VX-toolset for M16C and R8C/Tiny

TASKING™

HIGHLIGHTS

- Incorporates next-generation Viper compiler technology
- State-of-the-art C compiler
  - Fast and compact
- Support for MISRA-C:2004
- Run-time error checking facilities in the compiler
- Profiling using code instrumentation
- CrossView Pro debugger
  - M16C instruction set simulator
  - TASKING ROM monitor debugger
  - Renesas PC4701 Emulator support
- ELF/Dwarf support
- Available for:
  - PC/Windows
  - SUN/Solaris

M16C PLATFORM

R8C Tiny

RENESAS

THE TASKING C/C++ COMPILER AND DEBUGGER VX-TOOLSET FOR M16C AND R8C/TINY

The M16C family, including R8C/Tiny, from Renesas is one of the most versatile 16-bit microcontrollers on the market. The architecture provides developers of embedded software with features such as: high-speed processing, ultra-low power consumption, and EMI/EMS noise immunity, combined with high peripheral functionality and enhanced I/O capabilities.

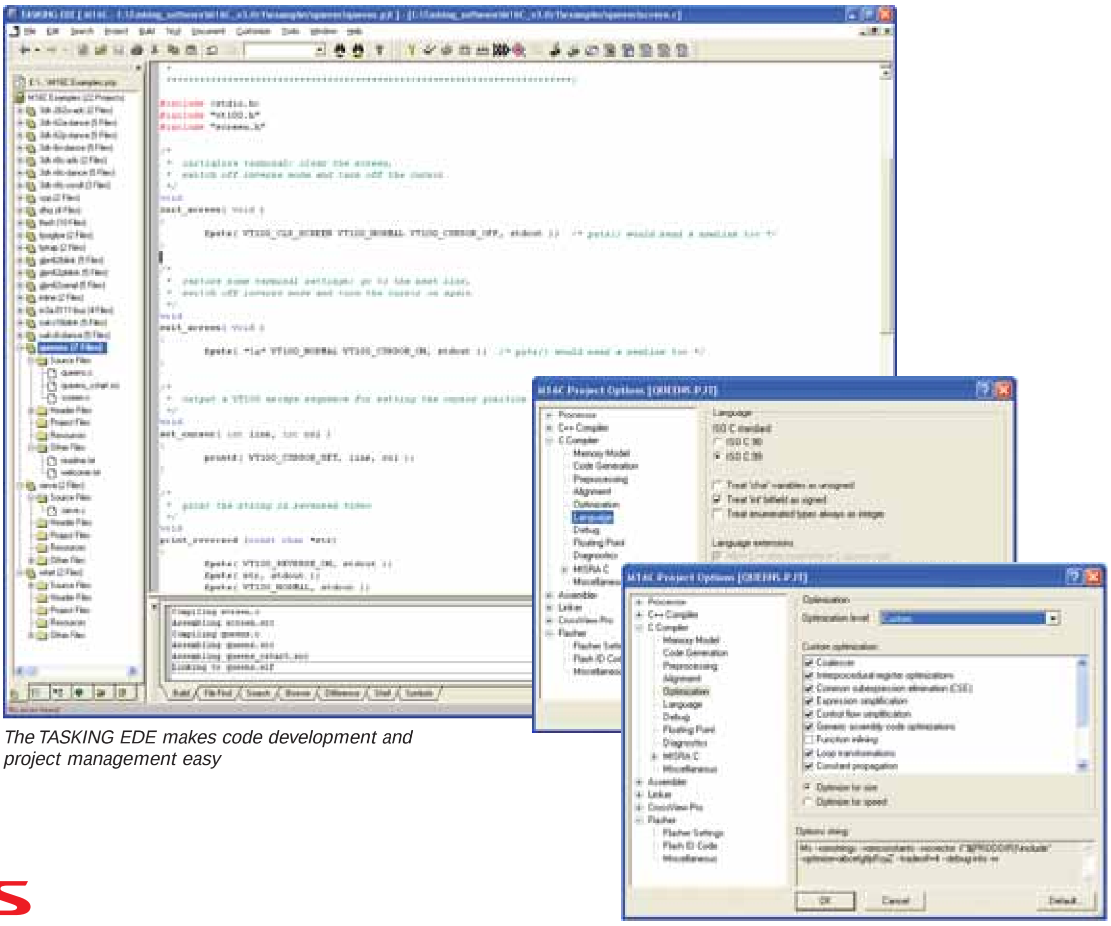
The TASKING VX-toolset for M16C provides developers with the power of Altium's in-house, next-generation compiler technology framework, codenamed Viper. The toolset enables you to take full advantage of the high-performance M16C architecture, generating code with the high level of execution speed and code density needed for automotive, industrial and consumer applications.

EMBEDDED DEVELOPMENT ENVIRONMENT

With the TASKING M16C Embedded Development Environment (EDE), you can create and maintain projects easily. All project-related aspects, such as the application source files, the tool options (compiler, assembler, linker/locator, CrossView Pro debugger), file management, and the options of the build process, are managed from one central point. File dependencies, as well as the sequence of operations required to build the application, are handled automatically.

The M16C EDE offers many productive features for application and code development, including:

- **Explorer-like** treeview control allowing simplified configuration of the TASKING tools and the M16C target processor for the experienced, as well as the novice, user
- **Menu structure** tuned according to the development work flow, offering an intuitive project management setup
- **Easy selection** of target processor and new edit controls for project settings and configurations
- **Project Spaces** that enable you to group multiple projects in one view, offering improved project management for more complex developments
- **CodeSense** that provides advanced coding assistance and offers rich type-ahead features, which help you in selecting the next expected function parameter or available structure members. When positioning your mouse pointer over a function name, the function prototype will be displayed
- **CodeFolio** which enables easy insertion of template code, adding to coding efficiency and consistency. It allows macro expansion and prompted input as you insert the code



The TASKING EDE makes code development and project management easy

C++/EC++ COMPILER

Altium is one of the few vendors to offer object-oriented design and coding possibilities for the M16C family through an ISO C++ compliant compiler. The advantages of C++ can be incorporated into an existing C application one module at a time, providing a graceful migration from C to C++. Inheritance reduces the number of places where software behavior is defined and thereby speeds up development. The C++ compiler automatically includes a pre-link phase when templates are used.

Scalable C++

Compatibility with the Embedded C++ (EC++) standard allows selective disabling of C++ features that may not be needed for your embedded application. Code size overhead and run-time inefficiency can be managed by selecting full, or partial, compliance to the EC++ standard.

C COMPILER

Based upon Altium's latest C compiler technologies, the TASKING M16C compiler is easy to use and generates the most optimal code possible, allowing you to take full advantage of the M16C architecture.

TASKING compilers are tested for ISO and ANSI conformity against authoritative validation suites such as Plum Hall and Perennial. In addition, the optimization techniques of the compilers are tested with various large real-world applications, as well as industry benchmark standards such as Nullstone.

Fast and compact code

The VX-toolset for M16C, based on Altium's Viper compiler technology, generates code which is small and has a fast execution speed in its default configuration. Depending on the specific requirements of your M16C application, optimizations can be tweaked for smaller code size or higher execution speed.

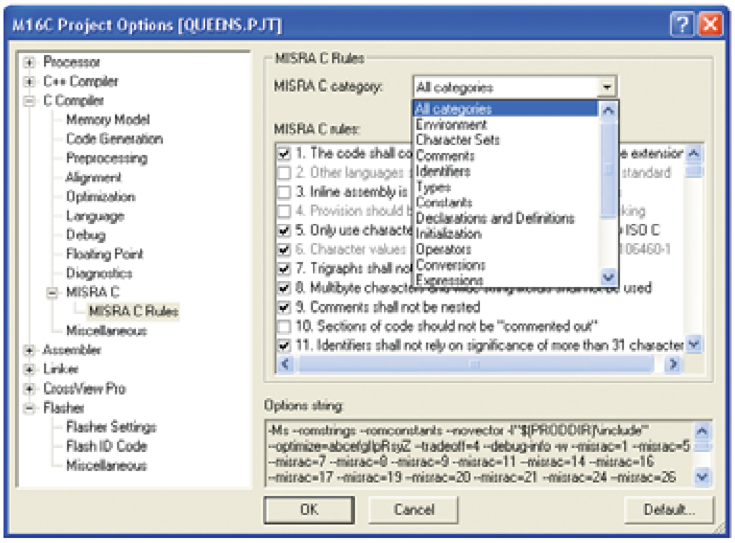
Compiler optimizations include:

- **Partial Redundancy and Elimination (PRE), which detects and eliminates repeating (sub-) expressions**
- **Value rangetracking**
- **Control-flow and code reduction optimizations that remove dead code and perform transformations in order to minimize jumps**
- **Function inlining, which replaces calls to small functions with inlined copies of the function code**
- **Interprocedural register allocation**
- **Code compaction that identifies identical code sequences, creates a new function from the sequence, and replaces the sequences with calls to the new function**

MISRA C

Based on the "Guidelines for the use of the C language in vehicle based software" published by the Motor Industry Software Reliability Association (MISRA®), Altium was the first to implement the MISRA C concept in a software development environment. Since then, the MISRA C guidelines have been widely adopted by users across automotive, aerospace, industrial and medical industries. MISRA C guides programmers to write more robust C-code by defining selectable C usage restriction rules. Through a system of strict code checking, the use of error-prone C constructs can be prevented.

A predefined configuration for compliance with the MISRA guidelines is available with a single click. It is also possible, using pull-down menus, to enable a custom set of MISRA C rules to suit specific company requirements.



MISRA C category: All categories

MISRA C rules:
- [✓] 1. The code shall co… …e extension
- [ ] 2. Other languages … …standard
- [ ] 3. Inline assembly is …
- [ ] 4. Provision should b… …king
- [✓] 5. Only use characte… …ISO C
- [✓] 6. Character values … …06460-1
- [✓] 7. Trigraphs shall no…
- [✓] 8. Multibyte character… …used
- [✓] 9. Comments shall not be nested
- [ ] 10. Sections of code should not be "commented out"
- [✓] 11. Identifiers shall not rely on significance of more than 31 character

Options string:

```
-Ms --romstrings --romconstants --novector -I"$(PRODDIR)\include"
--optimize=abcefgIlpRsyZ --tradeoff=4 --debug-info -w --misrac=1 --misrac=5
--misrac=7 --misrac=8 --misrac=9 --misrac=11 --misrac=14 --misrac=16
--misrac=17 --misrac=19 --misrac=20 --misrac=21 --misrac=24 --misrac=26
```

Fully-configurable MISRA C code checking

To ensure compliance with the MISRA C rules throughout the entire project, the M16C linker/locator can generate a MISRA C Quality Assurance report. It lists the different modules in the project with the respective MISRA C configurations which were used to compile them.

The M16C toolset supports the original MISRA-C:1998 standard as well as the new MISRA-C:2004 rules set.

Run-time error checking

TASKING's run-time error checking capabilities in the compiler offer a wealth of checks that reveal run-time errors when they first occur. The kind of errors found by run-time error checking are typically hard to find since they manifest themselves through secondary effects or, in the worst case, will not manifest at all prior to your product being shipped. By identifying the source line where the error first occurs, the run-time error checking facilities reduce the time spent in the debugger and increases the quality of your software. You can specify whether the application will terminate or continue when an error is detected. These checks are implemented by generating additional code and/or enabling additional code in the standard C library. Run-time error checking has a nominal effect on code size and execution speed and can be enabled on a module by module basis, making it practical for use in debugging large applications.

Profiling

The compiler is enhanced with a profiler that uses code instrumentation. Profiling can be used to determine which pieces of your code execute slower than expected and which functions contribute most to the overall execution time of a program. A profile can also tell you which functions are called more or less often than expected. The advantage of this new profiling method is that it can give a complete call graph of the application annotated with the time spent in each function and basic block.

Syntax and semantic checks

The compiler offers a vast array of syntax and semantic checks that warn about potential undesirable effects or bugs in your program. Early fixing of source code problems when reported by the compiler generally only takes minutes as compared to hours, or days, when the problem is discovered at run time.

Examples of compile-time checks include:

- **Validating printf and scanf format strings against the type of the actual arguments**
- **Using uninitialized memory locations**
- **Detecting unused variables**
- **Value tracking, which is used to detect errors such as**
  - **array subscript out of bounds**
  - **division by zero**
  - **constant conditions**

M16C architectural support

The TASKING M16C VX-toolset provides full support for all primary M16C series, including pre-prepared definitions, SFR header files, and automated Cstart code generation.

The supported M16C series variations are:

- **M16C/60**
- **M16C/30**
- **M16C/Tiny**
- **M16C/20**
- **M16C/10**
- **R8C/Tiny**

The TASKING M16C VX-toolset offers a wealth of built-in functions. Intrinsic functions appear as normal C functions, but the code generator interprets them and, where possible, generates more efficient code. Several pre-declared functions are available to generate inline assembly code at the location of the intrinsic function call, ensuring fastest execution by avoiding the standard function calling and parameter-passing overhead.

Memory models

To take full advantage of the M16C microcontroller, the M16C VX-toolset consists of three re-entrant memory models.

Memory models available:

- **Small**
- **Medium**
- **Large**

Model	Data
Small	default in 0 - 64 kB
Medium	default in 0 - 64 kB* **Constants default in 0 - 1 MB*
Large	default in 0 - 1 MB

Model	Pointers
Small	0 - 64 kB
Medium	0 - 1 MB, 16 bit arithmetic
Large	0 - 1 MB, 20 bit arithmetic

Qualifiers can be used in all memory models to overrule the defaults

Industry-standard libraries

The TASKING M16C VX-toolset compiler contains all the necessary ISO C++/ISO C libraries, run-time libraries, and floating-point libraries. The floating-point libraries are supplied in a number of highly useful variants that are assembly optimized.

Floating-point libraries include:

- **Optimized single precision float support**
- **Double precision floating point support**
- **Complex and imaginary type support**
- **Trapping and non-trapping support**

Source code for most of the library routines allows you to tailor the libraries to your specific application.

ASSEMBLER

The TASKING assembler is supplied complete with linker/locator, librarian and object format utilities.

Assembler features include:

- **Full macro and conditional assembler**
- **Optimizing jump/call instructions**
- **Extensive section directives**
- **Full assembly source-level debugging**

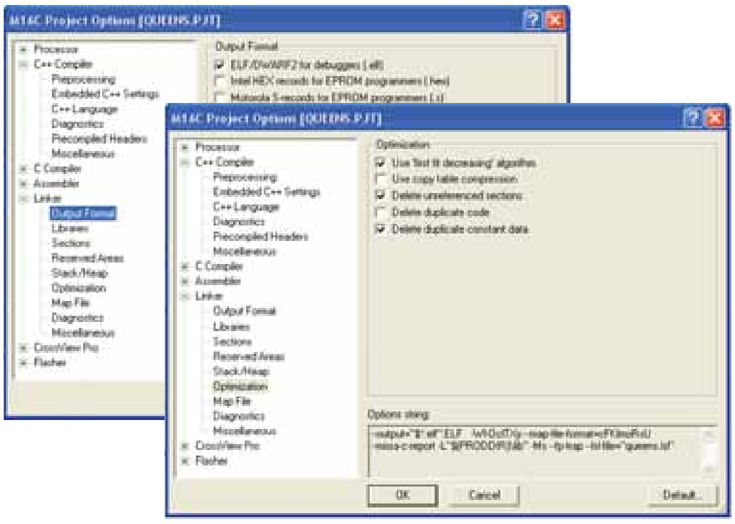
LINKER/LOCATOR

The linker/locator plays a pivotal role in the software building process by combining the compiler- and assembler-generated code and data sections with possible library functions and allocating the result into available target memory.

The target-independent linker in the TASKING M16C toolset allows you to accurately describe available target memory and fully control the behavior of the locating process, so that all pieces of code and data fall into their intended places.

Linker/Locator features include:

- **Automatic and user-specified allocation in target memory**
- **Powerful, intuitive linker script language**
- **Data/code section initialization**
- **Powerful data/code overlaying facilities**
- **Smart linking removing unreferenced and duplicate code/data**
- **Industry-Standard ELF/Dwarf object output format**
- **SREC and Intel HEX ROM image output formats**



Output Format
[✓] ELF/DWARF2 for debuggers (.elf)
[ ] Intel HEX records for EPROM programmers (.hex)
[ ] Motorola S-records for EPROM programmers (.s)

Optimization
[✓] Use 'first fit decreasing' algorithm
[ ] Use copy table compression
[✓] Delete unreferenced sections
[ ] Delete duplicate code
[✓] Delete duplicate constant data

Options string
```
[illegible]
-misra-c-report -L"$(PRODDIR)\lib" [illegible] -lsl-file="queens.lsl"
```

Fully-configurable linker

CROSSVIEW PRO DEBUGGER

The TASKING M16C CrossView Pro debugger is the perfect partner for checking, verifying and debugging your application. With its easy-to-use interface and powerful, extensive debugging features, CrossView Pro helps you debug your applications faster. CrossView Pro provides multiple, resizable and independently controlled windows, enabling you to choose the windows you need to view the relevant aspects of your code during debugging. It combines the flexibility of the C language with the control of code execution found in assembly language, bringing functionality that reduces the amount of time spent on testing and debugging.

Functionality includes:

- **Basic through to advanced debugging features**
- **Tracking scope and monitoring locals**
- **Intuitive navigation through the source window**
- **Double-click, right mouse button, and tip-point functions**
- **Clipboard copy and paste**
- **Bubble-Spy™ technology for easy and quick inspection of variable contents**
- **Code/data coverage and profiling (performance analysis) in CrossView Pro Simulator**

Source window

The main window is the source window. It allows you to view source; step through your code; set and clear breakpoints, assertions and code coverage markers; watch and show variables; search for strings, functions, lines and addresses; and evaluate expressions. The source window can display code in C source, assembly, or mixed.

While moving the mouse pointer over your source, our Bubble-Spy^{TM} technology provides quick checks of variable values and function prototypes. Double-clicking on a function call automatically navigates to the corresponding source. From the cursor in the debugger source window, you can jump directly into the EDE editor, allowing immediate access to the source line of the problem that needs correcting.

Multiple information windows

The CrossView Pro debugger offers a wealth of information windows allowing you to monitor and modify data objects, CPU registers, memory locations and the stack.

The **data window** enables you to watch and modify data objects. Data structures can be shown collapsed as well as expanded.

Register windows can be configured to display any set of CPU and peripheral registers as well as their values. Defining multiple Register windows helps you organize your focus.

The **stack window** displays the contents of the function-call stack frame. You can easily configure stack-level breakpoints, navigate to the function call's source, and monitor local variables for selected functions.

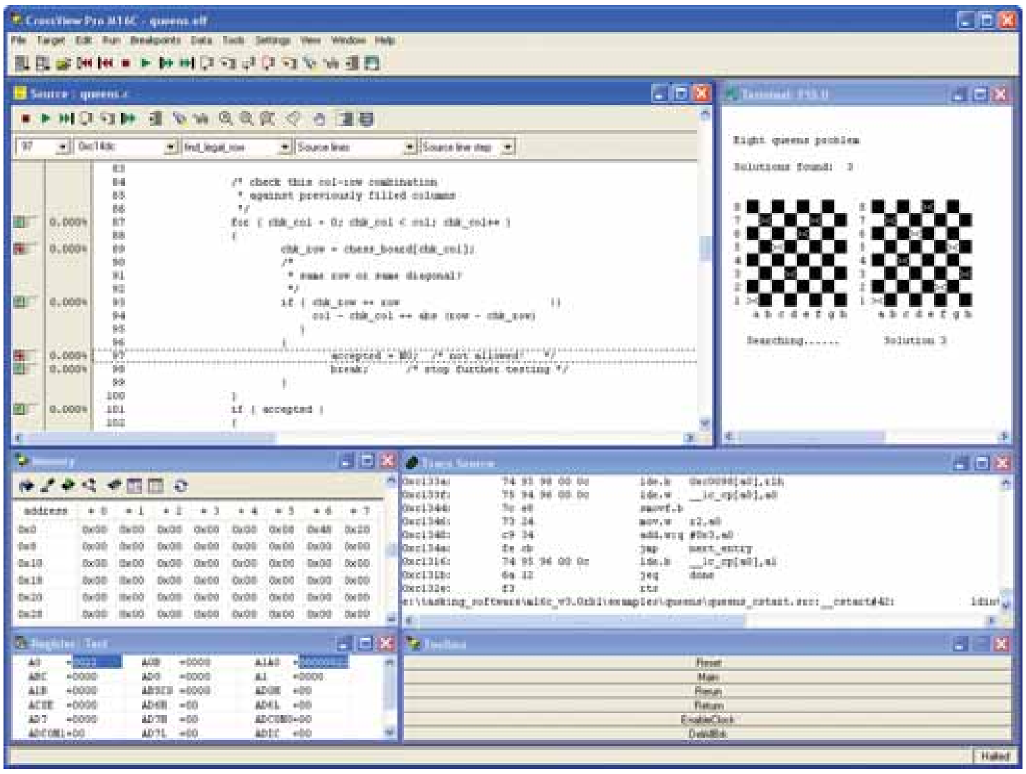
The **memory window** enables you to monitor and modify any memory location, with complete control over size and format of the data, as well as view coverage of the memory range. All information windows are automatically updated, and changed values are highlighted for easy identification. In-situ editing allows you to modify values on the spot.

Advanced breakpoints

Breakpoints halt program execution and return control to the user. In addition to industry-standard code and data breakpoints, you can configure your application to halt based upon instruction counts, cycle counts, or timer counts. All types of breakpoints can be defined as 'stop-and-go' probe points.

Probe points briefly halt and immediately resume execution of the application. During the brief period that the application is halted, only user-specified actions will be performed. Through this mechanism, probe points allow least-intrusive debugging of time critical applications.

Finally, any number and type of breakpoints can be combined into 'breakpoint-sequences'. This allows easy specification of the most complex conditions that need examining.



```
        /* check this col-row combination
         * against previously filled columns
         */
0.000%  for ( chk_col = 0; chk_col < col; chk_col++ )
        {
0.000%          chk_row = chess_board[chk_col];
                /*
                 * same row or same diagonal?
                 */
0.000%          if ( chk_row == row                        ||
                     col - chk_col == abs (row - chk_row)
                   )
                {
0.000%                  accepted = NO;  /* not allowed!   */
0.000%                  break;      /* stop further testing */
                }
        }
0.000%  if ( accepted )
        {
```

```
Eight queens problem

Solutions found:  3

Searching......          Solution 3
```

```
address   +0    +1    +2    +3    +4    +5    +6    +7
0x0       0x00  0x00  0x00  0x00  0x00  0x00  0x48  0x20
0x8       0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x10      0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x18      0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x20      0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x28      0x00  0x00  0x00  0x00  0x00  0x00  0x00  0x00
```

```
0xc133a:     74 95 98 00 0c      lde.b   0xc0098[a0],r1h
0xc133f:     75 94 96 00 0c      lde.w   __lc_cp[a0],a0
0xc1344:     7c e8               smovf.b
0xc1346:     73 24               mov.w   r2,a0
0xc1348:     c9 34               add.w:q #0x3,a0
0xc134a:     fe cb               jmp     next_entry
0xc1316:     74 95 96 00 0c      lde.b   __lc_cp[a0],a1
0xc131b:     6a 12               jeq     done
0xc132e:     f3                  rts
c:\tasking_software\m16c_v3.0r1\examples\queens\queens_cstart.src:__cstart#42:     ldint
```

```
A0     =0000      A0B    =0000      A1A0   =00000000
A0C    =0000      A0O    =0000      A1     =0000
A1B    =0000      A0SCU  =0000      AD0H   =00
ACIE   =0000      AD6H   =00        AD6L   =00
AD7    =0000      AD7H   =00        ADCON0=00
ADCON1=00         AD7L   =00        ADIC   =00
```

Spend less time debugging with CrossView Pro

Multiple execution environments

The CrossView Pro debugger supports multiple execution environments within a standard interface.

- **M16C instruction set simulator debugger**
  The simulator environment allows you to test, debug, and monitor the performance of code in a known and repeatable environment independent of target hardware. It uses the same description file as the linker/locator when locating your application and therefore knows exactly where and how memory is mapped.

  All CrossView Pro features, including C level trace, code/data coverage, performance analysis (profiling), and unlimited amount of code and data breakpoints are available to you, so you can test code before target hardware is available.

- **The TASKING ROM monitor debugger**
  The ROM monitor debugger is a powerful debug instrument that can be used to debug a target without requiring an expensive emulator. The key benefit of the TASKING ROM monitor is that it does not need any regular polling by the host platform and thus offers optimal run-time performance. Many commercially available evaluation boards are supported by the TASKING ROM monitor (e.g. GLYN and Renesas Diamond kit). As the ROM monitor is provided in C source, it can be ported to new or custom hardware easily.

- **Renesas Emulator**
  CrossView Pro supports the Renesas PC4701 Emulator. The GDI (Generic Debug Interface) is an open standard that defines the interface between the execution environment (e.g. simulator, ROM monitor, emulator) and CrossView Pro. It enables full support for enhanced features such as execution trace, code/data coverage and performance analysis.

TASKING ROM synchronization tool

Integrated in the EDE is the TASKING ROM synchronization tool. The 'Sync' dialog which is used with the ROM monitor debugger will synchronize project settings like processor and memory settings. Once the TASKING ROM monitor has been flashed onto the target board using this tool, it will simplify setting the target specifics of your project. By clicking the Sync button, your project settings will be updated with the target specifics found by the TASKING ROM monitor.

File system simulation

CrossView Pro I/O Simulation (IOS) allows the use of standard ISO C system calls such as open(), read(), printf() and scanf() within your embedded application in order to interface with the host platform file I/O services.

Using IOS, you can read from and write to files on the host platform or a CrossView Pro Virtual I/O window directly. I/O Simulation will work in any CrossView Pro target execution environment.

Program performance analysis

CrossView Pro provides a number of performance analysis capabilities to help you further optimize your application as well as shorten your debug session.

- **Code coverage**
  Code coverage enables you to check whether specific parts of your application code have actually been executed.

- **Profiling**
  Profiling allows you to perform timing analysis on the complete application or specific parts of it. Based upon the profiling information you can easily decide which functions should be optimized for speed.

- **Graphical Data Analysis**
  Graphical Data Analysis simplifies quick detection of gross errors in signal processing routines and allows you to analyze the data without the need of reviewing or post-processing large files of raw data.

  Four different analysis types are pre-defined:

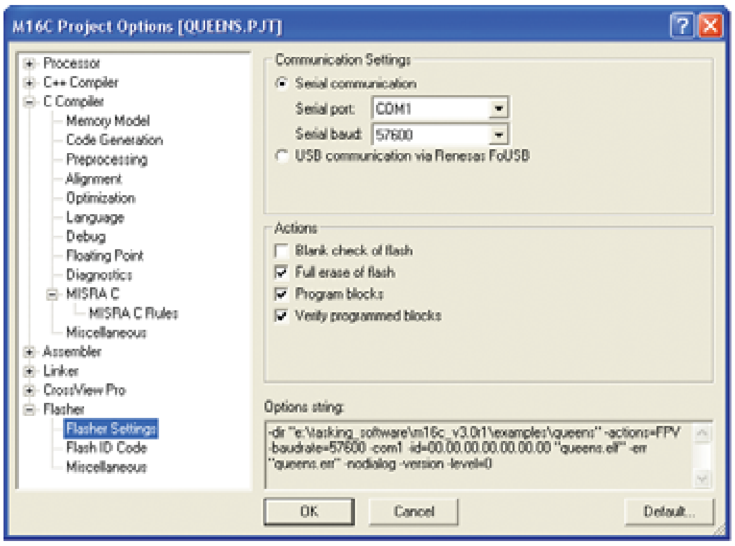
  - **x-t plotting**
  - **FFT (Fast Fourier Transformation)**
  - **Power spectrum**
  - **Eye diagram**

Easy debugging of RTOS-based applications

TASKING's Kernel-aware Debugging Interface (KDI) defines an open standard interface between CrossView Pro and an RTOS-Aware Debug Module (RADM). The RADM can be used to add kernel-awareness to CrossView Pro for any commercial or proprietary RTOS.

TASKING M16C FLASHER

Integrated in the TASKING EDE is an easy-to-use flash tool called the TASKING M16C Flasher. This flasher interfaces to the standard Renesas flash tool available on the chip. The TASKING M16C Flasher allows you to directly flash an ELF/Dwarf, Motorola S-Rec or Intel Hex file into the chip. Flashing a program can be done via a Serial or USB connection.



Communication Settings

[✓ Serial communication]

Serial port: COM1

Serial baud: 57600

USB communication via Renesas FoUSB

Actions

- [ ] Blank check of flash
- [x] Full erase of flash
- [x] Program blocks
- [x] Verify programmed blocks

Options string:

```
-dir "e:\tasking_software\m16c_v3.0r1\examples\queens" -actions=FPV
-baudrate=57600 -com1 -id=00.00.00.00.00.00.00 "queens.elf" -err
"queens.err" -nodialog -version -level=0
```

COOPERATION WITH THIRD-PARTIES

Our extensive third-party cooperation ensures that you have access to the tools you need to be most productive. Altium works closely together with manufacturers of in-circuit-emulators, real-time kernels, TCP/IP connectivity, CAN solutions and evaluation boards for the M16C. For more detailed information on M16C partners, please visit: www.tasking.com/m16c

CUSTOMER SUPPORT

When you purchase a TASKING product, it is the beginning of a long-term relationship. Altium is dedicated to providing quality products and support worldwide. This support includes program quality control, a product update service, and support personnel ready to answer your questions by telephone, fax or email.

A maintenance period is included with the purchase of TASKING products and entitles you to enhancements and improvements as well as individual response to problems. Annual maintenance agreements are available to extend the initial support period.

PRODUCT PACKAGING AND ORDERING CODES

Each TASKING product comes with full documentation in easy-to-use binders. The documentation is also available online and provides full-text search capabilities for quick and easy lookup of topics.

The VX-toolset for M16C is available for PC/Windows and SUN/Solaris.

Product code	Package contents
07-200-299-024	EDE/Editor
	C/C++/EC++ Compiler
	Assembler
	Linker/Locator
	CrossView Pro Simulator
	ROM Monitor Debugger
	TASKING Flasher

A trial version of the VX-toolset for M16C is downloadable from our website at: www.tasking.com/m16c

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

Australia

Altium Limited
3 Minna Close, Belrose
NSW 2085
Ph: +61 2 8622 8100
Fax: +61 2 8622 8140
sales.au@altium.com
support.au@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

www.tasking.com