C166/ST10 Tool Chain v7.0r1

RELEASE NOTE

C166/ST10 Product Page, C166/ST10 Support Page, TASKING Home Page

SUMMARY

This release note describes the changes and new features of all TASKING C166/ST10 products with respect to v7.0r1.

The following parts are described:

The solved and known problems are not part of this readme, they are described in separate files: "`solved_<name>_7_0r1.txt`".

The main reasons for this release are:

- New C166/ST10 Embedded Development Environment (EDE) with numerous new features like Easy / Expert modes, Project Spaces, improved project management using right mouse-click menus and clear windows, CodeSense, improved Tags and Snippets. Additional extensions like an HTML browser, an FTP client and much more can be easily loaded.

- New Infineon Technologies C166S v2.0 & STMicroelectonics Super10 architecture families support. These architecture families have been defined and supported from C++, C and assembly in the LONDON project. They offer higher speeds through higher clock rates, One Cycle Per Instruction (OCPI) technology, a Multiply-Accumulate (MAC) co-processor offering basic DSP operations, a scalable interrupt vector table, fast local register banks (non-memory mapped) for shorter interrupt latency, a fully interlocked pipeline and much more. Please contact Infineon Technologies for details and availability of derivatives of the C166S v2.0 architecture family. Please contact STMicroelectronics for details and availability of derivatives of the Super10 architecture family.

- Updated C++ compiler front-end, added C++ Standard Template Libraries (STL) and improved template instantiation offers an even more mature EC++/C++ development environment.

- MISRA C, enhanced error checking following the MISRA C guidelines (see http://www.misra.org.uk) to improve the quality of your application and therefore reduce debug time.

- C Compiler and assembler warnings can exit like errors to accommodate the robustness of application build procedures and therefore improve the quality of your application.

- Dramatic code size reduction improvements introduced in v6.0r6 which are further enhanced and extended in v7.0r1 can result in a code size reduction up to 7.5% on your total application. These optimizations include Common Tail Merging, improved Common Sub-expression Elimination (CSE) register allocation and improved function context management.

- New fast and full reentrant software floating point libraries. You can expect execution speed improvements up to 2 times for single precision and up to 1.5 times for double precision floating point operations with respect to any previous version of the C166/ST10 products. The new floating point library now utilizes the user stack, so special considerations regarding the former floating point stack are no longer necessary. This automatically results in full floating point reentrancy to accommodate RTOS' without any requirements whatsoever. Now the library sources are included as well.

- The libraries directory structure has been better organized to improve usability and flexibility. There is a more clear distinction with respect to the 3 supported architecture families, whether or not the used architecture family derivative requires silicon bug workarounds and whether the system stack or the user stack (User Stack Model) must be used for function call return addresses. In addition the formerly combined C and run-time library has been split in separate libraries to accommodate easy switching of the run-time library depending on whether or not the user architecture family derivate contains a MAC coprocessor. Additionally the former "`printf*.lib`" libraries containing the *`printf()`* and *`scanf()`* family functions in their LARGE I/O format, have been replaced by "`fmtio*.lib`" libraries and now also includes pre-build MEDIUM I/O formatter library variants. Finally the former "`f166*.lib`" and "`ntf166*.lib`" libraries have been replaced by the "`fp166*.lib`" libraries where a suffix character identifies trapping or non-trapping. Linking the correct library is handled automatically by both the control program cc166 and the C166/ST10 EDE.

- CrossView Pro now includes Files System Simulation (FSS) which offers an ANSI-C standard method for performing file I/O on an embedded system using the file system on the host PC. Regular ANSI-C *`fprintf()`* and *`fscanf()`* family functions can now be used to respectively send and received data from the embedded system to and from files which reside on the host PC using CrossView Pro as the host PC file handling engine. In addition the standard C library *`printf()`* and *`scanf()`* functions are now associated with FSS instead of Simulated I/O but the behavior remains the same.

- CrossView Pro now supports OCDS (On Chip Debug Support) as a debugging channel for Infineon Technologies devices like the C165UTAH and C161U. Connecting CrossView Pro to these OCDS enabled devices requires a parallel cable or a JTAG connection either on-board on an evaluation board or by using a stand alone wiggler. OCDS offers faster download speeds and more flexible debugging options.

- Added full support for long filenames (including white-space characters) to support any host platform, project structure or application build procedure.

- New documentation structure where all online documentation is stored in the "`doc`" directory of the

installed C166/ST10 product. This directory includes "`html`" and "`pdf`" directories where respectively the HTML and PDF variants of the online documentation are stored. The known and solved problems files "`solved_<name>_7_0r1.txt`" are stored in the top level directory where the C166/ST10 product is installed.

- Solving of reported problems.

EMBEDDED DEVELOPMENT ENVIRONMENT

The C166/ST10 Embedded Development Environment (EDE) has been dramatically renewed and extended with numerous new features like:

- Easy / Expert modes to make the C166/ST10 product easier for regular use and flexible for expert use.
- Project Spaces which groups a number of projects into a logical unit, improved project management using right mouse-click menus and clear project definition windows to make adding and removing files to and from a project easy.
- CodeSense to guide and assist you when you write your source code.
- Improved Tags to get a structured overview of your sources and their relations.
- Snippets for easy interactive clipboard kind of behavior.
- Additional extensions like an HTML browser, an FTP client and much more can be easily loaded.

Easy & Expert Mode

The 'Expert Mode' entry located in the 'EDE' menu allows switching between the Easy and Expert operation modes.

Easy operation mode means that a lot of the advanced items in the 'EDE' menu are no longer available for use. This mode allows you to quickly setup your project and start working right away without the need to know all possible options from the 'EDE' menu. The C166/ST10 EDE takes care of setting all options correctly as much as possible. This is the default mode setting.

Expert operation mode can be enabled by clicking on the 'Expert Mode' entry in the 'EDE' menu. Now all possible options of the tools are yours to control. If you switch back to the Easy operation mode, then all your Expert operation mode settings will remain active.

All examples which are part of the C166/ST10 products are pre-configured for the Easy operation mode.

Projects & Project Spaces

All your projects can now be grouped together in Project Spaces. The 'C166_ST10 Examples' project space is the default Project Space containing all C166/ST10 examples of the product you have installed. If you want to open an other than the default example (project) within this Project Space, just select one, click on the right mouse button and select the 'Set as Current Project' menu entry. The EDE make and rebuild commands which you use to build or rebuild your project only work on the current project.

Adding files to and removing files from a project has become easier than ever. Just click with your right mouse button on a source file and the pop-up dialog contains an entry to add this file to the current project. Click with your right mouse button on a file in the project window and the pop-dialog contains an entry to remove this file from the project.

You can create your own project spaces from the 'Project' menu. From this menu you can also add new or existing projects to your own Project Space.

For more information, please refer to the C166/ST10 EDE online help system.

CodeSense

CodeSense virtually looks over your shoulder and gives you useful information in the form of hints as you type your source code.

For example if you are programming a *`printf`* statement, it will show you the next expected parameter and the prototype of this function in a small yellow balloon-help box. If you have already defined a structure with a lot of members and from a certain location within your code you want to access a member, just type in the structure name and a dot, and CodeSense will show you a list with all possible members. You can select a member from this list or look-up where this member is defined. This also works for C++ language elements.

If you hover the mouse pointer over function name, CodeSense will show the prototype of this function. This also applies to variables, structures, etc. Just hover the mouse pointer over a C or C++ language element and CodeSense will show you whatever information is relevant.

In order to have all this information ready at hand, CodeSense automatically builds a database using all the files in the "`include`", "`include.cpp`" and "`examples`" directories when you first start the C166/ST10 EDE. This is shown by the CodeSense green light which is displayed near the right bottom corner of the EDE window which will become gray when this database is completed. That is the signal that CodeSense is operational.

You can also add your own databases by clicking with the left mouse button on the CodeSense light and select the libraries option. Please follow the instructions to add your own CodeSense information from your own application to the CodeSense database.

Browsing Tags

Tags reflect the cross references in your application. By building a Tags file and graphically browsing your source code using this Tags file, you can get a good overview on cross references in your application like what global variables are defined, where are they used, which enumeration types are defined, which global functions are defined and where are they used.

Browsing of Tags can help you in getting to know someone else's source code easily and quickly without the need to dive into every detail.

From the 'Projects' menu select the 'Build Tags' option to build a Tags file which reflects your current project. Next open this Tags file from the browse window which you find in the output window in the bottom of the EDE window. Please select the 'Browse' TAB in this window and open the Tags file which has the same name as your current project using the "`*.ptg`" file

extension. Start browsing your application.

Snippets

The C166/ST10 EDE comes with some pre-build Snippets which can aid you in improving your coding speed and efficiency. Basically Snippets are cut-copy-paste pieces of text which you can select from the Snippets library and drop into your source code. The pre-build Snippets are available from the Snippets library by selecting the right most icon which you can find in the left bottom of the EDE window.

You can even create your own Snippets by simply copying a piece of text or source code on the clipboard, click with the right mouse button on the Snippets library and select to add your Snippet to it.

Snippets can also contain some interactive elements which will be activated when you are dropping a Snippet in your source code. For example if you select a function header Snippet, it will ask for the name of the function which will automatically be filled in when the Snipped in dropped in the source code. Please see the C166/ST10 EDE online help system for more information.

Application extensions

There are a number of C166/ST10 EDE application extensions like an HTML browser and an FTP client which are not available by default. These application must be loaded in the EDE before they can be used. Please select the ‘Tools’ menu, select the ‘Customize’ menu entry and finally select the ‘Libraries’ sub-menu entry. This will open a list of application extensions which can be loaded by selecting the extensions you want to use.

C166S V2.0 / SUPER10 ARCHITECTURE FAMILIES

The Infineon Technologies C166S v2.0 & STMicroelectonics Super10 architecture families are now supported from a C++, C and assembly level. These architecture families have been defined and supported in the joined chip vendor LONDON project. These new architectures are noted throughout all printed and online manuals as the 2nd extended architecture "`ext2`".

These architecture families offer higher speeds through higher clock rates, One Cycle Per Instruction (OCPI) technology, a Multiply-Accumulate (MAC) co-processor offering basic DSP operations, a scalable interrupt vector table, fast local register banks (non-memory mapped) for shorter interrupt latency, a fully interlocked pipeline and much more.

At the moment of release of this version of the C166/ST10 products, no broad market derivatives were available yet for these new architecture families.

Although this was the case, two bondout (“C166S v2.0 Bondout” and “Super10 Bondout”) devices are selectable from the C166/ST10 EDE anyway, all architecture features are supported from a source code level (pragmas and controls) as well as from the control program *cc166* and the EDE, a cycle-accurate simulator is available and finally a serial connection to the STMicroelectronics “Super10 evaluation board” is available. All architecture support details regarding the C compiler, assembler, linker/locater and the CrossView Pro

debugger are described in the online manuals.

Please find below an excerpt of the architecture features which are directly supported from the C166/ST10 products:

- Scalable and relocatable interrupt vector table (4, 8, 16 or 32 bytes per vector and anywhere in memory) offering inlining of small pieces of code in a single vector.
- 2 non-memory mapped local register banks provide a shorter interrupt latency. Each register bank has its own user stack area.
- Bypass the interrupt vector table for 2 interrupt handlers provide a shorter interrupt latency.
- Saving and restoring MAC (Multiply-Accumulated unit for basic DSP operations) related registers for securing the context of a function when interrupted.
- Instruction reordering improves execution speed because of best possible utilization of the fully interlocked pipeline.
- Jump prediction improves the execution speed because of a pre-direction selection before a jump is actually evaluated and executed.

Please contact Infineon Technologies for details and availability of derivatives of the C166S v2.0 architecture family.

Please contact STMicroelectronics for details and availability of derivatives of the Super10 architecture family.

MISRA C

MISRA C provides enhanced error checking following the MISRA C guidelines to improve the quality of your application and therefore reduce debug time (see http://www.misra.org.uk).

Basically this is done by prohibiting certain hazardous (although fully valid) ANSI-C constructs. For example the if(a=b) statement is a fully valid ANSI-C statement. However it can cause a hazardous situation when your colleague starts working with your source code and misses the fact this a=b is not only a Boolean expression but also an assignment. This can cause a lot of confusion, a lot of debug time but even worse it can result in a run-time error when executing the application which can not be discovered during compilation.

MISRA C performs these kind of additional checks and reports errors when a violation is detected. Therefore the useful MISRA guidelines are implemented which each check a certain possible hazardous ANSI-C construct. These rules (or guidelines) can be set to the required rules defined by MISRA or you can set them to your own requirements.

The ‘Save/Restore Options’ menu entry from the ‘EDE’ menu allows you to save your own required MISRA C options into an option file which can be reused by your colleagues (or team members if you are a project manager) to enforce that everyone uses the same MISRA C rules. The reuse of this option file can be enabled from the ‘MISRA C Compiler Options’ menu entry from the ‘EDE’ menu where the MISRA C options can be read from an external file.

Finally the linker/locator can generate a report which lists all the used MISRA C rules for all objects and libraries which are used to build the application. This report can be used by project managers to further ensure that all team members base their work on the same set of MISRA C rules.

The list MISRA C rules (or MISRA guidelines) which are supported by the C166/ST10 products can be found in the online manuals

In addition to MISRA C, the C166/ST10 products now support not only exiting a build process when an error is encountered but also exit the build process when warnings are encountered. Using this method you can ensure that your application is error and warning free which of course results in improved application quality.

FLOATING POINT LIBRARIES

The new software floating point libraries are fast and fully reentrant. You can expect execution speed improvements up to 2 times for single precision and up to 1.5 times for double precision floating point operations with respect to any previous version of the C166/ST10 products. Additionally the floating point libraries now utilize the user stack, so special considerations regarding the former floating point stack are no longer necessary. This automatically results in full reentrancy to accommodate RTOS' without any requirements whatsoever. As a minor detail, the sources of the software floating point libraries are included in the C166/ST10 C and C/C++ Compiler Packages.

To improve the floating point performance dramatically, the assembly interface of the floating point functionality had to be changed to accommodate this. This new assembly interface is fully described in the online C Compiler manual. Normally there should not be a need for you to look up this information however there are some pitfalls when mixing old pre v7.0r1 floating point with this new floating point.

- Pre-build pre v7.0r1 object modules.
- Pre-build pre v7.0r1 libraries.
- Pre-build 3rd party object modules.
- Pre-build 3rd party libraries.
- Assembly which directly interfaces on floating point routines.

If you have either one of these situations, then do the following:

- Rebuild every C or C++ module that uses floating point from their source code.
- Check if your 3rd party modules and libraries use floating point that is v7.0r1 compliant. If not, rebuild them yourself if you have the source code or contact your 3rd party vendor if you are not sure about v7.0r1 compliancy.
- Change your assembly modules to interface on the new floating point. Please consult the online C Compiler manual for the floating point interface details.

The improved floating point library no longer uses a separate floating point stack for its calculations. Because of this change there is no need anymore to reserve any memory for this stack. All pre-defined symbols which were used to work with this stack are also no longer available.

In addition any call to the functions *_fppushus()* and *_fppopus()* can be removed from any source code. RTOS' always took care of the pre v7.0r1 floating point stack by using these functions. If you have the source code of the RTOS you are using, then please just simply remove these functions. If you do not have the source code, please contact your RTOS vendor for an updated RTOS which is compliant to the v7.0r1 release of the C166/ST10 products.

Please also see the section below about the changed library structure because it contains the changes for the floating point libraries with respect to the directory, the name and the options to link the correct version of these libraries with your application.

LIBRARY DIRECTORY STRUCTURE

The libraries directory structure has been reorganized to improve usability and flexibility. There is a more clear distinction with respect to the 3 supported architecture families, whether or not the used architecture family derivative requires silicon bug workarounds and whether the system stack or the user stack (User Stack Model) must be used for function call return addresses.

In addition the formerly combined C and run-time library has been split in separate libraries to accommodate easy switching of the run-time library depending on whether or not the user architecture family derivate contains a MAC coprocessor.

Additionally the former "`printf*.lib`" libraries containing the *printf()* and *scanf()* family functions in their LARGE I/O format, have been replaced by "`fmtio*.lib`" libraries and now also includes pre-build MEDIUM I/O formatter library variants. Finally the former "`f166*.lib`" and "`ntf166*.lib`" libraries have been replaced by the "`fp166*.lib`" libraries where a suffix character identifies trapping or non-trapping. Linking the correct library is handled automatically by both the control program cc166 and the C166/ST10 EDE.

The libraries are now organized in 4 basic library sets: one for C166 & ST10x166 architecture derivatives (subdirectory "`166`"), one for the Gold derivative (subdirectory "`goldp`"), one for C167, ST10x167 & ST10x262 architecture derivatives (subdirectory "`ext`") and one for C166S v2.0 & Super10 architecture derivatives (subdirectory "`ext2`").

These 4 basic library sets are additionally organized in 2 variants: one standard variant (not available for the Gold derivative) and one variant with all silicon bug workarounds enabled. The subdirectory names for this last variant are followed by the character "`p`" (subdirectories "`166p`", "`extp`", "`ext2p`" and "`goldp`").

All 7 library sets (4 basic * 2 variants -1 for the Gold derivative) are also available for the User Stack Model (USM). All these subdirectories are additionally prefixed with the character "`u`".

Which library set must be used depends on the used architecture derivative. By default the control program *cc166* assumes the usage of a C166 & ST10x166 architecture derivative without any silicon bug workarounds enabled which results in automatic usage of the library set in the subdirectory "`166`". By default the C166/ST10 EDE assumes the usage of a C167 architecture derivative without any silicon bug workarounds enabled which results in automatic usage of the library set in the subdirectory "`ext`".

Please find the details of this structure and the corresponding new tool options below.

Library directory structure (library sets)

lib\ *[u]166*	C166 & ST10x166 architecture family derivatives.
lib\ *[u]166p*	C166 & ST10x166 architectures family derivatives with all workarounds for appropriate cpu bugs enabled.
lib\ *[u]ext*	Extended C167 and ST10x167 architectures family derivatives like C167, C161, C163, C164, C165, ST10x167, 168, 169, 262, 272.
lib\ *[u]extp*	Extended C167 and ST10x167 architectures family derivatives with all workarounds for appropriate cpu bugs enabled.
lib\ *[u]ext2*	2nd extended architecture families like the C166S v2.0 and Super10.
lib\ *[u]ext2p*	2nd extended architectures families with all workarounds for appropriate cpu bugs enabled.
lib\ *[u]goldp*	Gold derivative with all workarounds for appropriate cpu bugs enabled.

Note: *u* = User Stack Model variant. By default the libraries use the system stack.

Options for *cc166* for selecting a library set

-B[bf]	Select the "`166p`" library set under the condition that [bf] is any upper case character which enables a silicon bug workaround which is appropriate for C166 & ST10x166 architecture family derivatives.
-x	Select the "`ext`" library set.
-x -B[bf]	Select the "`extp`" library set.
-x2	Select the "`ext2`" library set.
-x2 -B[bf]	Select the "`ext2p`" library set.
-xm	Select the "`goldp`" library set.
-lib <dir>	Select a user defined library set.
-P	Select User Stack Model library set variants.

Note: with *–lib <dir>*, the control program searches all known libraries in the <dir> directory. The only libraries known to the control program are the libraries which are described below.

Libraries included in a library set when applicable

rt166<mem>[s][m].lib	The run-time libraries.
c166<mem>[s].lib	The C libraries.
cp166<mem>[x].lib	The C++ libraries.
fp166<mem>[t].lib	The floating point libraries.
fmtio<mem><var>[s].lib	The `printf()` / `scanf()` I/O formatter libraries.
can166<mem>.lib	The Infineon Technologies CAN libraries.

Note: mem = memory model (t = tiny, s = small, m = medium, l = large).

Note: s = floating point single precision. By default the precision is double.

Note: m = MAC optimized. By default the run-time libraries do not use MAC instructions.

Note: x = exception handling. By default exception handling is disabled.

Note: t = trapping. By default non-trapping is used.

Note: var = variant (m = MEDIUM, l = LARGE).

Note: before v7.0r1, the default *f166*.lib* library was trapping. This default is changed into non-trapping because it is faster and due to the library changes themselves, trapping is no longer an issue. Less traps will occur with the new floating point libraries.

Note: the SMALL variant *fmtio*.lib* libraries are included in the C library by default. They do not support floating point printing, nor the usage of precision specifiers.

Note: the MEDIUM variant *fmtio*.lib* libraries do not support floating point printing. They do support the usage of precision specifiers. The LARGE variant *fmtio*.lib* libraries supports both floating point printing and the usage of precision specifiers.

Note: the MEDIUM *fmtio*.lib* library variants are not available in any floating point variant because floating point is not supported anyway in the MEDIUM variant.

Note: Please read the Infineon CAN application note the "`ap292201.pdf`" which is located in the "`doc\pdf`" directory of the product for the usage of the Infineon Technologies CAN libraries.

Options for *cc166* for selecting specific libraries

-libmac	Link the *rt166*m.lib* MAC optimized run-time libraries.
-notrap (default)	Link the *fp166*.lib* non-trapping floating point libraries.
-trap	Link the *fp166*t.lib* trapping floating point libraries.
-libfmtio<var>	Link the *fmtio*.lib* libraries.
-libcan	Link the *can166*.lib* libraries.

Note: the non MAC optimized libraries are selected by default.

Note: before v7.0r1, -trap was the default although there was no option to select this. This default is now changed to -notrap.

Note: *var* = variant (m = MEDIUM, l = LARGE).

FILE SYSTEM SIMULATION

The CrossView Pro debugger now includes File System Simulation (FSS) which enables the embedded application to use ANSI-C system calls such as `open()`, `read()`, `write()`, `fprintf()` and `fscanf()` that are handled by the host PC file I/O services. These files can be read directly from the host PC, and output can be written to a file on the host PC or in a CrossView Pro Virtual I/O window. File System Simulation is available for all execution environments.

In addition, the standard C library `printf()` and `scanf()` family of functions are now associated with FSS instead of Simulated I/O. The behavior remains the same but instead of displaying the output on dedicated Simulated I/O windows, it is now displayed on Virtual I/O windows. Additionally the libraries have been optimized to only attach the file I/O routines if the application actually uses file I/O. This includes the `exit()` routine, that must close all opened streams before returning to the debugger. FSS is faster than Simulated I/O because it buffers data much more efficiently.

There can be some pitfalls when mixing old pre v7.0r1 I/O routines with this new FSS:

- Pre-build pre v7.0r1 object modules.
- Pre-build pre v7.0r1 libraries.
- Pre-build 3rd party object modules.
- Pre-build 3rd party libraries.
- C++, C and Assembly source code which directly interfaces on the Simulated I/O functions.

If you have either one of these situations, then do the following:

- Rebuild every C or C++ module that use I/O routines from their source code.

- Check if your 3rd party modules and libraries use I/O routines which are v7.0r1 compliant. If not, rebuild them yourself if you have the source code or contact your 3rd party vendor if you are not sure about v7.0r1 compliancy.

- Change your C++, C and Assembly modules to interface on ANSI-C I/O routines.

If your source code interfaces directly on the *`_simi()`* and *`_simo()`* Simulated I/O functions, then please be aware that the v7.0r1 libraries do not utilize these functions anymore. They utilize FSS instead which can not be mixed with Simulated I/O in any way. Although CrossView Pro still supports Simulated I/O, it would be better anyway not to use the *`_simi()`* and *`_simo()`* Simulated I/O functions directly from your source code but use regular ANSI-C *`printf()`* and *`scanf()`* type functions instead which ensures portability of your application.

Because FSS buffers its data by default just as you were used to with Simulated I/O, please beware that you either use the `"\n"` new line character to terminate the information you are printing with *`printf()`* type functions, or use the *`fflush()`* function to flush this buffer in order to force FSS to display its information on the CrossView Pro Virtual I/O windows or to write it into a file. Neither CrossView Pro nor the FSS implementation in the library will perform this operation for you.

Unlike with what you were used to with Simulated I/O, it is no longer necessary to inform CrossView Pro about the use of any streams. The default I/O streams *`stdin`*, *`stdout`* and *`stderr`* are opened on the fly whenever file I/O is used. Please be aware that when you close a Virtual I/O window that it is automatically deactivated. In order to use this particular Virtual I/O window (to which the FSS streams are possibly associated) again, it must be reactivated from the ‘Virtual I/O Setup…’ dialog which can be opened from the ‘Debug’ menu.

Please see the online CrossView Pro manual section about File System Simulation for details on attaching your own streams to certain CrossView Pro Virtual I/O windows and how to capture and feed the default I/O streams *`stdin`*, *`stdout`* and *`stderr`*.

ON CHIP DEBUG SUPPORT

The CrossView Pro debugger now supports OCDS (On Chip Debug Support) as a debugging channel for Infineon Technologies devices like the C165UTAH and C161U. OCDS offers faster download speeds, more flexible debugging options and most importantly it frees the on-chip UART so you can use it freely in your application.

Connecting CrossView Pro to these OCDS enabled devices requires a parallel port connection using a parallel cable or a JTAG connection using a special JTAG cable and host PC JTAG board. Both connect to an evaluation board using an on-board or stand alone wiggler. At the moment of release of this version of the C166/ST10 products, only the Infineon Technologies Easy Utah (with C165UTAH or C161U) evaluation board was available for preparing plug and play support from the EDE and the CrossView Pro debugger. This board contains an on-board wiggler which can therefore be directly connected to a parallel port using a parallel cable.

When installing the CrossView Pro OCDS package, it will prompt you that the OCDS solution only works on MS-Windows NT and that you must have administrator rights in order to install the driver. After installation you will be asked to reboot your PC in order to activate the JTAG driver. This driver is installed in the Windows system directory and will automatically be started when the CrossView Pro OCDS package is used.

If you encounter any problems in trying to connect to the Easy Utah board using the OCDS connection, then please check the following:

- Are you working on MS-Windows NT.
- Did you get no error during installation.
- Did you reboot your host PC.
- Is the parallel cable connected to the Easy Utah board.
- Are the jumpers on the Easy Utah board set correctly.
- Is the parallel cable connected to the host PC.
- Is the parallel port enabled in your host PC BIOS.
- Is the parallel port enabled under MS-Windows NT.
- Is the JTAG driver started when CrossView Pro OCDS is started. Please check the MS-Windows NT Services which you can find in the Control Panel.

At the moment of release of this version of the C166/ST10 products, JTAG drivers for MS-Windows 95, 98 and 2000 were still under development at Infineon Technologies.

INFINEON TECHNOLOGIES DAVE 2.X

Previous versions of the C166/ST10 products already supported DAvE 1.x by means of the C startup code which was generated by DAvE and used when building a DAvE generated application using the C166/ST10 products. The previous version v6.0r6 of the C166/ST10 products already support DAvE 2.x by means of including a DAvE generated "`*.asm`" file in the “Macro Preprocessor…” dialog which you can launch from the “EDE” menu. This file is used by the C166/ST10 products to build a DAvE generated application. Both solutions have the major disadvantage that the options set in DAvE are not reflected by EDE and the EDE project must always be manually configured to include the files generated by DAvE.

This version of the C166/ST10 products supports DAvE 2.x by means of its generated project information file "`*.dpt`" which means that projects created using DAvE can now be easily imported in the C166/ST10 EDE where the memory model, all “CPU Options…” and “BUS Options…” (available from the “EDE” menu) now reflect the settings you made in DAvE. In addition all files created by DAvE will now automatically be added to your own EDE project.

Create your DAvE project (for example "`my_project.dav`") and generate code for the TASKING C166/ST10 products. Now create a new (or open your existing) C166/ST10 EDE project (for example "`ede_project.pjt`") and add the DAvE project information file (for example "`my_project.dpt`") to this EDE project. This file will be shown in the “Other Files” category of your project. Now all files included in your DAvE project will automatically be added to your C166/ST10 EDE project and all DAvE project options will automatically be set accordingly in your EDE project.

The C166/ST10 EDE will only reflect those (E)SFR registers which must be configured when booting the cpu

and before the execution of the EINIT instruction. These registers will be configured in the C startup code. All other registers will be configured from the C code which is generated by DAvE.

Every update to the DAvE project will automatically be imported in your EDE project. It will override any settings with respect to the memory model, “CPU Options…” and “BUS Options…” (both available from the “EDE” menu) you have made from the EDE because these settings are also defined from DAvE. All other C166/ST10 EDE settings will keep your configured values. Changes you make manually to your EDE project settings and the source code generated by DAvE, can not be imported back into your DAvE project. Therefore this should only be done if you plan not to use DAvE anymore for making changes to your project files and setting.

The DAvE support introduced in v6.0r6 is still available in this release of the C166/ST10 products. Please find below the details on how to use this "`*asm`" file mechanism:

1. Create a new EDE project.

2. Add all DAvE generated C modules and header files to the project.

3. Add the "`\lib\src\start.asm`" module to the project.

4. Select the "EDE" menu, select "Macro Preprocessor" from the drop-down menu and then select "Project Options...". Select the "Macro" tab and click on the "Include DAvE 2.x generated macros" checkbox. Now press the Browse button to find the DAvE generated "`my_project.asm`" file.

The EDE project is now fully setup to build the application you have created using DAvE.

LONG FILENAMES AND THE LINKER/LOCATOR

All tools in this version of the C166/ST10 products support long filenames. This has some impact on how options and controls are processed by the tools. The Linker/Locator l166 allows the use of arbitrary long filenames and filenames containing spaces. The latter group must be quoted on the command line or in control files like this:

```
l166 loc TO "out file with spaces" PTOG "filename with spaces.obj"
```

This behavior conflicts with previous behavior for the syntax of the ADDRESSES, MEMORY, RESERVE and ORDER controls. The syntax for those controls is:

```
<control> <sub control>
or
<control> ( <sub control> )
```

Previously, a line like the following would be interpreted correctly:

```
l166 loc TO a.out b.obj c.obj "ME IRAM(0F000h)"
```

However, the new long filename support will interpret the quoted control as not using space as separator. Therefore, "ME IRAM(0F000h)" will be read as the control "ME IRAM" instead of "ME". "ME IRAM" is not a valid control, so it will be interpreted as a filename and a file not found error will be issued. There are two ways around this.

1. Quote only the part that has parenthesis: l166 loc to a.out b.obj c.obj ME "IRAM(0F000h)"

2. Use the parenthesis version: l166 loc to a.out b.obj c.obj "ME(IRAM(0F000h))"

The second form is recommended, because it is easier to read, debug and conforms to the way other (newer) controls are processed.

Please note that this problem only occurs on the command line, where controls with parenthesis have to be quoted for most operating systems. In an invocation file, quoting is only necessary for filenames or identifiers containing spaces. Please also note that using several controls within the same quotes will result in the same problem in most circumstances:

    l166 loc to a.out b.obj "PR ME(IRAM(0F000h))"

This will have the linker/locator look for a file names "PR ME" Nonetheless, the following will be interpreted correctly:

    l166 loc to a.out b.obj "ME(IRAM(0F000h)) PR"

This is because the MEMORY control will be interpreted up to and including the last closing parenthesis. The linker/locator will find only the PRINT control in the remaining part and interpret it correctly. As long as all the controls within the same quotes are of the parenthesis form, or if it is a non-parenthesis control and the very
last within these quotes, interpretation will be correct. We recommend separating all (parenthesis) controls on the command line with separate quotes.

LAST MINUTE CHANGES

The C166/ST10 Macro Preprocessor has been extended with the following control which is not documented in the online or printed manuals.

LINES

Control: LINES[(level)]/NOLINES

Abbreviation: LN/NOLN

Class: Primary

Default: LINES(2)

Description: Sets the level of output of #line strings in the output file. Three levels are supported. At level 0, no #line is output at all. At level 1, the old (6.0r6 and earlier) behavior is available. At this level, #line is only printed at the start of a file change (into a new file or back to an old file) as a result of the include control. At level 2, #line is printed after all built-in macro's, after macro comments and after every new line inside a macro. This level makes sure that the assembler is aware of the exact line in the original source file (.asm) when an error is detected in the result file (.src). Please note that, because the #line directive for the assembler works as a line counter reset, the macro preprocessor will output a lot of #line's.

    Level: Output:

0	No #line.
1	#line at file change due to include().
2	#line at file change + after and during all interpreted macros.

Example: m166 code.asm "LN(2)"; output #line's for the assembler or other tools.

NEW OR CHANGED OPTIONS AND CONTROLS

Note: ext2 is used as a short-hand notation for the 2nd extended architecture families C166S v2.0 and Super10.

C166/ST10 Control program

-lib<dir>	Set library set directory.
-libmac	Link MAC optimized run-time library.
-libcan	Link Infineon Technologies CAN libraries.
-libfmtiom	Link MEDIUM I/O formatter library.
-libfmtiol	Link LARGE I/O formatter library.
-trap	Link trapping floating point library.

C166/ST10 C compiler

-Aw	Enable ‘const’ variable propagation ANSI compliancy check.
-AW	
-BA	Generate LONDON1751 cpu bug bypass code.
-Ba	
-BH	Generate BUS.18 cpu bug bypass code.
-Bh	
-BL	Generate LONDON1 cpu bug bypass code.
-Bl	

-exit	Alternative exit values for warnings.
-i<num>	Set ext2 vector scale factor.
-safer<n>,<n>,...	Enable individual MISRA C checks
-x2	Generate code for ext2.
-xd	Enable MAC support.
#pragma reorder	Enable instruction reordering for ext2 (default on with –x2).
#pragma noreorder	
#pragma savemac	Save MAC registers on interrupt (use with –xd only).
#pragma nosavemac	
#pragma m166include	Include user defined intrinsics macro preprocessor header file.
_cached	Configure vector table bypass for ext2
_localbank(<num>)	Configure local register banks for ext2.
_stacksize (<num>)	Configure local register banks stack size for ext2.
_xsfr	Define SFRs out of (E)SFR space.

C166/ST10 Macro Preprocessor

CHECKUNDEFINED	Warn when undefined macro is used legally.
CU	
NOCHECKUNDEFINED	
NOCU	
LINES[(level)]	Sets the level of output of #line strings in the output file.
LN[(level)]	
NOLINES	
NOLN	

C166/ST10 Assembler

EXTEND2	Generated code for ext2.
EX2	
NOEXTEND2	
NOEX2	
CHECKLONDON1	Check for LONDON1 cpu functional problem.
LONDON1	
NOCHECKLONDON1	
NOLONDON1	
CHECKLONDON1751	Check for LONDON1751 cpu functional problem.
LONDON1751	
NOCHECKLONDON1751	
NOLONDON1751	
CHECKLONDONRETP	Check for LONDON RETP cpu bug.
LONDONRETP	
NOCHECKLONDONRETP	
NOLONDONRETP	
WARNING(number)	Additionally set general warning level.
NOWARNING(number)	

C166/ST10 Linker/Locator

EXTEND2	Link ext2 modules and libraries.
X2	
NOEXTEND2	
NOX2	
EXTEND2_SEGMENT191	Reserve 2nd extended architecture (ext2) segment 191.
X2191	
SET(system settings)	Overrule default section, groups and classes limits (default sections=5000, groups=250, classes=250).

C166/ST10 Disassembler

- Disassemble using the ext2 instruction set.

x2

C166/ST10 Register manager

- Use the ext2 (E)SFR register set.

x2

NEW FILES

Register definition files

regc166v2_0bo.h	Infineon Technologies C166S v2.0 Bondout (ext2).
regc166v2_0bo.def	
regc166v2_0bo.dat	
regsuper10bo.h	STMicroelectronics Super10 Bondout (ext2).
regsuper10bo.def	
regsuper10bo.dat	

Target configuration files

easy161u.cfg	Infineon Technologies EASY UTAH with C161U.
evasuper10.cfg	STMicroelectonics Super10 evaluation board.
simc166sv2_0bo.cfg	C166S v2.0 simulator.
simsuper10bo.cfg	Super10 simulator.

ROM-Monitors and boot programs

mext2*.sre	Ext2 ROM-Monitors.
bext2*.sre	Ext2 boot programs.

Debug instruments

diocds166.dll	OCDS DI (for MS-Windows NT only).
gdilib.dll	Ext2 simulator DI (for MS-Windows only).
simu_fm.a.out	Ext2 simulator DI (for SUN/Solaris only).

C startup code

cstartx2.asm	Ext2 C startup code.
