

HIGHLIGHTS

- Interoperability with Motorola GNU
- Integrated tools delivering a rapid edit-compile-debug process
- Customize to your own environment
- Easy project setup and management
- Graphical symbol browsing
- C/C++/EC++ compiler
- Powerful language extensions
- Comprehensive optimization techniques
- Basic and advanced debugging
- Performance analysis in debugger
- RTOS aware

MAXIMIZE APPLICATION PERFORMANCE

Application size and speed are two of the most important aspects of embedded application development. The optimizing capabilities of the TASKING DSP56xxx C/C++/EC++ compiler tools and the program performance analysis functionality of the CrossView Pro Debugger impel you to produce the most efficient code.

The DSP56xxx Software Development Toolset consists of a C/C++/EC++ compiler, optimizing assembler, linker/locator, libraries, CrossView Pro debugger, and EDE (Embedded Development Environment).

COMPATIBILITY AND INTEGRATION WITH MOTOROLA GNU

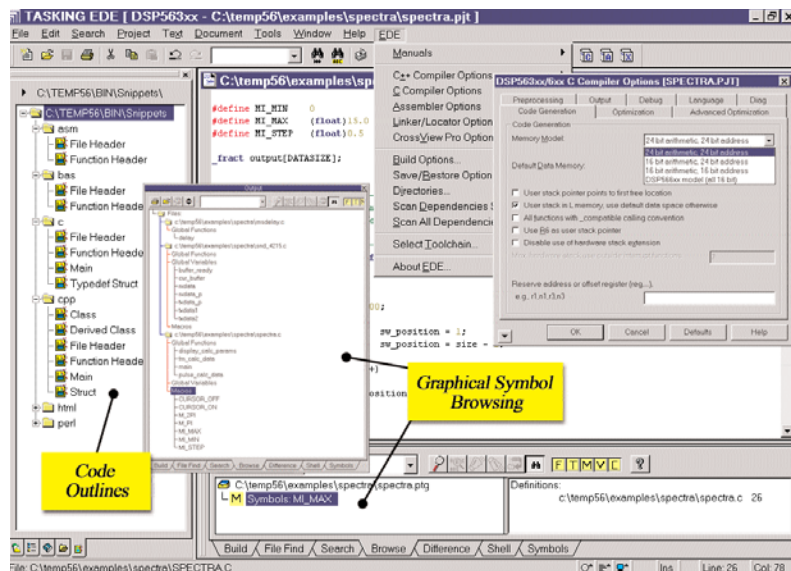
Our DSP56xxx Software Development Toolset supports COFF (.CLD) compatibility and the GNU calling convention, enabling you to generate code for the Motorola assembler and linker with COFF compliant debug information.

Support of the Motorola toolchain is fully integrated into our EDE so you can easily select the entire TASKING toolchain or the TASKING and Motorola toolchain. To ease migration from the Motorola GNU tools to the TASKING software development tools, a migration guide is available. This guide provides helpful hints to keep in mind during the migration, as well as how to obtain the highest performance from your application.

EMBEDDED DEVELOPMENT ENVIRONMENT

TASKING's EDE is an integrated environment for tools which create, edit, compile, and debug your application. With EDE you can create and maintain projects easily, so your application is always up-to-date. All aspects of a project are saved in the project file: the source files that make up your application, the compiler and debugger options, the tool directories, and the options that describe the building process. File dependencies as well as the sequence of operations required to build your application are handled automatically.

Code generation is streamlined by EDE. Graphical symbol browsing enables you to obtain a quick review of any source code by providing an overview of cross references in your application (such as defined global variables, functions, and enumerations types). Repetitive tasks are automated with a "type ahead" feature that automatically completes words and provides function parameter help. Graphical symbols distinguish portions of code, and "go to" buttons take you to their definition.



C/C++/EC++ COMPILER

- Complete compatibility with Motorola GNU compiler
- Code size optimized an average of 40% better than GNU
- ANSI C and C++ compliant
- Embedded C++ (EC++) support
- Static, reentrant, and mixed memory models
- Storage specifiers for X, Y, L, and P memory
- Optimizing assembler
- Multiple locator output formats

EDE also includes the following capabilities:

- Language sensitive editor
- Convenient code reuse
- Predefined and custom "code outlines"
- Automated make facility
- Librarian
- Built-in grep and file difference
- Tool option selection
- Object code reporter
- HTML language and web browser support

EDE integrates error message output from the TASKING tools into your editing environment. EDE interprets the error messages generated by the tools and indicates where the errors are, allowing you to fix them quickly.

Code reuse is supported by the ability to click and drag commonly used code into a folder or file view. These "snippets" enable your entire team to define project code outlines and/or reuse code with other projects.

Using Version Control is easy. EDE includes an interface for checking in your changes, checking a file for review or locking a file you plan to change. EDE also provides an interface to standard version control packages.

C COMPILER

The ANSI C DSP56xxx compilers are designed and built specially for each member of the Motorola DSP family. Use of the fractional datatype and memory space qualifiers allows the compiler to optimize loops extensively and exploit the parallel execution capability of the DSP. The compiler supports the 16- and 24-bit mode of the DSP563xx. These features help you reach unsurpassed code density and make specific DSP56xxx capabilities directly accessible from C.

The compiler optimizations developed by TASKING generate code size that is an average of 40% (or more) smaller and faster than GNU. Not only does the compiler generate more efficient code, but the assembler performs additional optimizations to minimize code size and execution time.

You can also create bootable EPROM images. Utilities included with the compiler extract the data directly from the .abs file and create images for programming a single 8-bit or parallel 24-bit EPROM's.

C++/EC++

The C++ compiler delivers the power of object-oriented design and coding to your DSP application. Full support of templates, dynamic casts, runtime type identification, and exception handling is provided. EC++ (Embedded C++) support is available to reduce the high application overhead often introduced by C++. The evolving EC++ standard addresses this issue by omitting a number of features that are not essential for most embedded applications. An EC++ option helps you conform to the EC++ standard.

CLAS Compatibility

The TASKING tool suite supports several features which provide compiler interoperability with the Motorola CLAS tool set. By default, the compiler uses a different calling convention than the CLAS compiler. This calling convention guarantees better use of registers, and therefore, less function call overhead. A special keyword (*_compatible*) allows you to prototype individual functions, thus allowing the compiler to apply the more optimal scheme when possible and use the compatible CLAS protocol when necessary.

Compiler Optimization

State-of-the-art optimization techniques are applied to reduce the size of generated code and/or decrease execution speed. The following are a sample of the optimization techniques used:

- Compiler generated DO and REP loops
- Effective use of DO-FOREVER and BRKcc
- MAC instruction in computational expressions
- *_near*, *_internal* and *_external* memory qualifiers
- *_fract* data type for fixed point arithmetic
- Complex data type
- Built-in support for overflow/saturation
- Circular buffer declarations with *_circ* type qualifier
- Cache handling pragma's
- Subexpression elimination
- Loop recognition
- Variable usage analysis
- Register contents tracking
- Automatic stack overflow checking

In addition to the extensive optimizations, various other features help you to optimize and tune your code:

In-line expansion of predefined functions, such as: `_abs`, `_asm`, `_rol`, `_ror`, `_stop`, `_cmac`, `_cmul`, `_cadd`, `_cdiv`, `_nop`, `_swi`, `_wait`, `_round`, `_pdiv`, `_fsqrt`. In-line functions do not incur the typical function call overhead: they translate directly and have no direct equivalent in C.

Adjustable code generation with `#pragma`'s. To control the individual compiler optimizations, allocation of character arrays ('packed' or a character per memory word to optimize for data size or code speed respectively) and `pragma`'s for cache handling.

Circular buffer type modifier for efficient filter implementation.

Floating point libraries with limited exception trapping to reduce runtime argument checking.

Memory models to control the allocation of parameters and automatics to fit the needs of your application. For the DSP5600x, a static model reduces the use of more expensive stack relative addressing modes. The reentrant model provides true stack allocation of objects, and with the mixed model, you can use a mixture of both flavors, tuning your code to use the stack only at those places where you really need it.

Data Types and Sizes

All ANSI C types are supported. Additionally a fixed point data type (`_frac`) has been included.

Data Type	DSP566xx	DSP5600x/3xx
	size in bits	size in bits
(un)signed char	8	8
(un)signed short	16	16
(un)signed int	16	24
(un)signed long	32	48
(long) <code>_frac</code>	16 (32)	24 (48)
pointer	16	16/24
float/double	16+8	24+8
enum	16	24

Libraries

The compilers are delivered with libraries for all the different members of the DSP56xxx families. Each set consists of ANSI C libraries, C++ template libraries, runtime libraries, and fixed and floating point libraries. The types float and double are both implemented as single precision floating point.

ASSEMBLER

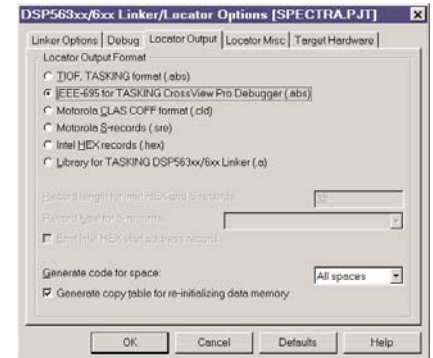
Features:

- Compliant with Motorola's CLAS assembler package
- Supports nested and fragmented sections
- Selects shortest possible branch (forward and backward)
- Accepts same model options as compiler
- Macro preprocessing
- Extended set of controls and pseudo instructions for section handling and list generation

- Built-in functions can retrieve the model, default, and stack memory space during code generation
- Extensive flow analysis
- Parallel instruction execution

To improve code quality, the assembler performs extensive flow analysis to determine how instructions can be rearranged to reduce the code size and increase execution speed. The assembler also checks whether instruction sequences with pipeline hazards occur and attempts to avoid them by reorganizing the code. If no suitable instruction sequences are available, then the assembler inserts a NOP instruction to force deterministic (and expected) behavior of the program.

The assembler has a built-in macro pre-processor which features an include file mechanism, macro definition and expansion, as well as conditional assembly. Macros can be defined and undefined at any place in the source. The pre-processor supports a set of controls which help you to create structured assembly programs.



Move instructions can be executed in parallel with other instructions if no resource conflicts occur. By automatically rearranging instructions such that parallel execution becomes possible, the assembler saves you time while making your code as compact and fast as possible.

For example, the code:

```
move x:Fbuffer_p, r6
gmove #0, b
```

can be replaced by the single instruction:

```
clr b x:Fbuffer_p, r6
```

Linker/Locator

Features:

- Overlaying linker to reduce memory usage
- Locator overlaying
- Partial linking
- Incremental linking
- Linker & locator map file with symbol values and memory assignments
- Flexible locator control language to control memory layout of your application
- Locator control files supplied for most commercial targets
- Automatic inclusion of library modules
- Global type checking
- IEEE-695, Motorola S-records, Intel-hex and CLAS (without symbolic information) locator output formats

CROSSVIEW PRO

- Graphical User Interface to all features
- Multiple DSP debugging
- Mixed-mode source/assembly display
- Multiple window displays
- High speed simulator
- Powerful breakpoint control
- Probe points
- Breakpoint sequencer
- Program performance analysis
- Hardware breakpoints
- Bubble-Spy™ technology for easy and quick inspection of variable contents
- Record and playback debug session
- Register grouping
- Single stepping without stopping
- File system simulation
- RTOS aware debugging

CROSSVIEW PRO DEBUGGER

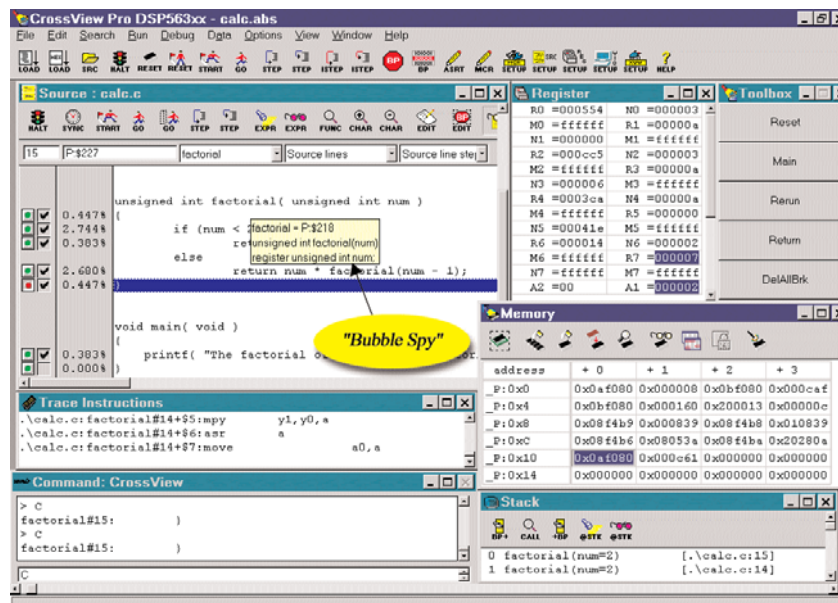
CrossView Pro DSP56xxx interfaces to the Motorola Application Development System (ADS), the Evaluation Modules DSP56xxx EVM, and the Domain Technologies SB-56K. An easy-to-use Graphical User Interface combined with powerful debugging features helps you debug your applications faster. With the Motorola Command Converter connected to a JTAG chain, the CrossView Pro debugger will allow you to debug multiple DSP's of the same family by switching from one processor to another while the others continue to run.

Features:

- Multiple viewing windows (source, register, trace, memory, stack, data, command) display the type and level of information needed at any point during the debugging session
 - Display your program as C source text, assembly source, or a mixture of both
 - Single-step through C or assembly source
 - Set, clear, enable, and disable breakpoints
 - Set software or hardware breakpoints based on code, data access, instruction/cycle count, stack level, or complex conditions
 - Probe Points can be used to set breakpoints based upon a sequence of events and controlling I/O simulation
 - Breakpoint Sequencer simplifies creating complex sequences by providing Boolean functions and supporting nested sequences
 - Code, data, complex, stack, and hardware breakpoints
 - Start and stop recording debug session at any time
 - Playback debug session to automate debugging or test application
- Double-click to modify values or expand and contract complex data structures
 - Use assertions for hard-to-find errors
 - Watch or show data (global or local)
 - Immediately view the values of a variable or function by simply moving your mouse (Bubble-Spy)
 - Edit, display, and group registers
 - Display multiple windows of different register groups
 - Examine the contents of the OnCE/JTAG port trace buffer
 - Observe the state of current stack frame, including function parameters
 - Monitor and edit the current value of memory locations
 - Open multiple memory windows for different ranges, and display a different format (ASCII or numeric) for each window
 - High speed instruction set simulator includes instruction trace, profiling, code, and data coverage
 - File system simulation enables the use of regular I/O functions such as fopen() and fprintf() to use files on the host system, with both keyboard and screen I/O redirected to CrossView Pro windows

Open Architecture

Public Interfaces such as the Kernel Debug Interface (KDI) and Generic Debug Instrument Interface (GDI) provide third party vendors easy access to the CrossView Pro framework. The KDI can also be used to provide RTOS aware debugging for "in-house" kernels so you can now easily debug your RTOS-based code.



BREAKPOINTS AND PROGRAM PERFORMANCE ANALYSIS

Powerful Breakpoints

Setting breakpoints is the feature most often used in a debugging session. The CrossView Pro debugger provides you with a variety of breakpoint capabilities enabling you to resolve a simple or difficult problem quickly.

- **Code breakpoints** halt the program at a particular statement or instruction so the values of variables can be observed.
- **Data breakpoints** let you determine when memory addresses are read and/or written to, and are useful for tracking the possible misuse of pointers, global variables, and memory mapped I/O ports.
- **Complex breakpoints**, after reaching an address, check either a register or memory location for specified values before taking the breakpoint.
- **Time breakpoints** halt the program after a specified count of cycles or instructions have been executed.
- **Breakpoint sequences** of the above, will halt the program only when all breakpoints in the sequence have occurred.
- **On-chip hardware breakpoints** enable you to set breakpoints on any type of memory access or memory range after a number of accesses and to place breakpoints in ROM.
- **Stack breakpoints** can be set at either function entry or exit

Program Performance Analysis

The CrossView Pro debugger provides several performance analysis capabilities to help you further optimize your application as well as shorten your debugging session.

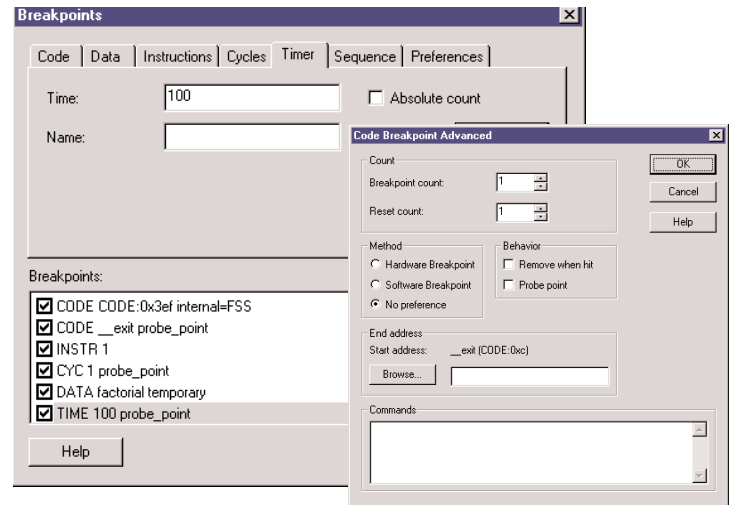
These capabilities include:

- **Profiling**
- **Code Coverage**
- **Cycle Counting**
- **Programmable Graphical Data Analysis**

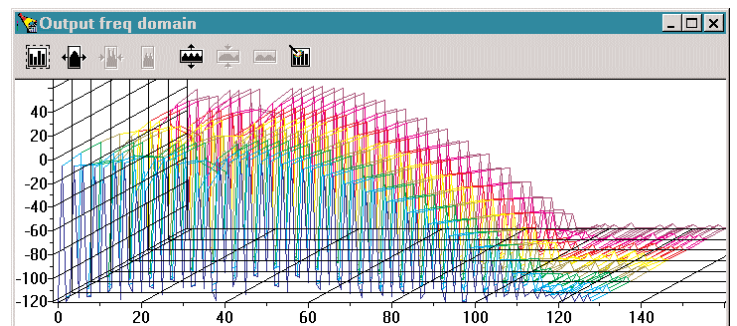
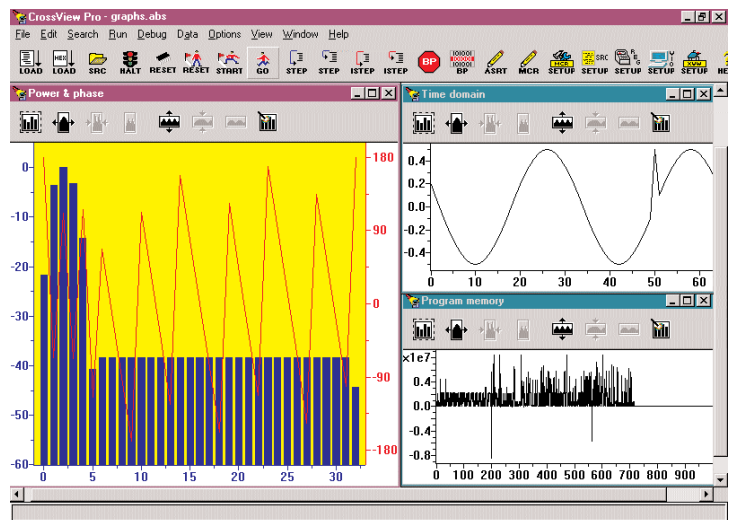
Profiling helps you identify bottlenecks in your code by enabling you to perform timing analysis by providing timing information about a particular function or set of functions. You can see how often a function is called and how much time is spent in each function.

Code coverage tracks all memory access (memory read, memory write, instruction fetch) so you can determine if there are any areas of unexecuted code. You can also use one of the timers for cycle counting on the real hardware. Call counts, timing per line, timing per assembly instruction, and coverage are available in the simulator.

The Programmable Graphical Data Analysis feature reduces large sets of data into meaningful visual diagrams to enable quick detection of gross errors in the input signal. The CrossView Pro debugger analyzes the data, according to pre-defined or user-defined specifications, and then displays the data the way you need it. You can also view the same data in several ways at the same time (for example, in the time and frequency domain).



Five pre-defined analysis types are now available: x-t plotting, x-y plotting, FFT power spectrum, FFT waterfall, and Eye diagram.



AVAILABILITY

The DSP56xxx solution is available for PC/Windows and Sun/Solaris.

SUPPORTED DERIVATIVES

DSP5600x	DSP563xx	DSP566xx
DSP5600xEVM	DSP563xxEVM	DSP566xxEVM
DSP5600xADS	DSP563xxADS	DSP566xxADS

Our detailed list of supported derivatives is updated continuously, so please check our web site for the specific derivatives supported or contact your local sales representative.

PRODUCT PACKAGING & ORDERING CODES

Each TASKING product comes with full documentation. The documentation is available on-line as well and provides full-text search capabilities for quick and easy lookup of topics.

Product Code Package contents

Combination Package for all DSP5600x, 3xx & 6xx

07-200-039-024 EDE, C/C++ Compiler, Assembler/Linker, CrossView Pro Simulator

07-200-039-049 CrossView Pro EVM/ADS Debugger

Demonstration versions of the DSP56xxx tools are downloadable from our web site at:
www.tasking.com/dsp56xxx

Developers forum: www.tasking.com/forum

ALTIUM OFFICES WORLDWIDE

North America

Altium Inc.
3207 Grey Hawk Court
Suite 100
Carlsbad, CA 92010
Ph: +1 760-231-0760
Fax: +1 760-231-0761
sales.na@altium.com
support.na@altium.com

Germany

Altium Europe GmbH
Philipp-Reis-Straße 3
76137 Karlsruhe
Ph: +49 721 8244 300
Fax: +49 721 8244 320
sales.de@altium.com
support.eu@altium.com

China

Altium Information Technology
(Shanghai) Co., Ltd
9C, East Hope Plaza
No.1777 Century Avenue
Shanghai 200122
Ph: +86 21 6182 3900
Fax: +86 21 6876 4015
sales.cn@altium.com
support.cn@altium.com

Japan

Altium Japan K.K.
Nomura Fudosan Yotsuya Bldg 7F
2-12-1 Yotsuya
Shinjuku-ku, Tokyo
160-0004
Ph: +81 3 6672 6155
Fax: +81 3 6672 6159
sales.japan@altium.com
support.japan@altium.com

The Netherlands

Altium Technology Centre
Altium BV
Saturnus 2
3824 ME Amersfoort
Ph: +31 33 4558584
Fax: +31 33 4550033
tasking@altium.com