

CROSSWARE® C680X0NT

The Crossware C680X0NT is an ANSI standard C compiler that generates code for the M68000, M68010, M68020 microprocessor and M68881 maths coprocessor and Motorola microcontrollers based upon the CPU32 and CPU32+ cores. Supported chips include the M68000, M68008, M68302, M68306, M68307, M68322, M68328, M68010, M68020, M68330, M68332, M68340, M68349 and M68360. It comes complete with the Crossware Embedded Development Studio and runs under Windows 9x, Windows NT 4.0, Windows 2000 and Windows XP.

680X0 ANSI C Compiler for Windows

HIGHLIGHTS

- Optimising ANSI C compiler with extensions for embedded development
- Source level debug output in IEEE695 format
- Pre-written library routines including 32 bit and 64 bit floating point arithmetic
- Wide range of output file formats
- Integrated Motorola compatible assembler
- Highly user friendly Embedded Development Studio integrated development environment (see separate data sheet)
- Data output for Embedded Development Studio browser

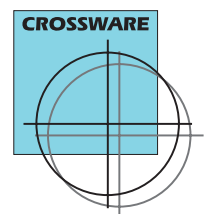
OVERVIEW

The package includes the following tools and utilities:

- optimising ANSI C compiler
- C support libraries
- relocatable macro cross assembler
- relocating linker
- make utility
- library manager
- Embedded Development Studio which includes
 - integrated project creation and maintenance utility
 - integrated editor
 - integrated terminal emulator
 - integrated C symbol browser
 - on-line books with full text search

(The Embedded Development Studio and the relocatable macro cross assembler are described in detail in separate data sheets)

All of these tools and utilities are Win32 applications built to make the most of your 32 bit computing environment. All tools can be used outside of the Embedded Development Studio if required using conventional command-line instructions.



*Crossware Products
Old Post House
Silver Street
Litlington
Royston
Herts
SG8 0QE
United Kingdom*

*Telephone
+ 44 (0) 1763 853500*

*Facsimile
+ 44 (0) 1763 853330*

*Web
<http://www.crossware.com>*

*E-mail
sales@crossware.com*

REF: C60X0NT/0302

TECHNICAL DETAILS

OPTIMISING C COMPILER

Language Definition

The compiler conforms to the ANSI C specification and in addition, provides a number of enhancements including:

- variables can be any length with all characters significant
- the C++ commenting convention "//" is supported
- the `_interrupt` keyword declares a C function as an interrupt routine

The support libraries are a subset of the ANSI standard libraries. The supported functions are listed below.

Data Sizes

The compiler uses the following sizes for the various C data types:

char and unsigned char	:1 byte
short int and unsigned short int	:2 bytes
int and unsigned int	:4 bytes;
long and unsigned long	:4 bytes
float	:4bytes (32 bits)
double	:8 bytes (64 bits)
long double	:8 bytes (64 bits)
enum	:up to 4 bytes (minimum size to accommodate members)
bit fields	:up to 32 bits

In-Line Assembler

Assembler code can be included in your C source files both within and outside of C functions. In-line assembler code is passed unmodified to the relocatable assembler and so any assembler constructs including macros, segment directives and conditional assembly directives can be included.

Optimizations

Optimizations include:

- constant folding
- dead code elimination
- strength reduction
- algebraic simplification
- jump/branch optimization
- suppression of integral promotion
- global register allocation

Code and data location

Compiler generated code and data are automatically located in appropriate memory segments. Linker options allow these segments to be located at user defined memory locations. String constants and objects declared as `const` are located in code space. Initialized and uninitialized data are located in separate segments and initialisation of these segments is automatically carried out at run time.

RELOCATING LINKER

The linker combines object modules created with the compiler and/or the assembler to create the final code that will run on your target system. It carries out the following functions:

- scans each module to collect segment and variable information
- arranges and positions segments at appropriate memory locations to suit the memory organisation of the target system and any specific location information supplied by the user
- finalises the values of all variables and calculates the results of any incomplete expressions
- extracts and relocates the code from each module to produce the final target program

The target program can be produced in a number of different formats including Motorola S2 and S3 records, HP/Microtec IEEE695 format or as a binary rom image. (A utility is provided for users who wish to divide the binary rom image into separate files suitable for 8 bit wide memory chips).

An optional link map will show the final location and sizes of all segments and the values of all global variables.

DEBUG OUTPUT

The compiler and assembler can both optionally generate full source level debug information. The linker updates these debug records to take account of the final location of the target program and outputs them to the target program file in IEEE695 format. Debuggers and in-circuit emulators that support this popular format can then use it to facilitate full source level debugging.

LIBRARY MANAGER

Instead of being used to create a final target program, the object modules produced by the compiler and assembler can be integrated into a library. The library manager performs the tasks of:

- combining object modules into a library
- adding modules to an existing library
- removing or extracting modules from an existing library
- listing the contents of a library

MAKE UTILITY

The MAKE utility simplifies the task of keeping object files, libraries and target programs up-to-date. It detects if any source or dependency files have changed since the last build and runs the appropriate tools (compiler, assembler, linker or library manager) to rebuild out-of-date files. It supports many advanced features including macros, inference rules, conditional inclusion and other preprocessing directives and in-line files with automatic temporary file creation.

Although the Embedded Development Studio uses its own integrated routines to keep projects up-to-date, this stand-alone MAKE utility can be used to build projects from the command-line or from within another application. The Embedded Development Studio will automatically create a makefile which is fully compatible with this stand-alone MAKE utility.

HOST SYSTEM REQUIREMENTS

IBM compatible PC with an Intel Pentium or above running Windows 9x, Windows NT4.0, Windows 2000 and Windows XP.

Support Library Routines:

<code>abs()</code>	<code>exit()</code>	<code>gets()</code>	<code>log10l()*</code>	<code>sinl()</code>	<code>strpbrk()</code>
<code>acos()*</code>	<code>exp()</code>	<code>getvect()</code>	<code>malloc()</code>	<code>sinh()</code>	<code>strchr()</code>
<code>acosl()*</code>	<code>expl()</code>	<code>isalnum()</code>	<code>memchr()</code>	<code>sinhl()</code>	<code>strspn()</code>
<code>asin()*</code>	<code>fabs()</code>	<code>isalpha()</code>	<code>memcmp()</code>	<code>sprintf()</code>	<code>strstr()</code>
<code>asini()*</code>	<code>fabsl()</code>	<code>isascii()</code>	<code>memcpy()</code>	<code>sqrt()</code>	<code>tan()</code>
<code>atan()*</code>	<code>_fcvt()</code>	<code>iscntrl()</code>	<code>memmove()</code>	<code>sqrtl()</code>	<code>tanl()</code>
<code>atanl()*</code>	<code>ferroe()</code>	<code>isdigit()</code>	<code>memset()</code>	<code>srand()</code>	<code>tanh()</code>
<code>atoi()</code>	<code>fgetc()</code>	<code>isgraph()</code>	<code>pow()</code>	<code>sscanf()</code>	<code>tanhl()*</code>
<code>atol()</code>	<code>fgets()</code>	<code>islower()</code>	<code>powl()</code>	<code>strcat()</code>	<code>time()</code>
<code>atoff()</code>	<code>fileno()</code>	<code>isprint()</code>	<code>printf()</code>	<code>scanf()</code>	<code>toascii()</code>
<code>atolf()</code>	<code>floor()</code>	<code>ispunct()</code>	<code>putc()</code>	<code>strchr()</code>	<code>tolower()</code>
<code>clearerr()</code>	<code>floorl()</code>	<code>isspace()</code>	<code>putchar()</code>	<code>strcmp()</code>	<code>toupper()</code>
<code>ceil()</code>	<code>fprintf()</code>	<code>isupper()</code>	<code>puts()</code>	<code>strcpy()</code>	<code>ungetc()</code>
<code>ceilf()</code>	<code>free()</code>	<code>isxdigit()</code>	<code>rand()</code>	<code>strcspn()</code>	<code>ungetch()</code>
<code>cos()</code>	<code>fscanf()</code>	<code>labs()</code>	<code>sbrk()</code>	<code>strlen()</code>	
<code>cosl()*</code>	<code>getc()</code>	<code>log()</code>	<code>scanf()</code>	<code>strncat()</code>	
<code>cosh()*</code>	<code>getchar()</code>	<code>logl()</code>	<code>setvect()</code>	<code>strncpy()</code>	
<code>coshl()</code>	<code>getche()</code>	<code>log10()*</code>	<code>sin()</code>		

* Support with 68881 co-processor only.