Hexmate User's Guide

Hexmate User's Guide

Notice to Customers

All documentation becomes dated and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions can differ from those in this document. Please refer to our web site (https://www.microchip.com) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXA,” where “XXXXX” is the document number and “A” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

Table of Contents

1. Conventions Used in This Guide

The following conventions may appear in this documentation:

Table 1-1. Documentation Conventions

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file.o`*, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options] file [options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

2. Recommended Reading

This user’s guide describes the use and features of the Hexmate application. The following Microchip documents are available and recommended as supplemental reference resources.

MPLAB® XC8 C Compiler User’s Guide
The Hexmate application is shipped with and automatically executed by the MPLAB XC8 C Compiler. If you are using this compiler to build projects, you can get access to most Hexmate features directly from the compiler. This user's guide indicates the compiler options that will invoke Hexmate functions.

MPLAB® XC8 C Compiler Release Notes for PIC® MCU
Changes made to Hexmate are listed in the MPLAB® XC8 C Compiler Release Notes (an HTML file) in the Docs subdirectory of the compiler’s installation directory. The release notes contain update information and known issues that cannot be included in this user’s guide.

Development Tools Release Notes
For the latest information on using other development tools, refer to the tool-specific Readme files in the docs subdirectory of the MPLAB X IDE installation directory.

3. Development Systems Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata that are related to a specified product family or development tool of interest.

To register, access the Microchip web site at https://www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

Compilers	The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB® C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
Emulators	The latest information on Microchip in-circuit emulators. This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.
In-Circuit Debuggers	The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICkitTM 3 debug express.
MPLAB® IDE	The latest information on Microchip MPLAB IDE, the Windows™ Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.
Programmers	The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are non-production development programmers such as PICSTART® Plus and PICkit 2 and 3.

4. Hexmate

The Hexmate application is a post-link-stage utility designed to merge, reformat, and insert useful information into Intel HEX files.

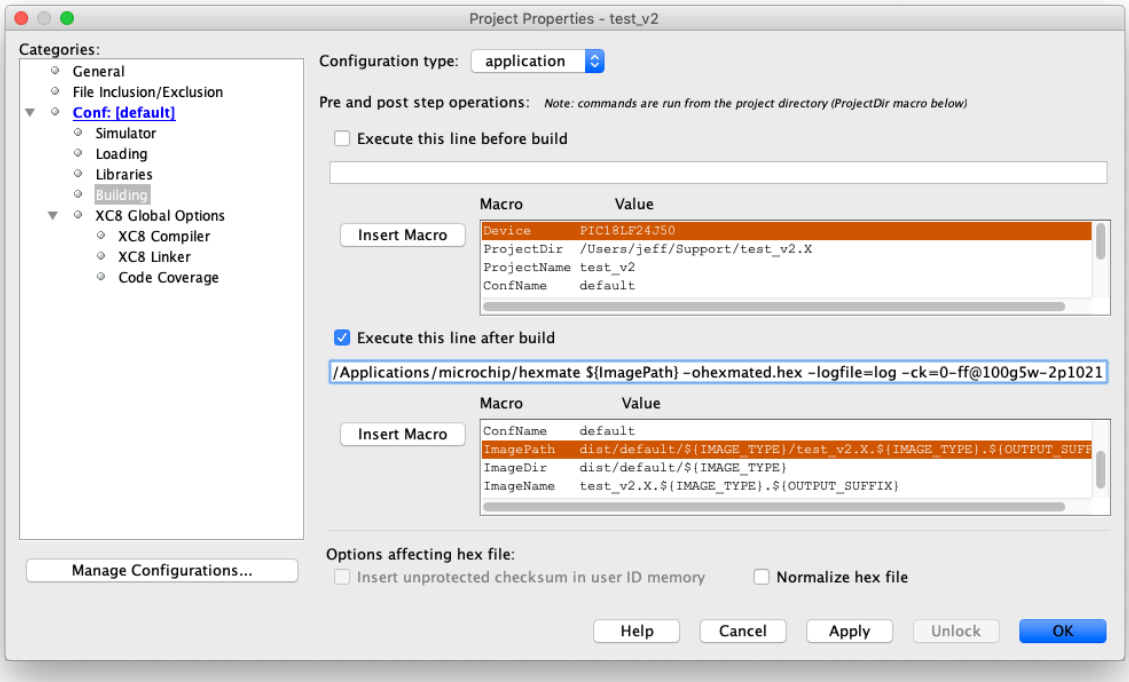
Hexmate is shipped with and automatically executed by the MPLAB XC8 C Compiler, but it can be executed as a stand-alone application to process Intel HEX files produced by any compiler. The MPLAB X IDE also ships with its own copy of Hexmate, which is used to merge HEX files when building loadable projects.

This guide has information on the tasks that Hexmate can perform and the command-line options that control the application.

Although some Hexmate features can be accessed directly from the `xc8-cc` command line driver or from the project properties associated with MPLAB XC8 projects, less commonly used features require Hexmate to be called explicitly after the project is built. Hexmate must also be executed explicitly when used with any HEX file generated from an MPLAB XC16 or XC32 project.

Hexmate can be added as a post-link build command inside the MPLAB X IDE so that you do not need access to a terminal application to run it. In the following screen shot of the Project Properties dialog in the MPLAB X IDE, Hexmate has been executed as a post-build step to calculate a CRC over some of the data in the HEX file produced for a project. The *MPLAB® X IDE User's Guide* has further information on how to set project build properties.

Figure 4-1. Executing Hexmate as a post-build step in the MPLAB X IDE



4.1 Hexmate Uses

Hexmate can be used for a variety of tasks relating to Intel HEX files. These include the following.

- Merging multiple Intel HEX files into one Intel HEX file.
- Calculating and storing variable-length hash values, such as CRC or SHA.
- Filling unused memory locations with known data sequences.
- Converting INHX32 files to other INHX formats (e.g., INHX8M).

- Detecting specific or partial opcode sequences within a HEX file.
- Finding/replacing specific or partial opcode sequences.
- Providing a map of addresses used in a HEX file.
- Changing or fixing the length of data records in a HEX file.
- Validating 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 hash value 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 program counter to a known location if it gets lost.
- Storing a serial number at a fixed address.
- Storing a string (e.g., time stamp) at a fixed address.
- Storing initial values at a particular memory address (e.g., initialize EEPROM).
- Detecting the occurrence of a buggy/restricted instructions.
- Adjusting HEX file to meet the requirements of particular bootloaders.

5. Hexmate Command Line Options

Run Hexmate directly with the following command format:

```
hexmate [specs,]file1.hex [... [specs,]fileN.hex] [options]
```

where *file1*.hex through to *fileN*.hex forms a list of input Intel HEX files to merge using Hexmate.

If only one HEX file is specified, no merging takes place, but other actions can be performed on the HEX file, as specified by additional options. Tabulated below are the command line options that Hexmate accepts.

Table 5-1. Hexmate Command-line Options

Option	Effect
--edf=*file*	Specify the message description file.
--emax=*n*	Set the maximum number of permitted errors before terminating.
--msgdisable=*number*	Disable messages with the numbers specified.
--sla=*address*	Set the start linear address for type 5 records.
--ver	Display version and build information then quit.
-addressing=*units*	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=spec	Calculate and store a hash value.
-fill=*spec*	Program unused locations with a known value.
-find=*spec*	Search and notify if a particular code sequence is detected.
-find=*spec*,delete	Remove the code sequence if it is detected (use with caution).
-find=*spec*,replace=*spec*	Replace the code sequence with a new code sequence.
-format=*type*	Specify maximum data record length or select INHX variant.
-help	Show all options or display help message for specific option.
-logfile=*file*	Save Hexmate analysis of output and various results to a file.
-mask=*spec*	Logically AND a memory range with a bitmask.
-o*file*	Specify the name of the output file.
-serial=*spec*	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=*spec*	Store an ASCII string at a fixed address.
-strpack=*spec*	Store an ASCII string at a fixed address using string packing.
-w*level*	Adjust warning sensitivity.
+	Prefix to any option to overwrite other data in its address range, if necessary.

The format or assumed radix of values associated with options are detailed with each option description. Note that any address fields specified in these options are to be entered as HEX file addresses, unless you use the -addressing option to change this.

5.1 Specifications And Filename

Hexmate can process Intel HEX files that use either INHX32 or INHX8M format. Additional specifications can be applied to each HEX file listed on the command line to place 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`*, where *`Start`* and *`End`* are both assumed to be hexadecimal values. Hexmate will only process data within the address range restriction. For example:

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

will use `myfile.hex` as input, but only process data which is addressed within the range 0x100-1FF (inclusive) from that file.

An address shift can be applied with the specification `s`*`Offset`*, where *`Offset`* is assumed to be an unqualified hexadecimal value. If an address shift is used, data read from this HEX file will be shifted (by the offset specified) to a new address in the output file. The offset can be either positive or negative. For example:

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

will shift the block of data from 0x100-1FF in `myfile.hex` to the new address range 0x2100-21FF in the output.

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

5.2 Override Prefix

When the `+` operator precedes an input file or option, the data obtained from that file or the data generated by that option will take priority and be forced into the output file, overwriting any another other data existing at the same addresses.

For example, if `input.hex` contains data at address 0x1000, the option:

```
input.hex +-string@1000="My string"
```

will have the data specified by the `-string` option placed at address 0x1000 in the output file; however:

```
+input.hex -string@1000="My string"
```

will copy the data contained in the input HEX file at address 0x1000 into the final output.

Without this option, Hexmate will issue an error if two sources try to store differing data at the same location.

5.3 Edf

The `--edf=`*`file`* specifies the message description file to use when displaying warning or error messages. The argument to this option should be the full path to the message file. Hexmate contains an internal copy of the message file, so this option is not normally required. Use this option if you want to specify an alternate file with updated contents.

Message files are shipped with the MPLAB XC8 C Compiler and are located in the compiler's `pic/dat` directory. The English language file is called `en_msgs.txt`.

5.4 Emax

The `--emax=`*`num`* option sets the maximum number of errors Hexmate will display before execution is terminated, e.g., `--emax=25`. By default, up to 20 error messages will be displayed.

5.5 Msgdisable

The `--msgdisable=`*`number`* option allows error, warning or advisory messages to be disabled during execution of Hexmate.

The option is passed a comma-separated list of message numbers that are to be disabled. Any error message numbers in this list are ignored unless they are followed by an `:off` argument. For example:

```
--msgdisable=2031,963
```

If the message list is specified as `0`, then all warnings are disabled.

5.6 Sla

The `--sla=`*`address`* option allows you to specify the linear start address for type 5 records in the HEX output file, e.g., `--sla=0x10000`.

5.7 Ver

The `--ver` option will ask Hexmate to print version and build information and then quit.

5.8 Addressing

The `-addressing=`*`units`* option allows the addressing units of any addresses in Hexmate's command line options to be changed from the default value of 1 to a maximum value of 4.

By default, all address arguments specified in Hexmate options are assumed to be byte addresses, as used by Intel HEX files. For example, in the option `-mask=0F@0-FF`, the mask will be performed on any HEX file data from address 0x0 to address 0xFF. In some device architectures, the native addressing format can be something other than byte addressing. For example, a HEX file might contain the bytes 0x0F and 0x55 at addresses 0x200 and 0x201, respectively, but when this HEX file is loaded into a Mid-range PIC device, these bytes will form one word at address 0x100 in the device. In this case, the word value at each device address expands into two byte values at separate addresses in the HEX file. If you prefer to use device addresses with Hexmate options, use this option to specify the mapping between HEX file addresses and device addresses.

This option takes one parameter that indicates the number of HEX file bytes that will be stored in each device address location. The parameter may range from the values 1 thru 4. For 8-bit AVR devices, Baseline, Mid-range, and 24-bit PIC devices, an addressing unit of 2 can be used, if desired, for example, `-addressing=2`. You may then specify device addresses in all Hexmate options. For all other Microchip devices, you would typically use the default addressing unit of 1 byte, for example use `-addressing=1` or omit this option entirely. For these devices, the HEX file and device addresses for any location are the same and no mapping is required.

5.9 Break

The `-break` option takes a comma-separated list of unqualified hexadecimal 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.

For example, if the output of Hexmate normally contains:

```
:100000000EEF0AF03412ADDEADDEADDEADDEADDEFC
:10001000ADDEADDEADDEADDEADDEADDEADDEADDE88
:10002000ADDEADDEADDEADDEADDEADDEADDEADDE78
...
```

then if the `-break=16` option is used, the output will become:

```
:100000000EEF0AF03412ADDEADDEADDEADDEADDEFC
:06001000ADDEADDEADDE49
```

```
:10001600ADDEADDEADDEADDEADDEADDEADDEADDE82
:10002600ADDEADDEADDEADDEADDEADDEADDEADDE72
...
```

Breaking data records can create a distinction between functionally different areas of the program space. Some HEX file readers depend on records being arranged this way.

5.10 Ck

The `-ck` option is for calculating a hash value. The usage of this option is:

```
-ck=start-end@dest[+offset][wWidth][tCode[.Base]][gAlgorithm][pPolynomial][rRevWidth]
[sSkipWidth[.SkipBytes]]
```

where:

- *`start`* and *`end`* specify the hexadecimal address range over which the hash will be calculated. If these addresses are not a multiple of the data width for checksum and Fletcher algorithms, the value zero will be padded into the relevant input word locations that are missing.
- *`dest`* is the hexadecimal address where the hash result will be stored. This address cannot be within the range of addresses over which the hash is calculated.
- *`offset`* is an optional initial hexadecimal value to be used in the hash calculations. It is not used with SHA algorithms.
- *`Width`* is optional and specifies the decimal width of the result. Results can be calculated for byte-widths of 1 to 4 bytes for most algorithms, but it represents the bit width for SHA algorithms. 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. This width argument is not required with any Fletcher algorithm, as they have fixed widths, but it may be used to alter the default endianism of the result.
- *`Code`* is an optional hexadecimal code sequence that will trail each byte in the result. Use this feature if you need each byte of the hash result to be embedded within an instruction or if the hash value has to be padded to allow the device to read it at runtime. For example, `t34` will embed each byte of the result in a `retlw` instruction (bit sequence 0x34xx) on Mid-range PIC devices. If the code sequence specifies multiple bytes, these are stored in big-endian order after the hash bytes, for example `tAABB` will append 0xAA immediately after the hash byte and 0xBB at the following address. The trailing code specification `t0000` will store two 0x00 bytes after each byte of the hash. The code sequence argument can be optionally followed by `.`*`Base`*, where *`Base`* is the number of bytes of hash to be output before the trailing code sequence is appended. A specification of `t11.2`, for example, will output the byte 0x11 after each *two* bytes of the hash result.
- *`Algorithm`* is a decimal integer to select which Hexmate hash algorithm to use to calculate the result. A list of selectable algorithms is provided in the table below. If unspecified, the default algorithm used is 8-bit checksum addition (algorithm 1).
- *`Polynomial`* is a hexadecimal value which is the polynomial to be used if you have selected a CRC algorithm.
- *`RevWidth`* is an optional reverse word width. If this is non-zero, then bytes within each word are read in reverse order when calculating a hash value. Words are aligned to the addresses in the HEX file. At present, the width must be 0 or 2. A zero width disables the reverse-byte feature, as if the `r` suboption was not present. This suboption is intended for situations when Hexmate is being used to match a CRC produced by a PIC hardware CRC module that uses the Scanner module to stream data to it. This feature will not affect the hash result when using any checksum algorithm (algorithms -4 thru 4).
- *`SkipWidth`* is an optional skip word width. If this is non-zero, then the byte at the highest address within each word is skipped for the purposes of calculating a hash value. Words are aligned to the addresses in the HEX file. At present, the width must be 0 (which disables the skip feature, as if the `s` suboption was not present) or greater than 1. This skip width argument can be optionally followed by `.`*`SkipBytes`*, where *`SkipBytes`* is a number representing the number of bytes to skip in each word, for example `s4.2` will skip the two bytes at the highest addresses in each 4-byte word. To avoid processing the 'phantom' 0x00 bytes added to HEX files by the MPLAB XC16 C Compiler in hash calculations, use `s4`.

A typical example of the use of this option to calculate a checksum is:

```
-ck=0-1FFF@2FFE+2100w-2g2
```

This will calculate a checksum (16-bit addition) over the range 0 to 0x1FFF and program the checksum result at address 0x2FFE. The checksum value will be offset by 0x2100. The result will be two bytes wide and stored in little-endian format.

Note that the reverse and skip features act on words that are aligned to the HEX file addresses, not to the starting byte of data in the sequence being processed. In other words, the positions of the words are not affected by the start and end addresses specified in the `-ck` option. Consider this option:

```
-ck=0-5@100w2g5p1021s2
```

which specifies that when calculating the hash value, every second byte be skipped (`s2`) over HEX addresses 0 thru 5. If it is acting on the HEX record (data underlined):

```
:10000000640025000030A75007600770078006400 1C
```

the hash will be calculated from the (hexadecimal) bytes 64, 25, and 03. Processing the same HEX record with an option that uses a different start and end address range (1 thru 6):

```
-ck=1-6@100w2g5p1021s2
```

the hash will be calculated from the (hexadecimal) bytes 25, 03, and 75. These features attempt to mimic data read limitations of code running on the device, and thus the words they use are aligned with device addresses, which are in turn aligned to HEX file addresses.

Table 5-2. Hexmate Hash Algorithm Selection

Selector	Algorithm Description
-5	Reflected cyclic redundancy check (CRC).
-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.
5	Cyclic redundancy check (CRC).
7	Fletcher’s checksum (8 bit calculation, 2-byte result width).
8	Fletcher’s checksum (16 bit calculation, 4-byte result width).
10	SHA-2 (currently only SHA256 is supported).

See 7. Hash Value Calculations for more details about the algorithms that are used to calculate hashes.

5.11 Fill

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

```
-fill=[wconst_width:]fill_expr@address[:end_address]
```

where:

- *const_width* signifies the decimal width (*n* bytes) of each constant in *fill_expr* and can range from 1 thru 9. If this width is not specified, the default value is two bytes. For example, `-fill=w1:1` with fill every unused byte with the value 0x01.
- *fill_expr* defines the values to fill and consists of *const*, which is a base value to place in the first memory location and optionally with *increment*, which indicates how this base value should change after each use. If the base value specifies more than one byte, these are stored in little-endian byte order. These following show the possible fill expressions:
  - *const* fill memory with a repeating constant; i.e., `-fill=0xBEEF` fills with the values 0xBEEF, 0xBEEF, 0xBEEF, 0xBEEF.
  - *const*+=*increment* fill memory with an incrementing constant; i.e., `-fill=0xBEEF+=1` fills with 0xBEEF, 0xBEF0, 0xBEF1, 0xBEF2.
  - *const*-=*increment* fill memory with a decrementing constant; i.e., `-fill=0xBEEF-=0x10` fills with 0xBEEF, 0xBEDF, 0xBECF, 0xBEBF.
  - *const*,*const*,...,*const* fill memory with a list of repeating constants; i.e., `-fill=0xDEAD,0xBEEF` fills with 0xDEAD,0xBEEF,0xDEAD,0xBEEF.
- The options following *fill_expr* result in the following behavior:
  - @*address* fill a specific address with *fill_expr*; i.e., `-fill=0xBEEF@0x1000` puts 0xBEEF at address 1000h. If the fill value is wider than the addressing value specified with `-addressing`, then only part of the fill value is placed in the output. For example, if the addressing is set to 1, the option above will place 0xEF at address 0x1000 and a warning will be issued.
  - @*address*:*end_address* fill a range of memory with *fill_expr*; i.e., `-fill=0xBEEF@0:0xFF` puts 0xBEEF in unused addresses between 0 and 255. If the address range (multiplied by the `-addressing` value) is not a multiple of the fill value width, the final location will only use part of the fill value, and a warning will be issued.

The fill values are word-aligned so they start on an address that is a multiple of the fill width. Should the fill value be an instruction opcode, this alignment ensures that the instruction can be executed correctly. Similarly, if the total length of the fill sequence is larger than 1 (and even if the specified width is 1), the fill sequence is aligned to that total length. For example the following fill option, which specifies 2 bytes of fill sequence and a starting address that is not a multiple of 2:

```
-fill=w1:0x11,0x22@0x11001:0x1100c
```

will result in the following HEX record, where the starting address was filled with the second byte of the fill sequence due to this alignment.

```
:0C1001002211221122112211221122112211B1
```

All fill constants (excluding the width specification) can be expressed in (unsigned) binary, octal, decimal or hexadecimal, as per normal C syntax, for example, 1234 is a decimal value, 0xFF00 is hexadecimal and FF00 is illegal.

5.12 Find

The `-find=`*opcode* option is used to detect and log occurrences of an opcode or code sequence. The usage of this option is:

```
-find=Findcode[mMask]@Start-End[/Align][w][t"Title"]
```

where:

- *Findcode* is the hexadecimal code sequence to search for. For example, to find a `clrf` instruction with the opcode 0x01F1, use `01F1` as the sequence. In the HEX file, this will appear as the byte sequence F1 01, that is 0xF1 at HEX address 0 and 0x01 at HEX address 1.
- *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 that 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.

All numerical arguments are assumed to be hexadecimal values.

Here are some examples.

The option `-find=1234@0-7FFF/2w` will detect the code sequence 1234h (stored in the HEX file as 34 12) 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=1234M0F00@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 `123`*x*h, where *x* is any digit. If a byte-mask is used, it 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.

When requested, a log file will contain the results of all searches. The `-find` option 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 separately).

5.13 Find And Delete

If the `delete` form of the `-find` option is used, any matching sequences will be deleted from the output file. This implies removal of the data entirely, not replacing it with zero bytes.

To have the `-find` option perform deletion, append `,delete` to the option, for example:

```
-find=ff@7fe0-7fff,delete
```

This function should be used with extreme caution and is not normally recommended for removal of executable code.

5.14 Find and Replace

If the `replace` form of the `-find` option is used, any matching sequences will be replaced, or partially replaced, with new codes.

To have the `-find` option perform replacement, append `,replace=`*spec* to the option in the following manner.

```
-find=spec,replace=Code[mMask]
```

where:

- *Code* is a hexadecimal code sequence 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 can 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 left unchanged.

For example:

```
-find=ff@7fe0-7fff,replace=55
```

This function should be used with extreme caution.

5.15 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 decimal, 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 that are part of the INHX32 standard. This bootloader can only program data addressed within the range 0 to 64k and 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 exactly 8 bytes of data. 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 8 bytes, just modify the previous option to become `-format=INHX8M,8`.

The possible types that are supported by this option are listed in Table 5-3. 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 5-3. Inhx Types

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.

5.16 Help

Using `-help` will list all Hexmate options. 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.

5.17 Logfile

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

```
-logfile=output.hxl
```

will analyze the HEX file that Hexmate is generating and save a report to a file named `output.hxl`.

5.18 Mask

Use the `-mask=spec` 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 value that will be ANDed with data within the *start* to *end* address range. All values are assumed to be hexadecimal. Multibyte mask values can be entered in little endian byte order.

5.19 O: Specify Output File

When using the `-ofile` option, the generated Intel HEX output will be created in the specified 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.

If this option is used without a filename, no output is produced, which may be useful if you want to use Hexmate to only show the size of a HEX file, for example. If this option is not used at all, the content of the output HEX file is printed to the standard output stream.

5.20 Serial

The `-serial=specs` option will store a particular HEX value sequence at a fixed address. The usage of this option is:

```
-serial=Code[+/-Increment]@Address[+/-Interval][rRepetitions]
```

where:

- *Code* is a hexadecimal sequence to store. The first byte specified is stored at the lowest address.
- *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.

All numerical arguments are assumed to be hexadecimal values, except for the *Repetitions* argument, which is decimal value by default.

For example:

```
-serial=000001@EFFE
```

will store HEX code 0x00001 to address 0xEFFE.

Another example:

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

will store 5 codes, beginning with value 0000 at address 0x1000. Subsequent codes will appear at address intervals of +0x10 and the code value will change in increments of +0x2.

5.21 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.

5.22 String

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

```
-string@Address[tCode]="Text"
```

where:

- *Address* is assumed to be a hexadecimal value representing the address at which the string will be stored.
- *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 null character terminator), at address 0x1000.

And again:

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

will store the same string, trailing every byte in the string with the HEX code 0x34.

5.23 Strpack

The `-strpack=`*spec* option performs the same function as `-string`, but with two important differences.

Whereas `-string` stores the full byte corresponding to each character, `-strpack` stores only the lower seven bits from each character. 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 often useful for Mid-range PIC devices, where the program memory is addressed as 14-bit words. If you intend to use this option, you must ensure that the encoded characters are fully readable and correctly interpreted at runtime.

The second difference between these two options is that the `t` specifier usable with `-string` is not applicable with the `-strpack` option.

5.24 W: Specify warning level

The `-w`*level* option sets a warning level threshold. The *level* value can be a digit from -9 thru 9, for example `-w5`.

The warning level determines how pedantic Hexmate is about dubious requests or file content. Each warning has a designated warning level; the higher the warning level, the more important the warning message. If the warning message's warning level exceeds the threshold set with this option, the warning is printed. The default warning level threshold is 0 and will allow all normal warning messages.

6. Internal Operation

Hexmate can perform many operations on the HEX file(s) it is processing. The following operations are listed in the order in which they are performed. An appreciation of this order is important, as it can affect the output.

- Create an internal master image by merging the data contained in all the input HEX files, processing them in the order in which they appear on the command-line and as per the read specification associated with each file.
- Find values specified using `-find` options in the master image, deleting and replacing these values as requested in the reverse order in which the options were issued.
- Insert serial data specified using `-serial` options into the master image in the reverse order in which the options were issued.
- Embed strings specified using `-string` and `-strpack` options into the master image in the reverse order in which the options were issued.
- Reserve space for hashes requested using `-ck` options and trailing codes (`t` argument to `-ck`) in the master image.
- Fill unused locations specified using `-fill` options in the master image in the order in which the options were issued.
- Calculate each specified hash from the master image and insert these and any trailing codes into the master image in the order in which the options were issued.
- Encode the final master image into an Intel HEX file format and output this file as dictated by the `-o` option.

Masking of data specified using `-mask` options is performed whenever the master image is modified, whether the written data has come from an input HEX file or the result of a Hexmate operation being stored in the master image.

7. Hash Value Calculations

A hash value is a small fixed-size value that is calculated from, and used to represent, all the values in an arbitrary-sized block of data. If that data block is copied, a hash recalculated from the new block can be compared to the original hash. Agreement between the two hashes provides a high level of certainty that the copy is valid. There are many hash algorithms. More complex algorithms provide a more robust verification, but can sometimes be too computationally demanding when used in an embedded environment, particularly for smaller devices.

Hexmate can be used to calculate the hash of a program image that is contained in a HEX file. This hash can be embedded into that same HEX file and burned into the target device along with the program image. At runtime, the target device can run a similar hash algorithm over the program image, now stored in its memory. If the stored and calculated hashes are the same, the embedded program can assume that it has a valid program image to execute.

Hexmate implements several hash algorithms, such as checksums and cyclic redundancy checks, which can be selected to calculate the hash value.

When executing Hexmate explicitly, the `-ck` option requests that a hash be calculated, as described in 5.10 Ck.

Some consideration is required when a hash value is being calculated over memory that contains unused memory locations. When executing Hexmate explicitly, consider using the `-fill` option (see 5.11 Fill) to have these locations programmed with a known value.

Hexmate can produce a hash value from any Intel HEX file, regardless of which compiler produced the file and which device that file is intended to program. However, the architecture of the target device may restrict which memory locations can be read at runtime, thus requiring modification to the way in which Hexmate should perform hash calculations, so that the two hashes are calculated similarly and agree. In addition, some compilers might insert padding or phantom bytes into the HEX file that are not present in the device memory. These bytes might need to be ignored by Hexmate when it calculates a hash value and the following discussion indicates possible solutions.

Not all devices can read the entire width of their program memory. For example, Baseline and Mid-range PIC devices can only read the lower byte of each program memory location. The HEX file, however, will contain two bytes for each program memory word and both these bytes will normally be processed by Hexmate when calculating a hash value. When executing Hexmate explicitly, use the `s2` suboption to the `-ck` option to have the MSB of each 2-byte word skipped. Note, however, that this sort of verification process is not considering corruption in the MSB of each program word.

To accommodate the 24-bit (3 byte) program memory word size on dsPIC and PIC24 devices, the MPLAB XC16 C compiler inserts a 0x00 phantom byte after each 3-byte instruction to make up a 4-byte word. Hexmate will normally see and process these phantom bytes when calculating a hash value, whereas code running on the device to perform the same calculation might not. When executing Hexmate explicitly, use the `s4` suboption to the `-ck` option to have the MSB of each 4-byte word skipped for these devices.

Some devices have hardware CRC modules which can calculate a CRC hash value. If desired, program memory data can be streamed to this module using the Scanner module to automate the calculation. As the Scanner module reads the MSB of each program memory word first, you need to have Hexmate also process HEX file bytes within an instruction word in the reverse order. When executing Hexmate explicitly, use the `r2` suboption to the `-ck` option to have Hexmate process bytes in a 2-byte word in reverse order.

Some consideration must also be given to how the Hexmate hash value encoded in the HEX file can be read at runtime.

Baseline and Mid-range PIC devices must store data in program memory using `retlw` instructions. Thus they need one instruction to store each byte of the hash value calculated by Hexmate. When executing Hexmate explicitly, use the `t34` suboption to the `-ck` option to have Hexmate process store each bytes as part of a `retlw` instruction.

The dsPIC and PIC24 devices cannot read the full width of their program memory and data to be loaded to this memory is typically stored in 2-byte chunks per every 4 bytes of HEX file data.When executing Hexmate explicitly, use the `t0000.2` suboption to the `-ck` option to have Hexmate store two bytes of the hash value in the lower half of each 4-bytes of the HEX file, with the upper bytes set to zero.

7.1 Hash Algorithms

The following sections provide examples of the algorithms that Hexmate uses and that can be used to calculate the corresponding hash value at runtime. Note that these examples may require modification for the intended device and situation.

7.1.1 Addition Algorithms

Hexmate has several simple checksum algorithms that sum data values over a range in the program image. These algorithms correspond to the selector values 1, 2, 3 and 4 in the algorithm suboption and read the data in the program image as 1, 2, 3 or 4 byte quantities, respectively. This summation is added to an initial value (offset) that is supplied to the algorithm via the same option. The width to which the final checksum is truncated is also specified by this option and can be 1, 2, 3 or 4 bytes. Hexmate will automatically store the checksum in the HEX file at the address specified in the checksum option.

The function shown below can be customized to work with any combination of data size (`read_t`) and checksum width (`result_t`).

```
#include <stdint.h>
typedef uint8_t read_t;        // size of data values read and summed
typedef uint16_t result_t;     // size of checksum result

// add to offset, n additions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to sum
// n:    the number of sums to perform
// offset: the intial value to which the sum is added
result_t ck_add(const read_t *data, unsigned n, result_t offset)
{
    result_t chksum;
    chksum = offset;
    while(n--) {
        chksum += *data;
        data++;
    }
    return chksum;
}
```

The `read_t` and `result_t` type definitions should be adjusted to suit the data read/sum width and checksum result width, respectively. If you never use an offset, that parameter can be removed and `chksum` assigned 0 before the loop.

Here is how this function might be used when, for example, a 2-byte-wide checksum is to be calculated from the addition of 1-byte-wide values over the address range 0x100 to 0x7fd, starting with an offset of 0x20. The checksum is to be stored at 0x7fe and 0x7ff in little endian format.

When executing Hexmate explicitly, use the option:

```
-ck=100-7fd@7fe+20g1w-2
```

Adapt the following MPLAB XC8 code snippet for PIC devices, which calls `ck_add()` and compares the runtime checksum with that stored by Hexmate at compile time.

```
extern const read_t ck_range[0x6fe/sizeof(read_t)] __at(0x100);
extern const result_t hexmate __at(0x7fe);
result_t result;

result = ck_add(ck_range, sizeof(ck_range)/sizeof(read_t), 0x20);
if(result != hexmate)
    ck_failure();  // take appropriate action
```

This code uses the placeholder array, `ck_range`, to represent the memory over which the checksum is calculated and the variable `hexmate` is mapped over the locations where Hexmate will have stored its checksum result. Being `extern` and absolute, neither of these objects consume additional device memory. Adjust the addresses and sizes of these objects to match the option you pass to Hexmate.

Hexmate can calculate a checksum over any address range; however, the test function, `ck_add()`, assumes that the start and end address of the range being summed are a multiple of the `read_t` width. This is a non-issue if the size of `read_t` is 1. It is recommended that your checksum specification adheres to this assumption, otherwise you will need to modify the test code to perform partial reads of the starting and/or ending data values. This will significantly increase the code complexity.

7.1.2 Subtraction Algorithms

Hexmate has several checksum algorithms that subtract data values over a range in the program image. These algorithms correspond to the selector values -1, -2, -3, and -4 in the algorithm suboption and read the data in the program image as 1-, 2-, 3- or 4-byte quantities, respectively. In other respects, these algorithms are identical to the addition algorithms. See 7.1.1 Addition Algorithms for further information regarding the subtraction algorithms.

The function shown below can be customized to work with any combination of data size (`read_t`) and checksum width (`result_t`).

```
#include <stdint.h>
typedef uint8_t read_t;        // size of data values read and subtracted
typedef uint16_t result_t;     // size of checksum result

// add to offset n subtractions of values starting at address data,
// truncating and returning the result
// data: the address of the first value to subtract
// n:    the number of subtractions to perform
// offset: the intial value to which the subtraction is added
result_t ck_sub(const read_t *data, unsigned n, result_t offset)
{
    result_t chksum;
    chksum = offset;
    while(n--) {
        chksum -= *data;
        data++;
    }
    return chksum;
}
```

Here is how this function might be used when, for example, a 4-byte-wide checksum is to be calculated from the addition of 2-byte-wide values over the address range 0x0 to 0x7fd, starting with an offset of 0x0. The checksum is to be stored at 0x7fe and 0x7ff in little endian format.

When executing Hexmate explicitly, use the option:

```
-ck=0-7fd@7fe+0g-2w-4
```

Adapt the following MPLAB XC8 code snippet for PIC devices, which calls `ck_sub()` and compares the runtime checksum with that stored by Hexmate at compile time.

```
extern const read_t ck_range[0x7fe/sizeof(read_t)] __at(0x0);
extern const result_t hexmate __at(0x7fe);
result_t result;

result = ck_sub(ck_range, sizeof(ck_range)/sizeof(read_t), 0x0);
if(result != hexmate)
    ck_failure();  // take appropriate action
```

7.1.3 Fletcher Algorithms

Hexmate has several algorithms that implement Fletcher's checksum. These algorithms are more complex, providing a robustness approaching that of a cyclic redundancy check, but with less computational effort. There are two forms of this algorithm which correspond to the selector values 7 and 8 in the algorithm suboption and which implement a 1-byte calculation and 2-byte result, with a 2-byte calculation and 4-byte result, respectively. Hexmate will automatically store the checksum in the HEX file at the address specified in the checksum option.

The function shown below performs a 1-byte-wide addition and produces a 2-byte result.

```
#include <stdint.h>
typedef uint16_t result_t;     // size of fletcher result
```

```
result_t
fletcher8(const unsigned char * data, unsigned int n )
{
    result_t sum = 0xff, sumB = 0xff;
    unsigned char tlen;
    while (n) {
        tlen = n > 20 ? 20 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xff) + (sum >> 8);
        sumB = (sumB & 0xff) + (sumB >> 8);
    }
    sum = (sum & 0xff) + (sum >> 8);
    sumB = (sumB & 0xff) + (sumB >> 8);
    return sumB << 8 | sum;
}
```

Here is how this function might be used when, for example, a 2-byte-wide Fletcher hash is to be calculated over the address range 0x100 to 0x7fb, starting with an offset of 0x20. The checksum is to be stored at 0x7fc thru 0x7ff in little endian format.

When executing Hexmate explicitly, use the following option. Note that the width cannot be controlled with the `w` suboption, but the sign of this suboption's argument is used to indicate the required endianism of the result.

```
-ck=100-7bd@7fc+20g8w-4
```

This code can be called in a manner similar to that shown for the addition algorithms (see 7.1.1 Addition Algorithms).

The code for the 2-byte-addition Fletcher algorithm, producing a 4-byte result is shown below.

```
#include <stdint.h>
typedef uint32_t result_t;     // size of fletcher result

result_t
fletcher16(const unsigned int * data, unsigned n)
{
    result_t sum = 0xffff, sumB = 0xffff;
    unsigned tlen;
    while (n) {
        tlen = n > 359 ? 359 : n;
        n -= tlen;
        do {
            sumB += sum += *data++;
        } while (--tlen);
        sum = (sum & 0xffff) + (sum >> 16);
        sumB = (sumB & 0xffff) + (sumB >> 16);
    }
    sum = (sum & 0xffff) + (sum >> 16);
    sumB = (sumB & 0xffff) + (sumB >> 16);
    return sumB << 16 | sum;
}
```

7.1.4 CRC Algorithms

Hexmate has several algorithms that implement the robust cyclic redundancy checks (CRC). There is a choice of two algorithms that correspond to the selector values 5 and -5 in the algorithm suboption and that implement a CRC calculation and reflected CRC calculation, respectively. The reflected algorithm works on the least significant bit of the data first.

The polynomial to be used and the initial value can be specified in the option. Hexmate will automatically store the CRC result in the HEX file at the address specified in the checksum option.

Some devices implement a CRC module in hardware that can be used to calculate a CRC at runtime. These modules can stream data read from program memory using a Scanner module. To ensure that the order of the bytes processed by Hexmate and the CRC/Scanner module are identical, you must specify a reverse word width of 2, which will read each 2-byte word in the HEX file in order, but process the bytes within those words in reverse order. When running Hexmate explicitly, use the `r2` suboption to `-ck`.

The function shown below can be customized to work with any result width (`result_t`). It calculates a CRC hash value using the polynomial specified by the `POLYNOMIAL` macro.

```
#include <stdint.h>
typedef uint16_t result_t;     // size of CRC result
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(result_t))
#define MSb     ((result_t)1 << (WIDTH - 1))

result_t
crc(const unsigned char * data, unsigned n, result_t remainder) {
    unsigned pos;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++) {
        remainder ^= ((result_t)data[pos] << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    return remainder;
}
```

The `result_t` type definition should be adjusted to suit the result width.

Here is how this function might be used when, for example, a 2-byte-wide CRC hash value is to be calculated values over the address range 0x0 to 0xFF, starting with an initial value of 0xFFFF. The result is to be stored at 0x100 and 0x101 in little endian format.

When executing Hexmate explicitly, use the option:

```
-ck=0-ff@100+ffffg5w-2p1021
```

Adapt the following MPLAB XC8 code snippet for PIC devices, which calls `crc()` and compares the runtime hash result with that stored by Hexmate at compile time.

```
extern const unsigned char ck_range[0x100] __at(0x0);
extern const result_t hexmate __at(0x100);
result_t   result;

result = crc(ck_range, sizeof(ck_range), 0xFFFF);
if(result != hexmate){
    // something's not right, take appropriate action
    ck_failure();
}
// data verifies okay, continue with the program
```

The reflected CRC result can be calculated by reflecting the input data and final result, or by reflecting the polynomial. The functions shown below can be customized to work with any result width (`result_t`). The `crc_reflected_IO()` function calculates a reflected CRC hash value by reflecting the data stream bit positions. Alternatively, the `crc_reflected_poly()` function does not adjust the data stream but reflects instead the polynomial, which in both functions is specified by the `POLYNOMIAL` macro. Both functions use the `reflect()` function to perform bit reflection.

```
#include <stdint.h>
typedef uint16_t result_t;     // size of CRC result
typedef unsigned char read_t;
typedef unsigned int reflectWidth;
// This is the polynomial used by the CRC-16 algorithm we are using.
#define POLYNOMIAL     0x1021
#define WIDTH   (8 * sizeof(result_t))
#define MSb     ((result_t)1 << (WIDTH - 1))
#define LSb     (1)
#define REFLECT_DATA(X)       ((read_t) reflect((X), 8))
#define REFLECT_REMAINDER(X)  (reflect((X), WIDTH))

reflectWidth
```

```
reflect(reflectWidth data, unsigned char nBits)
{
    reflectWidth reflection = 0;
    reflectWidth reflectMask = (reflectWidth)1 << nBits - 1;
    unsigned char bitp;
    for (bitp = 0; bitp != nBits; bitp++) {
        if (data & 0x01) {
            reflection |= reflectMask;
        }
        data >>= 1;
        reflectMask >>= 1;
    }
    return reflection;
}

result_t
crc_reflected_IO(const unsigned char * data, unsigned n, result_t remainder) {
    unsigned pos;
    unsigned char reflected;
    unsigned char bitp;
    for (pos = 0; pos != n; pos++) {
        reflected = REFLECT_DATA(data[pos]);
        remainder ^= ((result_t)reflected << (WIDTH - 8));
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & MSb) {
                remainder = (remainder << 1) ^ POLYNOMIAL;
            } else {
                remainder <<= 1;
            }
        }
    }
    remainder = REFLECT_REMAINDER(remainder);
    return remainder;
}

result_t
crc_reflected_poly(const unsigned char * data, unsigned n, result_t remainder) {
    unsigned pos;
    unsigned char bitp;
    result_t rpoly;
    rpoly = reflect(POLYNOMIAL, WIDTH);
    for (pos = 0; pos != n; pos++) {
        remainder ^= data[pos];
        for (bitp = 8; bitp > 0; bitp--) {
            if (remainder & LSb) {
                remainder = (remainder >> 1) ^ rpoly;
            } else {
                remainder >>= 1;
            }
        }
    }
    return remainder;
}
```

Here is how this function might be used when, for example, a 2-byte-wide reflected CRC result is to be calculated over the address range 0x0 to 0xFF, starting with an initial value of 0xFFFF. The result is to be stored at 0x100 and 0x101 in little endian format.

When executing Hexmate explicitly, instead use the following option, noting that the algorithm selected is negative 5 in this case.

```
-ck=0-ff@100+ffffg-5w-2p1021
```

In your project, call either the `crc_reflected_IO()` or `crc_reflected_poly()` functions, as shown previously.

7.1.5 SHA Algorithms

Hexmate implements a secure hash algorithm (SHA). The selector value 10 selects the SHA256 algorithm, a 256-bit variant of the SHA-2 algorithm.

The code to implement a SHA256 is more complex than other algorithms supported by Hexmate, and as its name suggests, the result of such a hash is 256 bits (32 bytes) wide. Public-domain implementations of this algorithm are available for download from third-party websites, such as github.com/B-Con/crypto-algorithms.

Here is how Hexmate might be used when, for example, a SHA256 hash value is to be calculated values over the address range 0x0 to 0x1FF. The result is to be stored at a starting address of 0x1000 in little endian format.

```
-ck=0-1ff@1000g10w-256
```

8. Document Revision History

Revision A (September 2020)

- Initial release of this document, adapted from content in the MPLAB® XC8 C Compiler User's Guide, DS 50002053.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

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.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6685-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820