Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

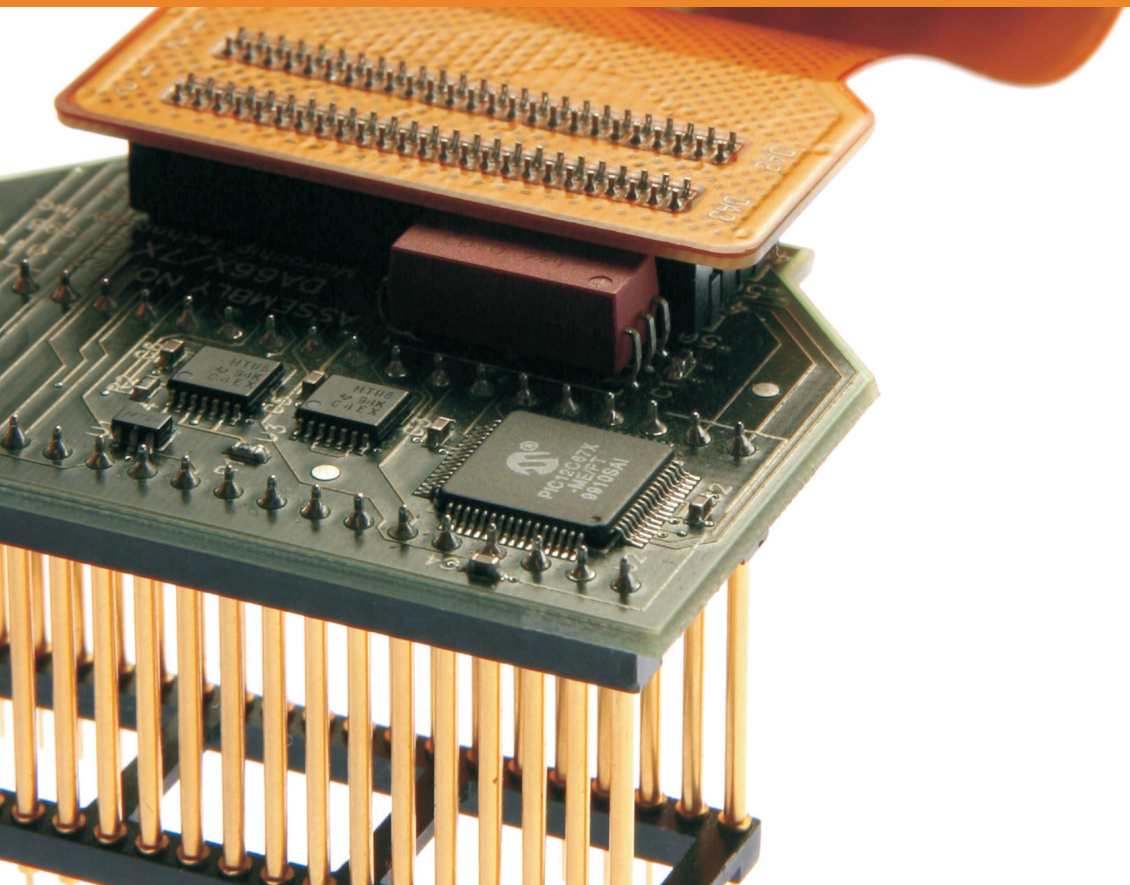

## Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

HI-TECH
SOFTWARE

# PICC™ STD

ANSI C COMPILER




# HI-TECH PICC STD Compiler

## HI-TECH Software.


PICC is licensed exclusively to HI-TECH Software
by Microchip Technology Inc.
Produced on: February 21, 2008

HI-TECH Software Pty. Ltd.
ACN 002 724 549
45 Colebard Street West
Acacia Ridge QLD 4110
Australia

email: hitech@htsoft.com
web: http://www.htsoft.com
ftp: ftp://www.htsoft.com

# Contents

# List of Tables

# Chapter 1

# Introduction

## 1.1 Typographic conventions

Different fonts and styles are used throughout this manual to indicate special words or text. Computer prompts, responses and filenames will be printed in `constant-spaced` type**.** When the filename is the name of a standard header file, the name will be enclosed in angle brackets, e.g. `<stdio.h>`. These header files can be found in the `INCLUDE` directory of your distribution.

Samples of code, C keywords or types, assembler instructions and labels will also be printed in a `constant-space` type. Assembler code is printed in a font similar to that used by C code.

Particularly useful points and new terms will be emphasized using *italicized type*. When part of a term requires substitution, that part should be printed in the appropriate font, but in *italics*. For example: `#include <`*`filename.h`*`>`.

# Chapter 2

# PICC Command-line Driver

PICC is the driver invoked from the command line to compile and/or link C programs. PICC has the following basic command format:

```
PICC [options] files [libraries]
```

It is conventional to supply the options (identified by a leading *dash* "-" or *double dash* "–") before the filenames.

The options are discussed below. The files may be a mixture of source files (C or assembler) and object files. The order of the files is not important, except that it will affect the order in which code or data appears in memory. *Libraries* are a list of library names, or -L options, see Section 2.4.6. Source files, object files and library files are distinguished by PICC solely by the *file type* or *extension*. Recognized file types are listed in Table 2.1. This means, for example, that an assembler file must always have a .as extension (alphabetic case is not important).

PICC will check each file argument and perform appropriate actions. C files will be compiled; assembler files will be assembled. At the end, unless suppressed by one of the options discussed later,

Table 2.1: PICC file types

| File Type | Meaning |
|---|---|
| .c | C source file |
| .as | Assembler source file |
| .obj | Relocatable object code file |
| .lib | Relocatable object library file |
| .hex | Intel HEX file |

all object files resulting from compilation or assembly, or those listed explicitly on the command line, will be linked together with the standard runtime code and libraries and any user-specified libraries. Functions in libraries will be linked into the resulting output file only if referenced in the source code.

Invoking `PICC` with only object files specified as the file arguments (i.e. no source files) will mean only the link stage is performed. It is typical in Makefiles to use `PICC` with a `-C` option to compile several source files to object files, then to create the final program by invoking `PICC` again with only the generated object files and appropriate libraries (and appropriate options). If a .lib output file type is selected, the object files will be stored in a library instead of going through to the final link.

When a HEX file is given on the command line, `PICC` will invoke the HEXMATE utility and will merge the named hex file with the hex file currently being generated. This feature can be useful when, for example, a single hex file is desired which contains a bootloader and application program.

## 2.1 Long Command Lines

The PICC driver is capable of processing command lines exceeding any operating system limitation. To do this, the driver may be passed options via a command file. The command file is read by using the `@` symbol. For example:

```
PICC @xyz.cmd
```

## 2.2 Default Libraries

PICC will search the appropriate standard C library by default for symbol definitions. This will always be done last, after any user-specified libraries. The particular library used will be dependent on the processor selected.

## 2.3 Standard Runtime Code

PICC will automatically generate standard runtime start-up code appropriate for the processor and options selected unless you have specified the to disable this via the `--RUNTIME` option. If you require any special powerup initialization, you should use the *powerup* routine feature (see Section 3.3.4.4).

## 2.4 PICC Compiler Options

Most aspects of the compilation can be controlled using the command-line driver, PICC. The driver will configure and execute all required applications, such as the code generator, assembler and linker.

PICC recognizes the compiler options listed in the table below. The case of the options is not important, however command shells in UNIX based operating systems are case sensitive when it comes to names of files.

PICC Command-line Options

| Option | Meaning |
|---|---|
| -C | Compile to object files only |
| -D*macro* | Define preprocessor macro |
| -E+*file* | Redirect and optionally append errors to a file |
| -G*file* | Generate source-level debugging information |
| -I*path* | Specify a directory pathname for include files |
| -L*library* | Specify a library to be scanned by the linker |
| -L-*option* | Specify *-option* to be passed directly to the linker |
| -M*file* | Request generation of a MAP file |
| -N*size* | Specify identifier length |
| -O*file* | Set basename for output files |
| -P | Preprocess assembler files |
| -Q | Specify quiet mode |
| -S | Compile to assembler source files only |
| -U*symbol* | Undefine a predefined preprocessor symbol |
| -V | Verbose: display compiler pass command lines |
| -X | Eliminate local symbols from symbol table |
| --ASMLIST | Generate assembler .LST file for each compilation |
| --CHAR=*type* | Make the default char signed or unsigned |
| --CHECKSUM=*start-end@dest* | Calculate a checksum over an address range |
| --CHIP=*processor* | Selects which processor to compile for |
| --CHIPINFO | Displays a list of supported processors |
| --CODEOFFSET=*address* | Offset program code to address |
| --CR=*file* | Generate cross-reference listing |
| --DEBUGGER=*type* | Select the debugger that will be used |
| --DOUBLE=*type* | Selects size/kind of double types |
| --ECHO | Echo the PICC command line |
| --ERRFORMAT<=*format*> | Format error message strings to the given style |
| *continued...* | |

PICC Command-line Options

| Option | Meaning |
|---|---|
| --ERRORS=*number* | Sets the maximun number of errors displayed |
| --FILL=*opcode* | Fill unused program locations with this hexadecimal code |
| --GETOPTION=*app,file* | Get the command line options for the named application |
| --HELP<=*option*> | Display the compiler's command line options |
| --IDE=*ide* | Configure the compiler for use by the named IDE |
| --LANG=*language* | Specify language for compiler messages |
| --MEMMAP=*file* | Display memory summary information for the map file |
| --MSGFORMAT<=*format*> | Format general message strings to the given style |
| --MSGDISABLE<=*numbers*> | Disable these warning or advisory messages |
| --NODEL | Do not remove temporary files generated by the compiler |
| --NOEXEC | Go through the motions of compiling without actually compiling |
| --OPT<=*type*> | Enable general compiler optimizations |
| --OUTDIR | Specify output files directory |
| --OUTPUT=*type* | Generate output file type |
| --PRE | Produce preprocessed source files |
| --PROTO | Generate function prototype information |
| --RAM=lo-hi<,lo-hi,...> | Specify and/or reserve RAM ranges |
| --ROM=lo-hi<,lo-hi,...> | Specify and/or reserve ROM ranges |
| --RUNTIME=*type* | Configure the C runtime libraries to the specified type |
| --SCANDEP | Generate file dependency ".DEP files" |
| --SERIAL=*code@address* | Store this hexadecimal code at an address in program memory |
| --SETOPTION=*app,file* | Set the command line options for the named application |
| --SETUP=argument | Setup the product |
| --STRICT | Enable strict ANSI keyword conformance |
| --SUMMARY=*type* | Selects the type of memory summary output |
| --TIME | Show execution time in each stage of build process |
| --VER | Display the compiler's version number |
| --WARN=*level* | Set the compiler's warning level |
| *continued...* | |

PICC Command-line Options

| Option | Meaning |
|---|---|
| --WARNFORMAT=*format* | Format warning message strings to given style |

All single letter options are identified by a leading *dash* character, "-", e.g. -C. Some single letter options specify an additional data field which follows the option name immediately and without any whitespace, e.g. -Ddebug.

Multi-letter, or word, options have two leading *dash* characters, e.g. --ASMLIST. (Because of the double *dash*, you can determine that the option --ASMLIST, for example, is not a -A option followed by the argument SMLIST.) Some of these options define suboptions which typically appear as a *comma*-separated list following an *equal* character, =, e.g. --OUTPUT=intel,cof. The exact format of the options varies and are described in detail in the following sections.

Some commonly used suboptions include default, which represent the default specification that would be used if this option was absent altogether; all, which indicates that all the available suboptions should be enabled as if they had each been listed; and none, which indicates that all suboptions should be disabled. Some suboptions may be prefixed with a plus character, +, to indicate that they are in addition to the other suboptions present, or a minus character "-", to indicate that they should be excluded. In the following sections, *angle brackets,* < >, are used to indicate optional parts of the command.

### 2.4.1 -C: Compile to Object File

The -C option is used to halt compilation after generating a relocatable object file. This option is frequently used when compiling multiple source files using a "make" utility. If multiple source files are specified to the compiler each will be compiled to a separate .obj file. The object files will be placed in the directory in which PICC was invoked, to handle situations where source files are located in read-only directories. To compile three source files main.c, module1.c and asmcode.as to object files you could use a command similar to:

```
PICC --CHIP=16F877A -C main.c module1.c asmcode.as
```

The compiler will produce three object files main.obj, module1.obj and asmcode.obj which could then be linked to produce an *Intel* HEX file using the command:

```
PICC --CHIP=16F877A main.obj module1.obj asmcode.obj
```

### 2.4.2 -D*macro*: Define Macro

The -D option is used to define a preprocessor macro on the command line, exactly as if it had been defined using a #define directive in the source code. This option may take one of two forms, -Dmacro which is equivalent to:

```
#define  macro  1
```

placed at the top of each module compiled using this option, or -Dmacro=*text* which is equivalent to:

```
#define  macro  text
```

where *text* is the textual substitution required. Thus, the command:

```
PICC --CHIP=16F877A -Ddebug -Dbuffers=10 test.c
```

will compile test.c with macros defined exactly as if the C source code had included the directives:

```
#define  debug    1
#define  buffers 10
```

### 2.4.3 -E*file*: Redirect Compiler Errors to a File

This option has two purposes. The first is to change the format of displayed messages. The second is to optionally allow messages to be directed to a file as some editors do not allow the standard command line redirection facilities to be used when invoking the compiler.

The gernal form of messages produced with the -E option in force is:

```
filename  line_number:  (message number)  message string  (message type)
```

If a filename is specified immediately after -E, it is treated as the name of a file to which all messages (errors, warnings etc) will be printed. For example, to compile x.c and redirect all errors to x.err, use the command:

```
PICC --CHIP=16F877A -Ex.err x.c
```

The -E option also allows errors to be appended to an existing file by specifying an *addition* character, +, at the start of the error filename, for example:

```
PICC --CHIP=16F877A -E+x.err y.c
```

If you wish to compile several files and combine all of the errors generated into a single text file, use the -E option to create the file then use -E+ when compiling all the other source files. For example, to compile a number of files with all errors combined into a file called project.err, you could use the -E option as follows:

```
PICC --CHIP=16F877A -Eproject.err -O -C main.c
PICC --CHIP=16F877A -E+project.err -O -C part1.c
PICC --CHIP=16F877A -E+project.err -C asmcode.as
```

Section 2.4.27.1 has more information regarding this option as well as an overview of the messaging system and other related driver options.

### 2.4.4 -Gfile: Generate Source-level Symbol File

The -G option generates a *source-level symbol file* (i.e. a file which allows tools to determine which line of source code is associated with machine code instructions, and determine which source-level variable names correspond with areas of memory, etc.) for use with supported debuggers and simulators such as HI-TIDE™ and MPLAB®. If no filename is given, the symbol file will have the same base name as the project name, and an extension of .sym. For example the option -Gtest.sym generates a symbol file called test.sym. Symbol files generated using the -G option include source-level information for use with source-level debuggers.

Note that all source files for which source-level debugging is required should be compiled with the -G option. The option is also required at the link stage, if this is performed separately. For example:

```
PICC --CHIP=16F877A -G -C test.c modules1.c
PICC --CHIP=16F877A -Gtest.sym test.p1 module1.p1
```

The --IDE option, see Section 2.4.32 will typically enable the -G option.

### 2.4.5 -Ipath: Include Search Path

Use -I to specify an additional directory to use when searching for header files which have been included using the #include directive. The -I option can be used more than once if multiple directories are to be searched.

The default include directory containing all standard header files are always searched even if no -I option is present. The default search path is searched after any user-specified directories have been searched. For example:

```
PICC --CHIP=16F877A -C -Ic:\include -Id:\myapp\include test.c
```

will search the directories c:\include and d:\myapp\include for any header files included into the source code, then search the default include directory (the include directory where the compiler was installed).

This option has no effect for files that are included into assembly source using the INCLUDE directive. See Section 4.3.9.3.

### 2.4.6 -L*library*: Scan Library

The -L option is used to specify additional libraries which are to be scanned by the linker. Libraries specified using the -L option are scanned before the standard C library, allowing additional versions of standard library functions to be accessed.

The argument to -L is a library keyword to which the prefix pic; numbers representing the processor range, number of ROM pages and the number of RAM banks; and the suffix .lib are added. Thus the option -LL when compiling for a 16F877 will, for example, scan the library pic42c-l.lib and the option -Lxx will scan a library called pic42c-xx.lib. All libraries must be located in the LIB subdirectory of the compiler installation directory. As indicated, the argument to the -L option is *not* a complete library filename.

If you wish the linker to scan libraries whose names do not follow the above naming convention or whose locations are not in the LIB subdirectory, simply include the libraries' names on the command line along with your source files. Alternatively, the linker may be invoked directly allowing the user to manually specify all the libraries to be scanned.

### 2.4.7 -L-*option*: Adjust Linker Options Directly

The -L driver option can also be used to specify an option which will be passed directly to the linker. If -L is followed immediately by text starting with a *dash* character "-", the text will be passed directly to the linker without being interpreted by PICC. For example, if the option -L-FOO is specified, the -FOO option will be passed on to the linker. The linker will then process this option, when, and if, it is invoked, and perform the appropriate function, or issue an error if the option is invalid.

Take care with command-line options. The linker cannot interpret driver options; similarly the command-line driver cannot interpret linker options. In most situations, it is always the command-line driver, PICC, that is being executed. If you need to add alternate settings in the linker tab in an MPLAB Build options... dialogue, these are the *driver* options (not linker options), but which are used by the driver to generate the appropriate linker options during the linking process.

The -L option is especially useful when linking code which contains non-standard program sections (or psects), as may be the case if the program contains C code which makes use of the #pragma psect directive or assembly code which contains user-defined psects. See Section 3.12.3.5 for more information. Without this -L option, it would be necessary to invoke the linker manually to allow the linker options to be adjusted.

One commonly used linker option is -N, which sorts the symbol table in the map file by address, rather than by name. This would be passed to PICC as the option -L-N.

This option can also be used to replace default linker options: If the string starting from the first character after the -L up to the first = character matches first part of a default linker option, then that default linker option is replaced by the option specified by the -L.

TUTORIAL

**REPLACING DEFAULT LINKER OPTIONS** In a particular project, the psect entry is used, but the programmer needs to ensure that this psect is positioned above the address 800h. This can be achieved by adjusting the default linker option that positions this psect. First, a map file is generated to determine how this psect is normally allocated memory. The Linker command line: in the map file indicates that this psect is normally linked using the linker option:

```
-pentry=CODE
```

Which places entry anywhere in the memory defined by the CODE class. The programmer then re-links the project, but now using the driver option:

```
-L-pentry=CODE+800h
```

to ensure that the psect is placed above 800h. Another map file is generated and the Linker command line: section is checked to ensure that the option was recieved and executed by the linker. Next, the address of the psect entry is noted in the psect lists that appear later in the map file. See Section 5.11 for more information on the contents of the map file.

If there are no characters following the first = character in the -L option, then any matching default linker option will be deleted. For example: -L-pfirst= will remove any default linker option that begins with the string -pfirst=. No warning is generated if such a default linker option cannot be found.

TUTORIAL

**ADDING AND DELETING DEFAULT LINKER OPTIONS** The default linker options for for a project links several psects in the following fashion.

```
-pone=600h,two,three
```

which links one at 600h, then follows this with two, then three. It has been decided