unimplemented bits in program words (if any) are left blank. The usage of this option is as follows:

-MASK=*hexcode*@*start*-*end*

Where *hexcode* is a hexadecimal value that will be ANDed with data within the *start* to *end* address range. Multibyte mask values can be entered in little endian byte order.

6.6.1.14 -O*FILE*

The generated Intel HEX output will be created in this file. For example:

```
-Oprogram.hex
```

will save the resultant output to program.hex. The output file can take the same name as one of its input files, but by doing so it will replace the input file entirely.

6.6.1.15 -SERIAL

This option will store a particular HEX value at a fixed address. The usage of this option is:

`-SERIAL=`*`Code`* `[+/-`*`Increment`*`]@`*`Address`* `[+/-`*`Interval`*`][r`*`Repetitions`*`]`

where:

- *`Code`* is a hexadecimal value to store and is entered in little endian byte order.
- *`Increment`* is optional and allows the value of *`Code`* to change by this value with each repetition (if requested).
- *`Address`* is the location to store this code, or the first repetition thereof.
- *`Interval`* is optional and specifies the address shift per repetition of this code.
- *`Repetitions`* is optional and specifies the number of times to repeat this code.

For example:

```
-SERIAL=000001@EFFE
```

will store HEX code `00001h` to address `EFFEh`.

Another example:

```
-SERIAL=0000+2@1000+10r5
```

will store 5 codes, beginning with value `0000` at address `1000h`. Subsequent codes will appear at address intervals of `+10h` and the code value will change in increments of `+2h`.

6.6.1.16 -SIZE

Using the `-SIZE` option will report the number of bytes of data within the resultant HEX image to standard output. The size will also be recorded in the log file if one has been requested.

6.6.1.17 -STRING

The `-STRING` option will embed an ASCII string at a fixed address. The usage of this option is:

`-STRING@`*`Address`* `[t`*`Code`*`]=”`*`Text`*`”`

where:

- *`Address`* is the location to store this string.
- *`Code`* is optional and allows a byte sequence to trail each byte in the string. This can allow the bytes of the string to be encoded within an instruction.
- *`Text`* is the string to convert to ASCII and embed.

For example:

```
-STRING@1000="My favorite string"
```

will store the ASCII data for the string, `My favorite string` (including the nul character terminator) at address `1000h`.

And again:

```
-STRING@1000t34="My favorite string"
```

will store the same string with every byte in the string being trailed with the HEX code `34h`.

6.6.1.18 -STRPACK

This option performs the same function as `-STRING` but with two important differences. Firstly, only the lower seven bits from each character are stored. Pairs of 7 bit characters are then concatenated and stored as a 14 bit word rather than in separate bytes. This is known as string packing. This is usually only useful for devices where program space is addressed as 14 bit words (PIC10/12/16 devices). The second difference is that `-STRING`'s `t` specifier is not applicable with the `-STRPACK` option.