MSP430 GCC

User's Guide

Literature Number: SLAU646C
September 2015–Revised May 2018

Contents

List of Figures

List of Tables

MSP430 GCC

Preface: Read This First

How to Use This User's Guide

This manual describes only the setup and basic operation of the MSP430™ GCC compiler and the software development environment. It does not fully describe the MSP430 GCC compiler or MSP430 microcontrollers or the complete development software and hardware systems. For details on these items, see the appropriate documents listed in Related Documentation.

This manual applies to the use of MSP430 GCC as stand-alone package or within the Code Composer Studio™ (CCS) IDE v8.x and with the TI MSP-FET, MSP-FET430UIF, eZ-FET, and eZ430 development tools series.

These tools contain the most up-to-date materials available at the time of packaging. For the latest materials (including data sheets, user's guides, software, and application information), visit the TI MSP430 website or contact your local TI sales office.

Information About Cautions and Warnings

This document may contain cautions and warnings. The information in a caution or a warning is provided for your protection. Read each caution and warning carefully.

CAUTION

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

WARNING

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

Related Documentation

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The MSP430 website contains the most recent version of these documents.

The GCC documentation can be found at http://www.gnu.org. All related information for the MSP430 GCC compiler is available at http://www.ti.com/tool/msp430-gcc-opensource.

Documents that describe the Code Composer Studio tools (CCS IDE, assembler, C compiler, linker, and librarian) can be found at http://www.ti.com/tool/ccstudio. A CCS-specific Wiki page (FAQ) and the TI E2E™ Community support forums provide additional help.

(1) MSP430, Code Composer Studio, E2E, eZ430-Chronos, LaunchPad are trademarks of Texas Instruments.
(2) macOS is a registered trademark of Apple Inc.
(3) Linux is a registered trademark of Linus Torvalds.
(4) Windows is a registered trademark of Microsoft Corp.
(5) All other trademarks are the property of their respective owners.

MSP430 GCC documentation

Using the GNU Compiler Collection, Richard M. Stallman (http://gcc.gnu.org/onlinedocs/gcc.pdf). Refer to the *MSP430 Options* section.

GDB: The GNU Project Debugger, Free Software Foundation, Inc. (https://sourceware.org/gdb/current/onlinedocs/)

GCC for MSP430™ Microcontrollers Quick Start Guide

Calling Convention and ABI Changes in MSP GCC

CCS documentation

MSP430™ Assembly Language Tools User's Guide

MSP430™ Optimizing C/C++ Compiler User's Guide

Code Composer Studio™ v8.x for MSP430™ User's Guide

MSP430 development tools documentation

MSP430™ Hardware Tools User's Guide

eZ430-F2013 Development Tool User's Guide

eZ430-RF2480 User's Guide

eZ430-RF2500 Development Tool User's Guide

eZ430-RF2500-SEH Development Tool User's Guide

eZ430-Chronos™ Development Tool User's Guide

MSP-EXP430G2 LaunchPad™ Development Kit User's Guide

Advanced Debugging Using the Enhanced Emulation Module (EEM) With Code Composer Studio Version 6

MSP430 device data sheets

MSP430 device family user's guides

MSP430x1xx Family User's Guide

MSP430x2xx Family User's Guide

MSP430x3xx Family User's Guide

MSP430x4xx Family User's Guide

MSP430x5xx and MSP430x6xx Family User's Guide

MSP430FR4xx and MSP430FR2xx Family User's Guide

MSP430FR57xx Family User's Guide

MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide

If You Need Assistance

Support for the MSP430 devices and the hardware development tools is provided by the TI Product Information Center (PIC). Contact information for the PIC can be found on the TI website. The TI E2E Community support forums for the MSP430 provide open interaction with peer engineers, TI engineers, and other experts. Additional device-specific information can be found on the MSP430 website.

1 Introduction

TI has partnered with Mitto Systems Limited (http://www.mittosystems.com) to bring you a new and fully supported open-source compiler as the successor to the community driven MSPGCC. The MSP430 GCC uses the MSP430 ABI and is compatible with the TI compiler. This free GCC compiler supports all MSP430 devices and has no code size limit. In addition, this compiler can be used as a stand-alone package or used within Code Composer Studio (CCS) IDE v6.0 or later. Get started today in Windows, Linux, or macOS environments.

Table 1 compares the MSP430 TI and GCC compilers.

Table 1. MSP430 TI and GCC Compilers Comparison

Compiler	Proprietary TI Compiler	MSP430 GCC	MSPGCC
Code Size and Performance	✓✓✓	✓	✓
ABI	TI	TI	Community
Integrated in CCS	✓	✓(1)	✗
Stand-alone	✗	✓	✓
Support	TI	TI	Community
Cost Free	✓	✓	✓

(1) The combination of CCS+GCC is completely free of charge with no code size limit.

The MSP430 GCC supports the following:

- MSP430 CPU 16-bit architecture
- MSP430 CPUX 20-bit architecture
- MSP430 CPUXv2 20-bit architecture
- Code and data placement in the lower (<64K) and upper (>64K) memory areas and across the memory boundary
- The hardware multiplier of the MSP430 microcontrollers

This manual describes the use of the MSP430 GCC compiler with the MSP430 ultra-low-power microcontrollers. The MSP430 GCC compiler can be used within CCS version 6.0 or later, or it can be used as a stand-alone package. The compiler supports Windows®, Linux®, and macOS® operating systems. This manual describes only CCS for Windows operating systems. The versions of CCS for Linux and macOS operating systems are similar and, therefore, are not described separately.

2 Installing MSP430 GCC Compiler

MSP430 GCC supports Windows, Linux, and macOS:

- Windows 7 32 bit or 64 bit
- Windows 8 32 bit or 64 bit
- Windows 10 32 bit or 64 bit
- Linux 32 bit or 64 bit
- macOS 64 bit

You can install the MSP430 GCC using any of the following methods:

- MSP430 GCC compiler is installed by default by CCS v7.2 and higher.
- In CCS releases prior to v7.2, the MSP430 GCC (compiler only) is available in the CCS Apps Center. The corresponding MSP430 GCC support files (header and linkers) are downloaded with a standard MSP430 emulation package. For details, see Section 2.1.
- MSP430 GCC can be also downloaded as stand-alone package. For details, see Section 2.2.

2.1 Installing MSP430 GCC in CCS Releases Prior to v7.2

MSP430 GCC compiler can be installed within CCS v6.0 or higher in two ways: either when CCS is installed or as an add-on to an existing CCS installation.

1. During the install process of CCS v6.0, select the MSP430 GCC compiler to be installed as an "add-on" (see Figure 1). MSP430 GCC is installed the first time you run CCS (see Figure 2).

Figure 1. MSP430 GCC With CCS Installer

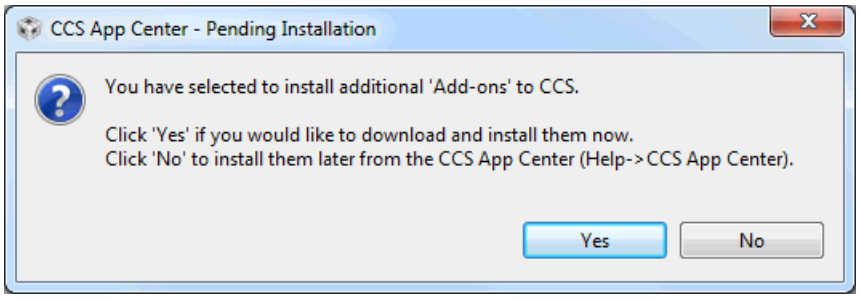

Figure 2. MSP430 GCC With CCS Installer

2. If CCS is already installed without MSP430 GCC, MSP430 GCC can be added at a later time through the CCS Apps Center (see Figure 3).
   1. Go to the menu **View** → **CCS App Center**.
   2. Select MSP430 GCC
   3. Click the **Install Software** button to start the installation.

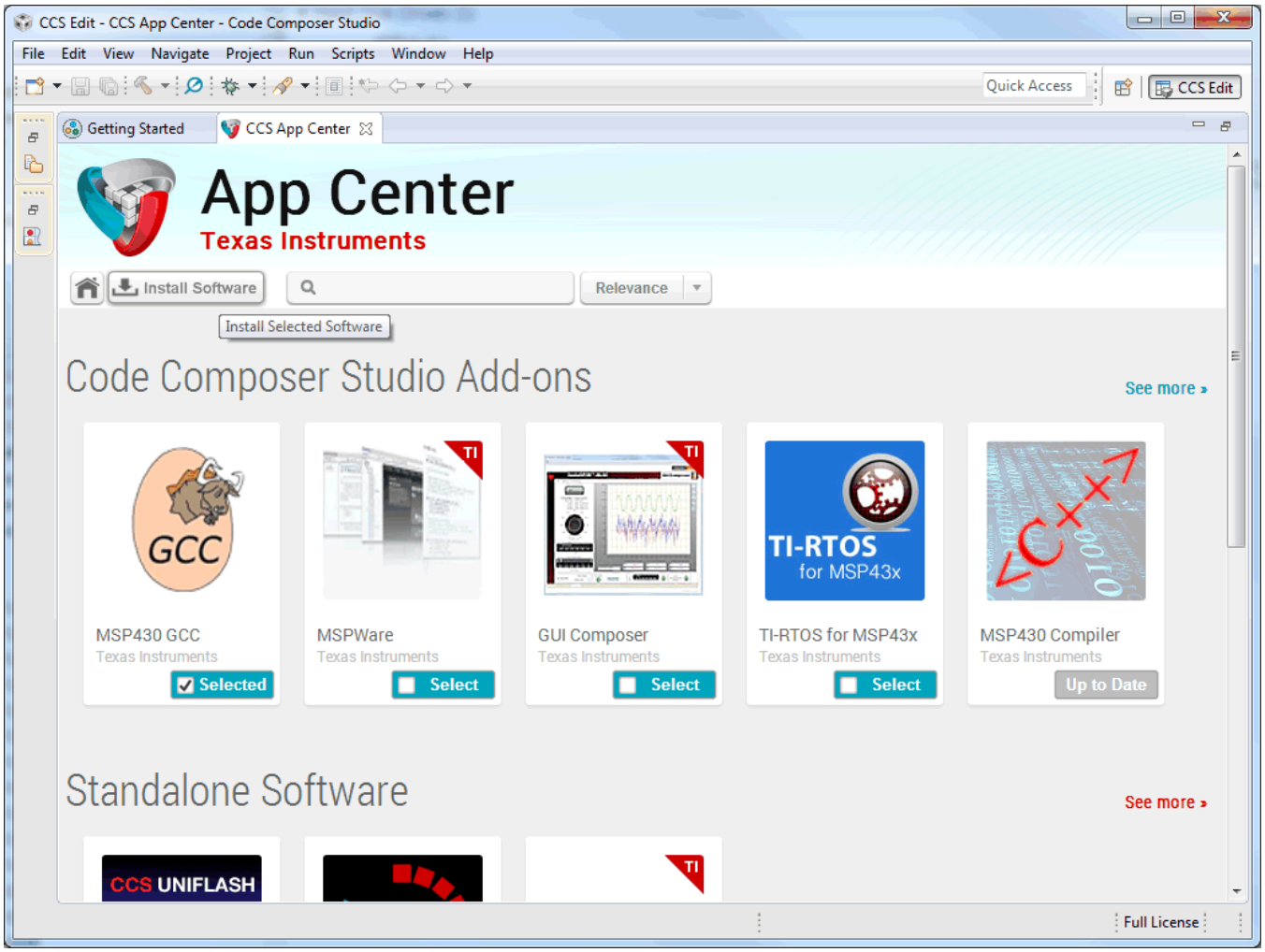

Figure 3. Installing MSP430 GCC Through CCS Apps Center

3. After installation is complete, the GCC compiler tools are in the following directory within the CCS installation: ccsv6\tools\compiler\gcc_msp430_x.x.x (where xxx denotes the version number).

2.2 Installing MSP430 GCC as Stand-Alone Package

The MSP430 GCC full stand-alone package can be downloaded from the TI website for all supported operating systems. The MSP430 GCC stand-alone package contains the compiler, device support files, debug stack, and USB drivers.

To install the package:

1. Download the corresponding package installer and run it (see Figure 4).

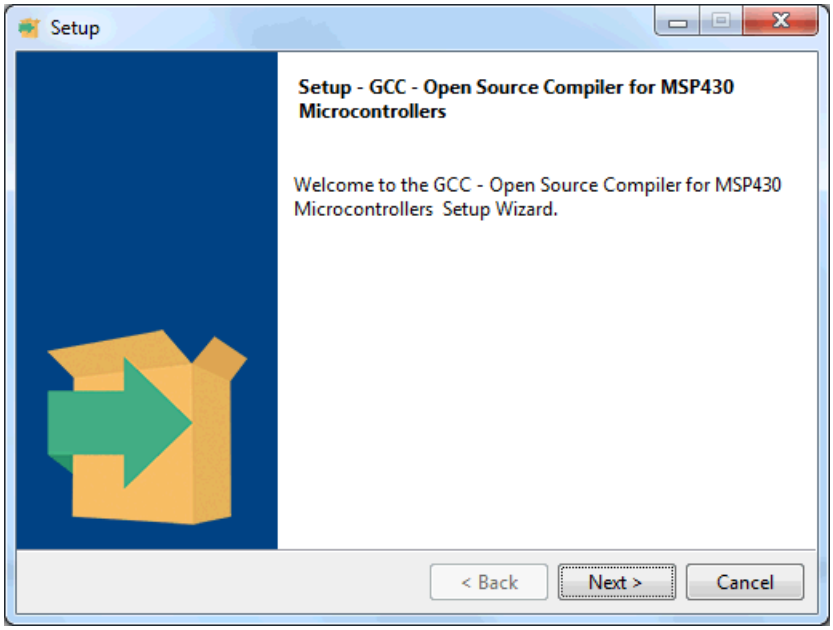

Figure 4. MSP430 GCC Stand-Alone Package Installer

2. Select the install directory and click **Next** (see Figure 5).

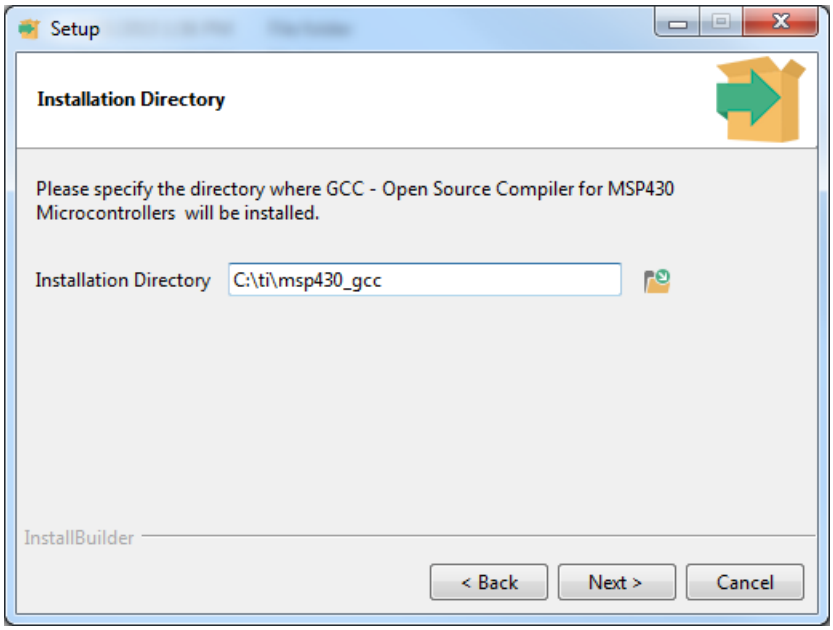

Figure 5. MSP430 GCC Stand-Alone Package Installation Directory

NOTE: For the Linux installer, apply **sudo chmod +x** <installer> before executing the package.

3 Using MSP430 GCC Within CCS

3.1 Create New Project

This section describes the step-by-step instructions to create an assembly or C project from scratch and to download and run an application on the MSP430 MCU using the MSP430 GCC compiler. Also, the CCS Help presents a more detailed information of the process.

1. Start CCS (**Start** → **All Programs** → **Texas Instruments** → **Code Composer Studio** → **Code Composer Studio**).
2. Create a new project (**File** → **New** → **CCS Project**). Select the appropriate MSP430 device variant in the Target field and enter the name for the project.
3. Select **GNU v7.3.0.9 (Mitto Systems Limited)** for Compiler version (or any newer version).
4. In the Project template and examples section, select **Empty Project (with main.c)**. For assembly-only projects, select **Empty Project**.

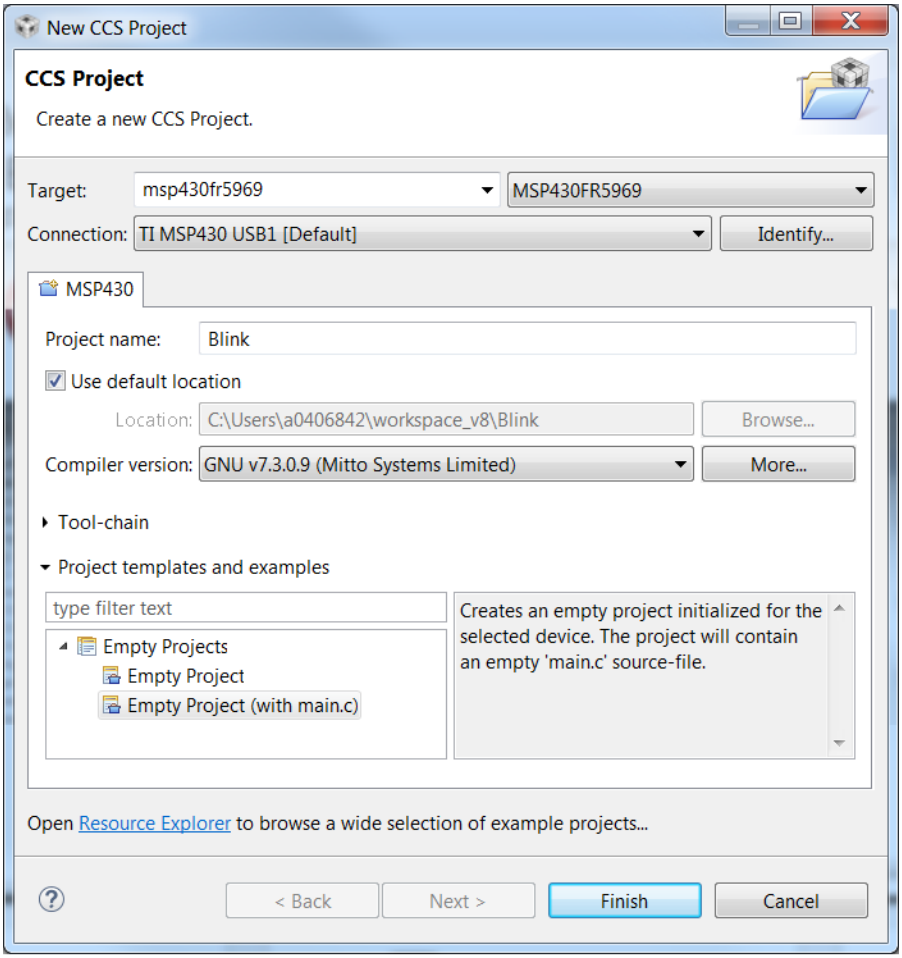

Figure 6. Creating New CCS Project Using MSP430 GCC

5. If you are using a USB Flash Emulation Tool such as the MSP-FET, MSP-FET430UIF, eZ-FET, or the eZ430 Development Tool, they should be already configured by default.
6. For C projects, the setup is complete now.
7. Click **Finish** to create a new project that is then visible in the Project Explorer view.

   Notice that the project contains a .ld file (appropriate for the target selected). This is the linker script that contains the memory layout and section allocation. This file is the equivalent of the TI linker command file (.cmd) used by TI MSP430 Compiler and Linker.
8. Enter the program code into the main.c file.

To use an existing source file for the project, click **Project → Add Files...** and browse to the file of interest. Single click on the file and click **Open** or double-click on the file name to complete the addition of it into the project folder.

Now add the necessary source files to the project and build. Similar to TI tools, additional compiler and linker options can be set from **Project Properties**.

9. Build the project (**Project → Build Project**).

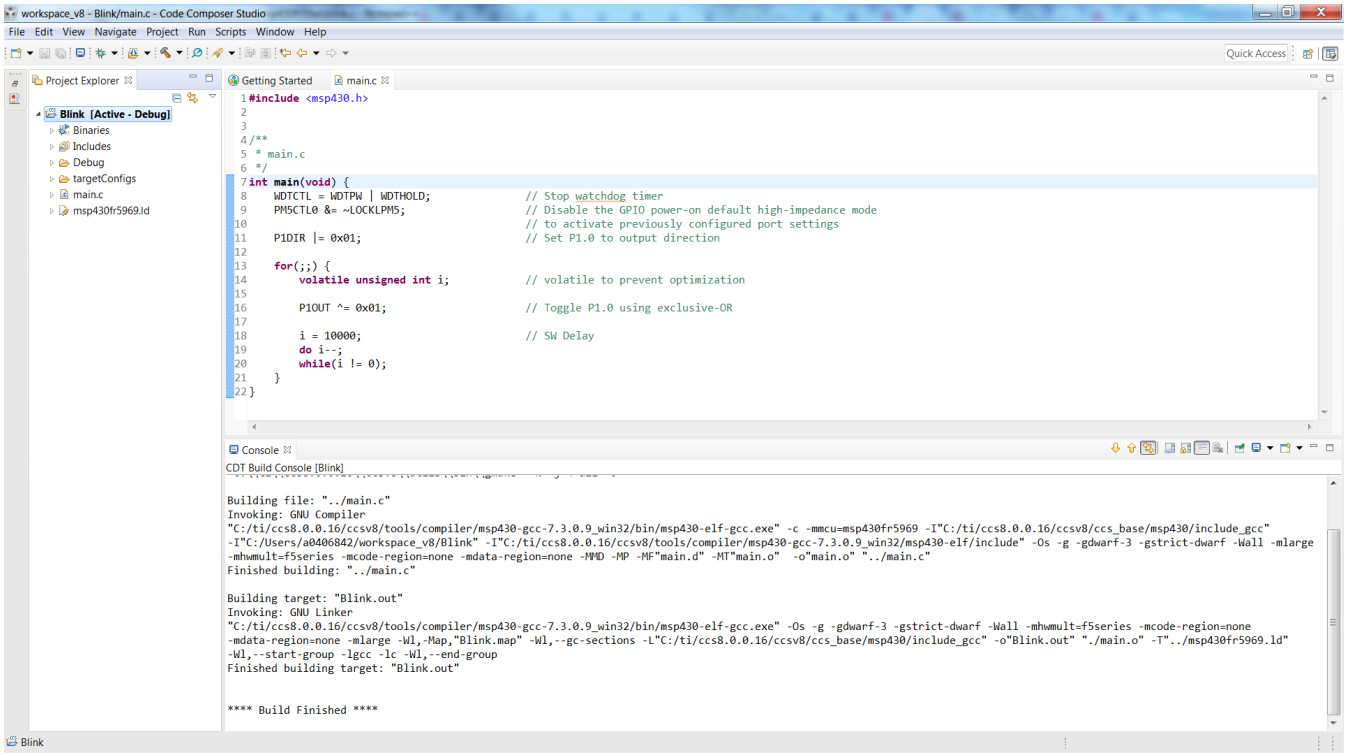

Figure 7. CCS Project Using MSP430 GCC

10. Debug the application (**Run → Debug (F11)**). This starts the debugger, which gains control of the target, erases the target memory, programs the target memory with the application, and resets the target.
11. Click **Run → Resume (F8)** to start the application.
12. Click **Run → Terminate** to stop the application and to exit the debugger. CCS automatically returns to the C/C++ view (code editor).

3.2 Debug Using MSP-FET, MSPFET430UIF, eZ-FET, eZ430

MSP430 devices can be debugged in CCS using MSP-FET, MSPFET430UIF, eZ-FET, and eZ430 debuggers. For more details, refer to the *Code Composer Studio™ v8.x for MSP430™ User's Guide*.

3.3 Build Options for MSP430 GCC

The settings required to configure the GCC are numerous and detailed and are not all described here. Most projects can be compiled and debugged with default factory settings.

To access the project settings for the active project, click **Project → Properties**.

The following project settings are common:

- Specify the target device for debug session (**Project → Properties → General → Device → Variant**). The corresponding Linker Command File and Runtime Support Library are selected automatically.
- To more easily debug a C project, disable optimization (-O0) or use -Og, which enables only those optimizations that don't interfere with debugging. The -Og option reduces code size and improves

performance compared to -O0.

- Specify the search paths for the C preprocessor (**Project → Properties → Build → GNU Compiler → Directories → Include Paths (-I)**).
- Specify the search paths for any libraries being used (**Project → Properties → Build → GNU Linker → Libraries → Library search path (-L, --library-path)**).
- Specify the debugger interface (**Project → Properties → General → Device → Connection**). Select TI MSP430 USBx for the USB interface.
- Enable the erasure of the Main and Information memories before object code download (**Project → Properties → Debug → MSP430 Properties → Download Options → Erase Main and Information Memory**).
- To ensure proper stand-alone operation, select Hardware Breakpoints (**Project → Properties → Debug → MSP430 Properties**). If Software Breakpoints are enabled (**Project → Properties → Debug → Misc/Other Options → Allow software breakpoints to be used**), ensure proper termination of each debug session while the target is connected. Otherwise, the target may not work as expected stand-alone as the application on the device still contains the software breakpoint instructions.

3.3.1 GNU Compiler

Figure 8 shows the MSP430 GCC settings window.

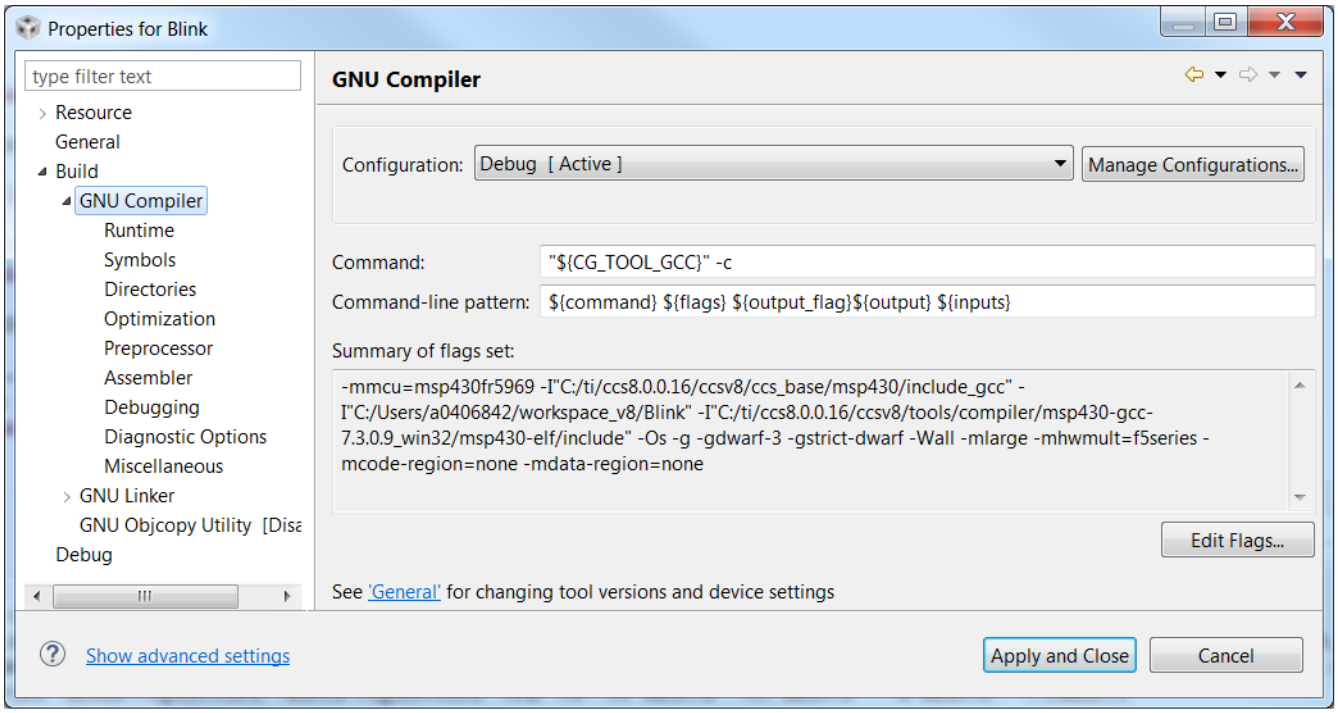

Figure 8. MSP430 GCC Settings

Table 2 describes the options that are available for MSP430 GCC Settings.

Table 2. MSP430 GCC Settings

Option	Description
Command	Compiler location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the compiler is called. Displays all the flags passed to the compiler.

3.3.2 GNU Compiler: Runtime

Figure 9 shows the MSP430 GCC Runtime settings window.

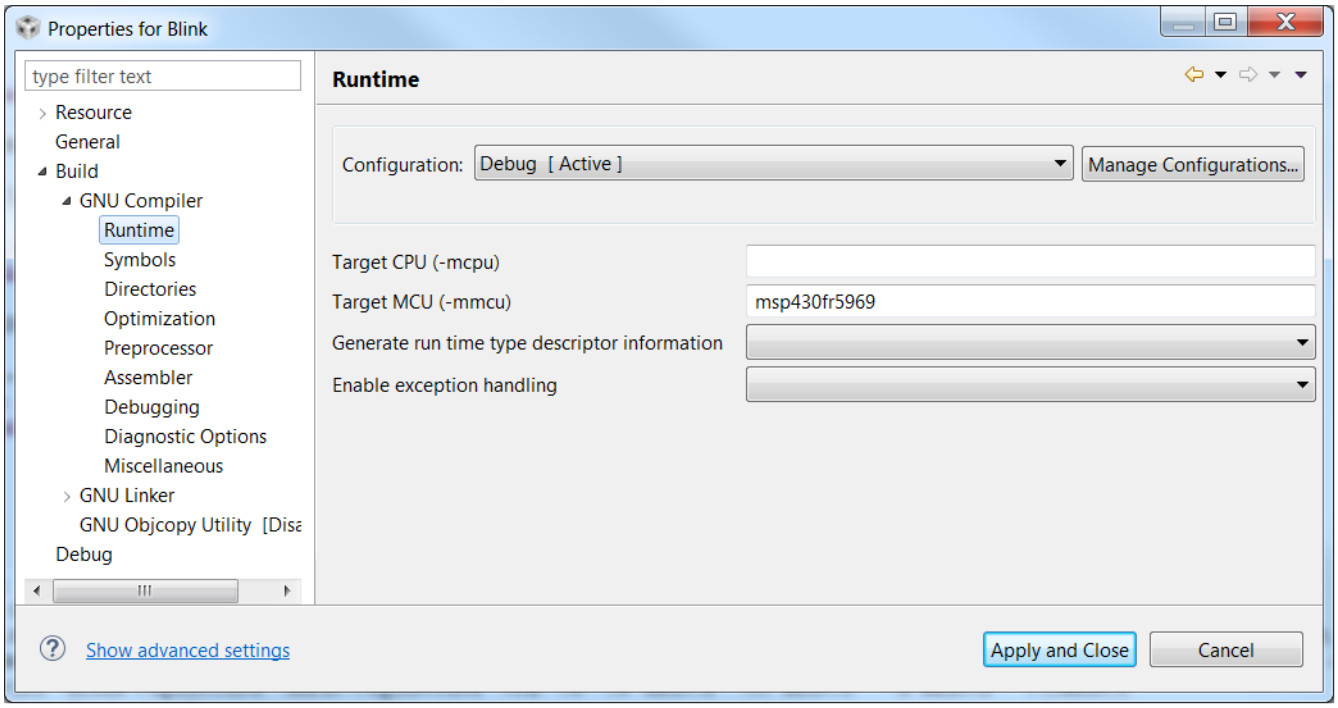

Figure 9. MSP430 GCC Settings: Runtime

Table 3 describes the options that are available for MSP430 GCC Runtime settings.

Table 3. MSP430 GCC Settings: Runtime

Option	Description
Target CPU (-mcpu)	Specifies the Instruction Set Architecture (ISA) to use. Accepted values are msp430, msp430x, and msp430xv2. This option is deprecated. The '-mmcu=' option should be used to select the ISA.
Target MCU (-mmcu)	Select the MCU to target. This is used to create a C preprocessor symbol based on the MCU name, converted to upper case and prefixed and postfixed with__. This in turn is used by the msp430.h header file to select an MCU-specific supplementary header file. The option also sets the ISA to use. If the MCU name is one that is known to only support the 430 ISA then that is selected, otherwise the 430X ISA is selected. A generic MCU name of msp430 can also be used to select the 430 ISA. Similarly the generic msp430x MCU name selects the 430X ISA. In addition, an MCU-specific linker script is added to the linker command line. The script's name is the name of the MCU with ".ld" appended. Thus, specifying '-mmcu=xxx' on the gcc command line defines the C preprocessor symbol __XXX__ and causes the linker to search for a script called 'xxx.ld'. This option is also passed to the assembler.
Generate run time type descriptor information	Enable or disable generation of information about every class with virtual functions for use by the C++ runtime type identification features. • On (-frtti) • Off (-fno-rtti)
Enable exception handling	Enable or disable exception handling. Generates extra code needed to propagate exceptions. • On (-fexceptions) • Off (-fno-exceptions)

3.3.3 GNU Compiler: Symbols

Figure 10 shows the MSP430 GCC Symbols settings window.

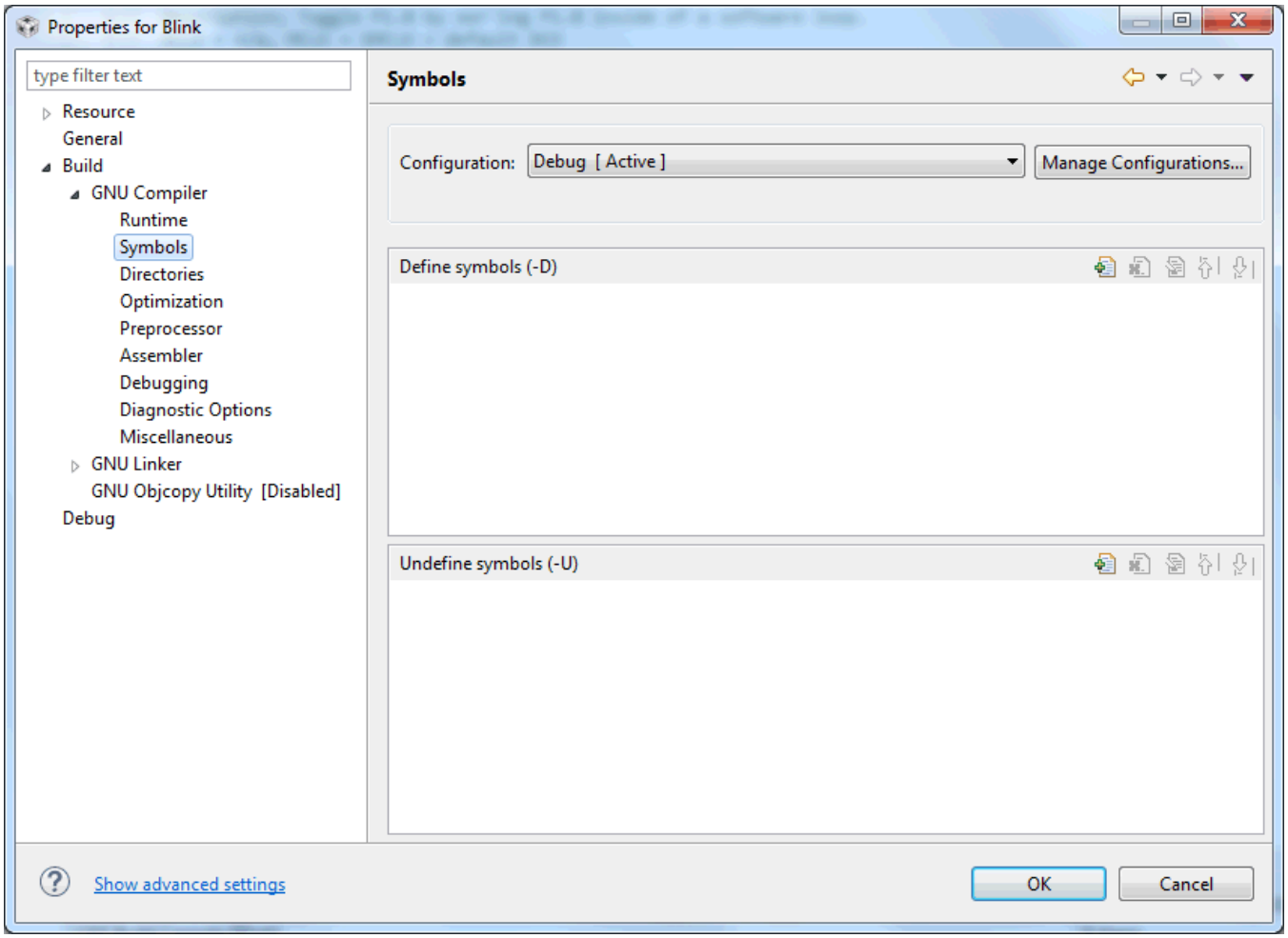

Figure 10. MSP430 GCC Settings: Symbols

Table 4 describes the options that are available for MSP430 GCC Symbols settings.

Table 4. MSP430 GCC Settings: Symbols

Option	Description
Define symbols (-D)	-D name Predefine name as a macro. -D name=definition Predefine name as a macro, with definition 1.
Undefine symbols (-U)	-U name Cancel any previous definition of name, either built-in or provided with a -D option.

3.3.4 GNU Compiler: Directories

Figure 11 shows the MSP430 GCC Directories settings window.

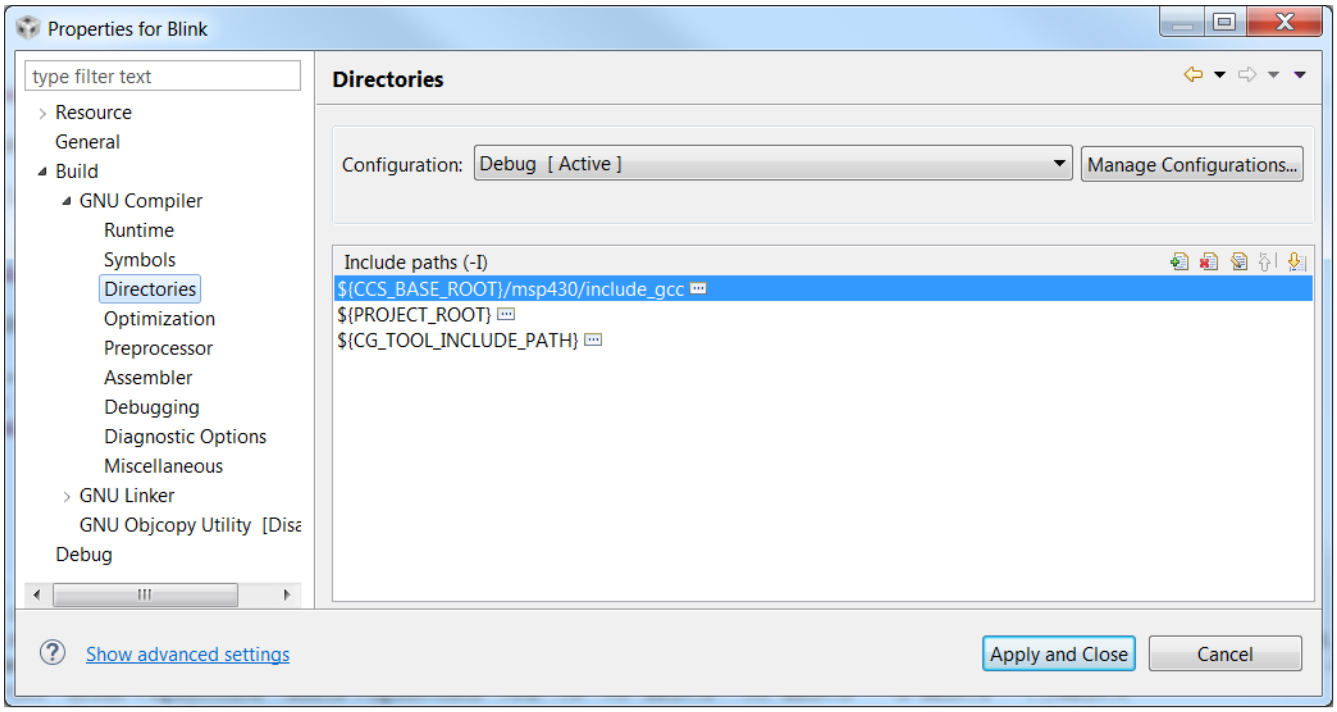

Figure 11. MSP430 GCC Settings: Directories

Table 5 describes the options that are available for MSP430 GCC Directories settings.

Table 5. MSP430 GCC Settings: Directories

Option	Description
Include paths (-I)	Add the directory to the list of directories to be searched for header files.

3.3.5 GNU Compiler: Optimization

Figure 12 shows the MSP430 GCC Optimization settings window.

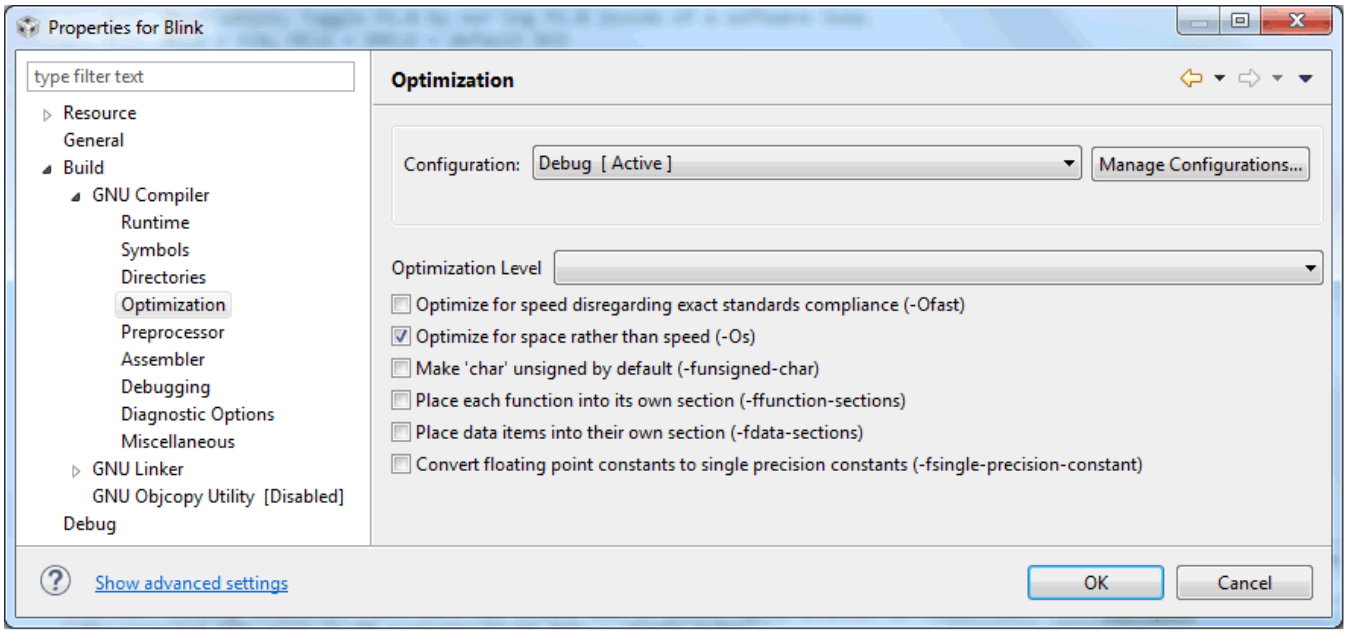

Figure 12. MSP430 GCC Settings: Optimization

Table 6 describes the options that are available for MSP430 GCC Optimization settings.

Table 6. MSP430 GCC Settings: Optimization

Option	Description
Optimization Level	Specifies the optimizations that the compiler applies to the generated object code. The options available are: • None (O0): Disable optimizations. This setting is equivalent to specifying the -O0 command-line option. The compiler generates unoptimized linear assembly language code. • Optimize (O1): The compiler performs all targets independent (that is, nonparallelized) optimizations, such as function inlining. This setting is equivalent to specifying the -O1 command-line option. The compiler omits all target-specific optimizations and generates linear assembly language code. • Optimize more (O2): The compiler performs all optimizations (both target-independent and target-specific). This setting is equivalent to specifying the -O2 command-line option. The compiler outputs optimized nonlinear parallelized assembly language code. • Optimize most (O3): The compiler performs all the level 2 optimizations, then the low-level optimizer performs global-algorithm register allocation. This setting is equivalent to specifying the -O3 command-line option. At this optimization level, the compiler generates code that is usually faster than the code generated from level 2 optimizations. • Optimize for space rather than speed (-Os): Enables all -O2 optimizations that do not typically increase code size. The -Os option also performs further optimizations designed to reduce code size. • Optimize for speed disregarding exact standards compliance (-Ofast): Enables all -O3 optimizations. The -Ofast option also enables optimizations that are not valid for all standard-compliant programs, such as -ffast-math.
Make 'char' unsigned by default (-funsigned-char)	Enable this option to ensure that the char is signed.
Place each function into its own section (-ffunction-sections)	Enable this option to place each function in its own section in the output file.
Place data items into their own section (-fdata-sections)	Enable this option to place each data item in its own section in the output file.

Table 6. MSP430 GCC Settings: Optimization (continued)

Option	Description
Convert floating point constants to single precision constants (-fsingle-precision-constant)	Treat floating-point constants as single precision instead of implicitly converting them to double-precision constants.

NOTE: Use the -ffunction-sections and -fdata-sections options in conjunction with the --gc-sections linker option to reduce code size by allowing the linker to remove unused sections.

3.3.6 GNU Compiler: Preprocessor

Figure 13 shows the MSP430 GCC Preprocessor settings window.

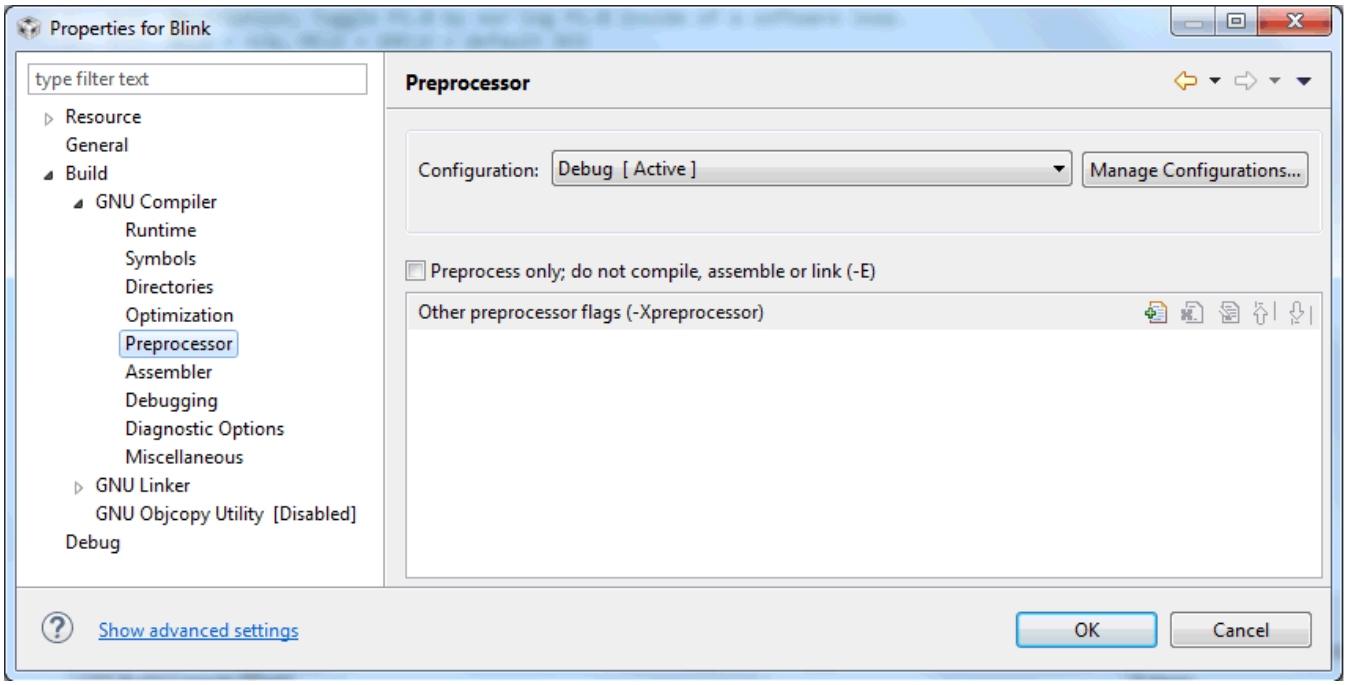

Figure 13. MSP430 GCC Settings: Preprocessor

Table 7 describes the options that are available for MSP430 GCC Preprocessor settings.

Table 7. MSP430 GCC Settings: Preprocessor

Option	Description
Preprocess only; do not compile, assemble or link (-E)	Enable this option to preprocess only without compiling or assembling or linking.
Other preprocessor flags (-Xpreprocessor)	Use this to supply system-specific preprocessor options that GCC does not recognize. To pass an option that takes an argument, use -Xpreprocessor twice, once for the option and once for the argument.

3.3.7 GNU Compiler: Assembler

Figure 14 shows the MSP430 GCC Assembler settings window.

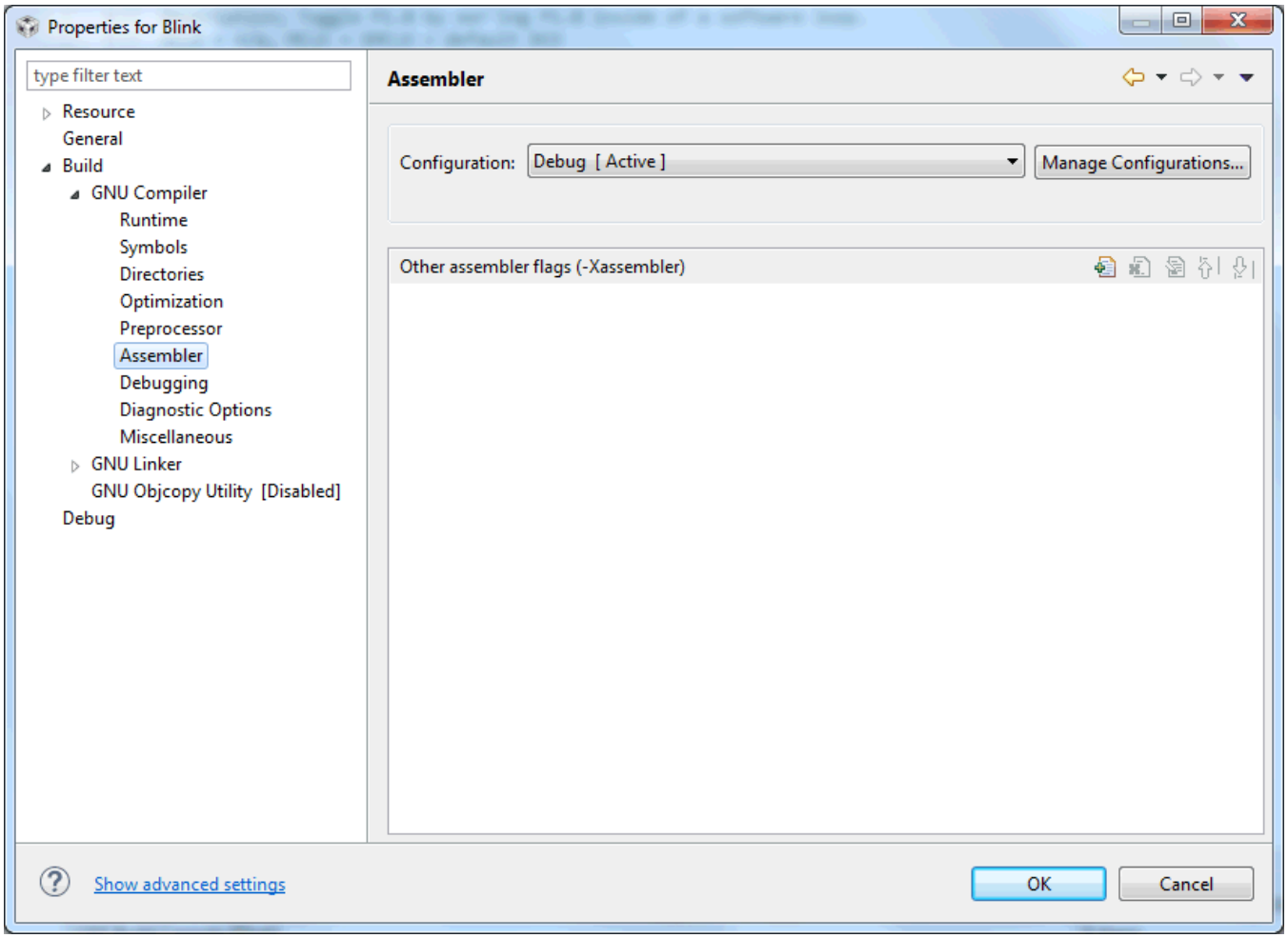

Figure 14. MSP430 GCC Settings: Assembler

Table 8 describes the options that are available for MSP430 GCC Assembler settings.

Table 8. MSP430 GCC Settings: Assembler

Option	Description
Other assembler flags (-Xassembler)	Specifies individual flag based on the user requirements.

3.3.8 GNU Compiler: Debugging

Figure 15 shows the MSP430 GCC Debugging settings window.

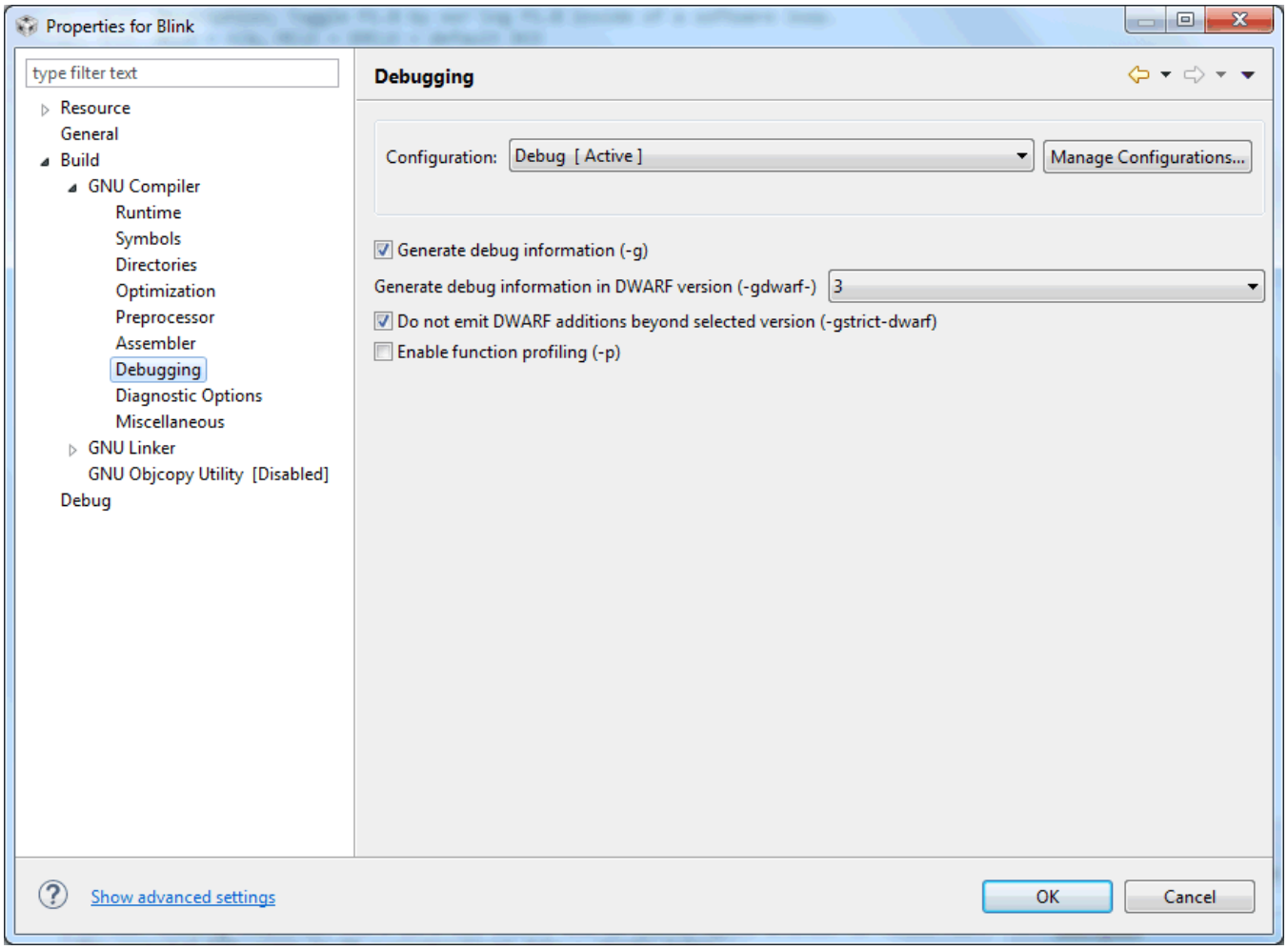

Figure 15. MSP430 GCC Settings: Debugging

Table 9 describes the options that are available for MSP430 GCC Debugging settings.

Table 9. MSP430 GCC Settings: Debugging

Option	Description
Generate debug information (-g)	Produce debugging information. This information is required by the GDB debugger.
Generate debug information in DWARF version (-gdwarf-)	Produce debugging information in DWARF format (if that is supported). The value of version may be 2, 3 or 4; the default version for most targets is 4.
Do not emit DWARF additions beyond selected version (-gstrict-dwarf)	Disallow using extensions of later DWARF standard version than selected with -gdwarf-version. On most targets using nonconflicting DWARF extensions from later standard versions is allowed.
Enable function profiling (-p)	Generate extra code to write profile information suitable for the analysis program. This option is required when compiling source files for which data is needed, and it is also required when linking.

3.3.9 GNU Compiler: Diagnostic Options

Figure 16 shows the MSP430 GCC Diagnostic Options settings window.

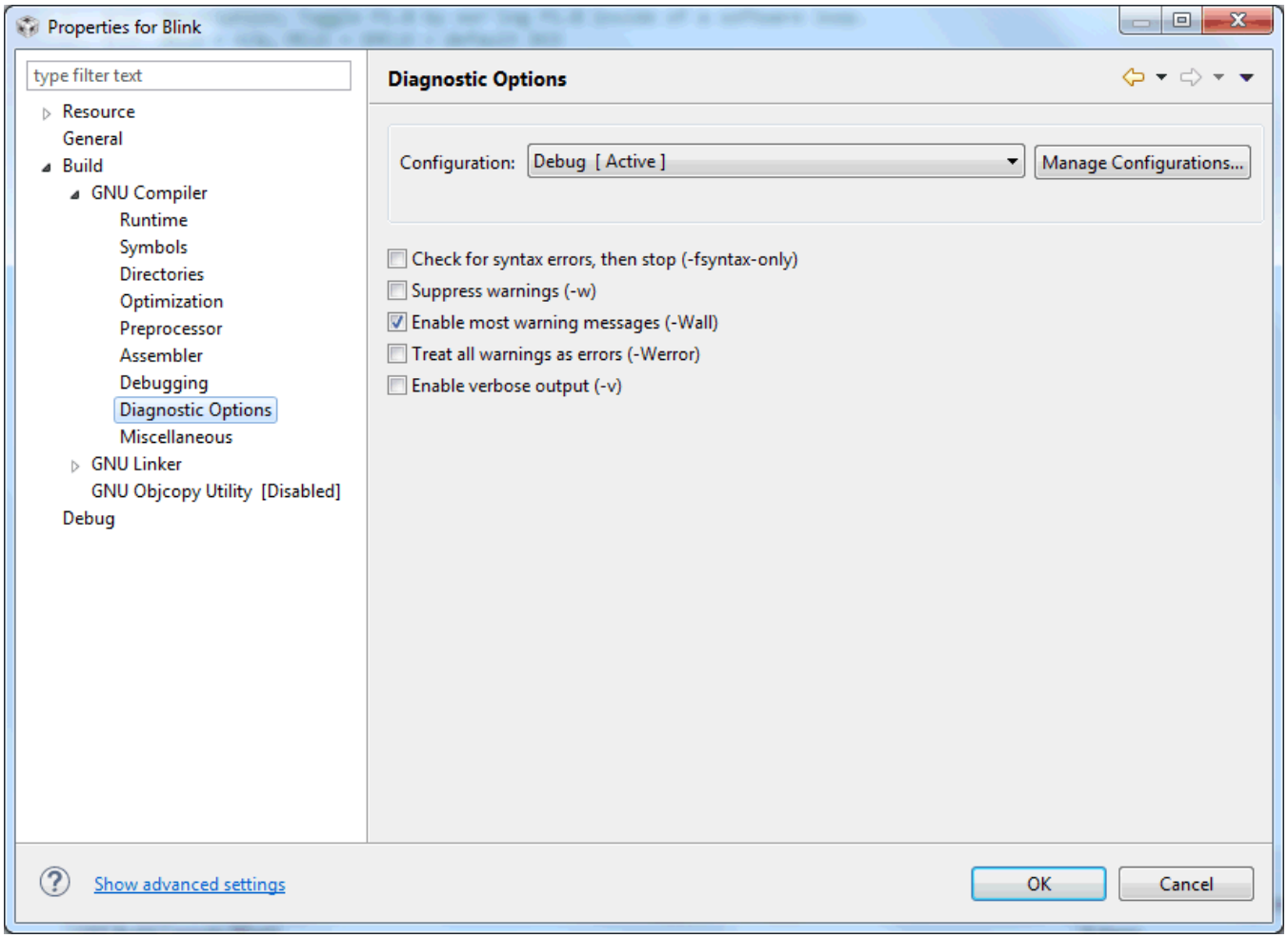

Figure 16. MSP430 GCC Settings: Diagnostic Options

Table 10 describes the options that are available for MSP430 GCC Diagnostic Options settings.

Table 10. MSP430 GCC Settings: Diagnostic Options

Option	Description
Check for syntax errors, then stop (-fsyntax-only)	Enable this option to check the syntax of the code and report any errors.
Suppress warnings (-w)	Inhibit all warning messages.
Enable most warning messages (-Wall)	Enable this option to enable all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
Treat all warnings as errors (-Werror)	Enable this option to make all warnings into hard errors. Source code that triggers warnings is rejected.
Enable verbose output (-v)	Enable this option for the IDE to show each command line that it passes to the shell, along with all progress, error, warning, and informational messages that the tools emits. This setting is equivalent to specifying the –v command-line option. By default, this checkbox is clear. The IDE displays only error messages that the compiler emits. The IDE suppresses warning and informational messages.

3.3.10 GNU Compiler: Miscellaneous

Figure 17 shows the MSP430 GCC Miscellaneous settings window.

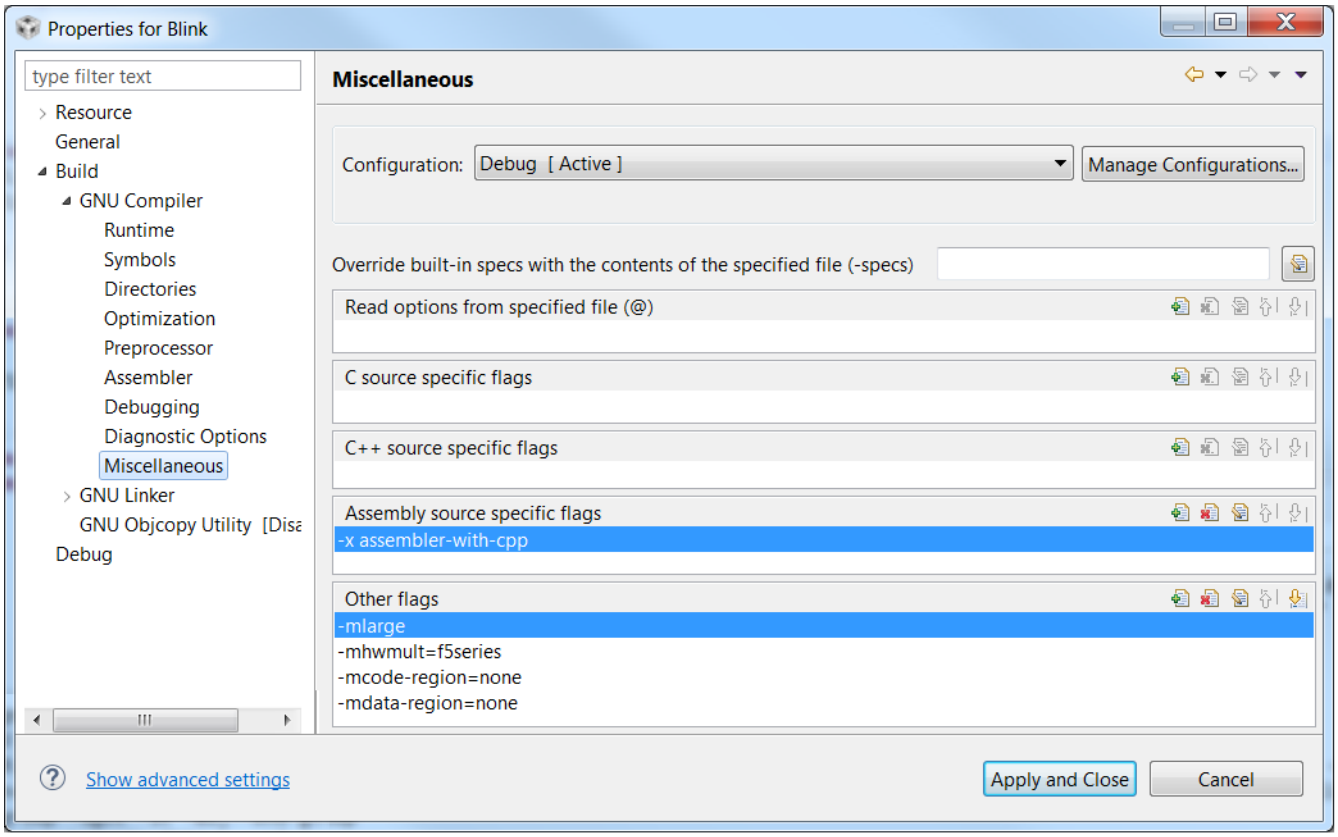

Figure 17. MSP430 GCC Settings: Miscellaneous

Table 11 describes the options that are available for MSP430 GCC Miscellaneous settings.

Table 11. MSP430 GCC Settings: Miscellaneous

Option	Description
Override built-in specs with the contents of the specified file (-specs)	The spec strings built into GCC can be overridden by using the -specs= command-line switch to specify a spec file.
Other flags	-mlarge Use large-model addressing (20-bit pointers, 32-bit size_t). -mcode-region=none -mdata-region=none The MSP430 compiler has the ability to automatically distribute code and data between low memory (addresses below 64K) and high memory. This only applies to parts that actually have both memory regions and only if the linker script for the part has been specifically set up to support this feature. See Table 21 for more information.

3.3.11 GNU Linker

Figure 18 shows the MSP430 GCC Linker settings window.

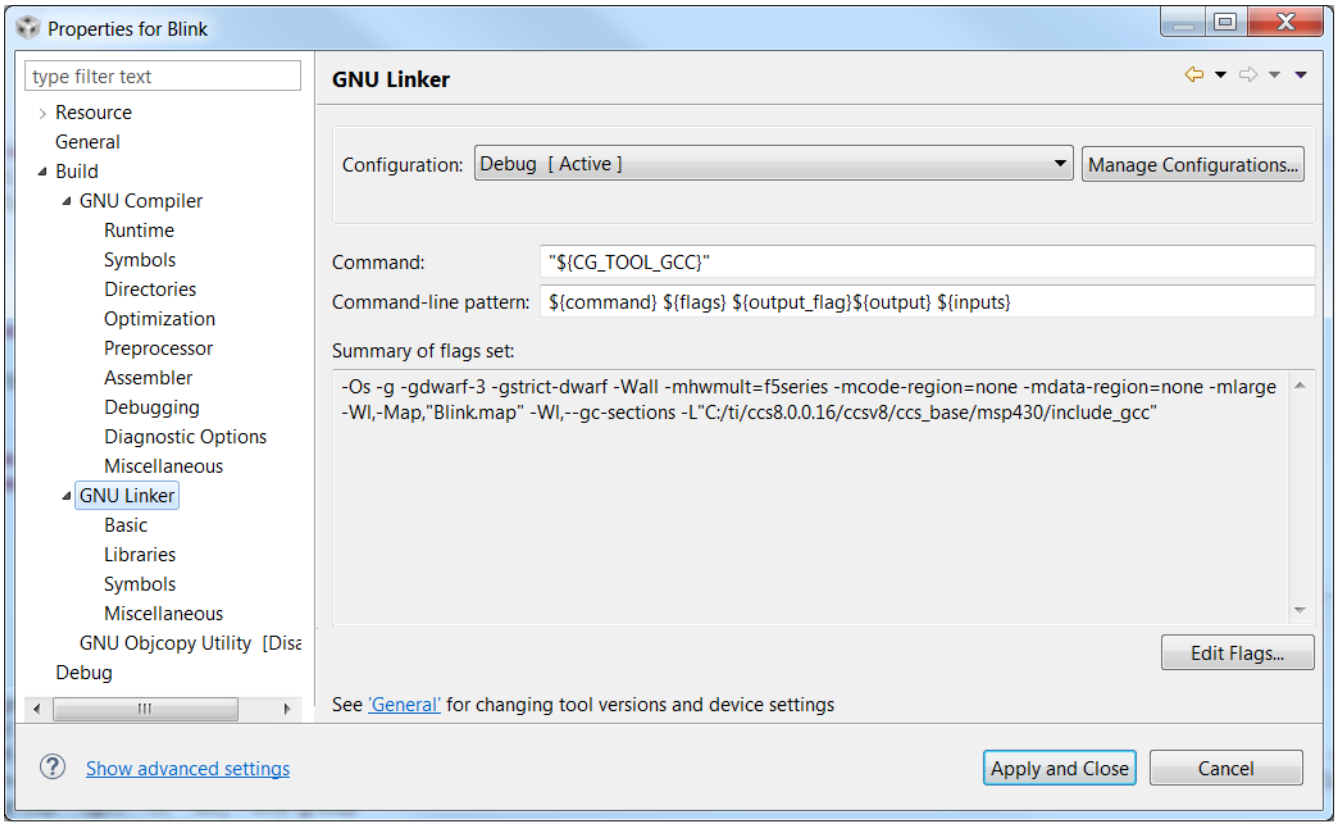

Figure 18. MSP430 GCC Linker Settings

Table 12 describes the options that are available for MSP430 GCC Linker settings.

Table 12. MSP430 GCC Linker Settings

Option	Description
Command	Linker location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the compiler is called. Displays all the flags passed to the linker.

3.3.12 GNU Linker: Basic

Figure 19 shows the MSP430 GCC Linker Basic settings window.

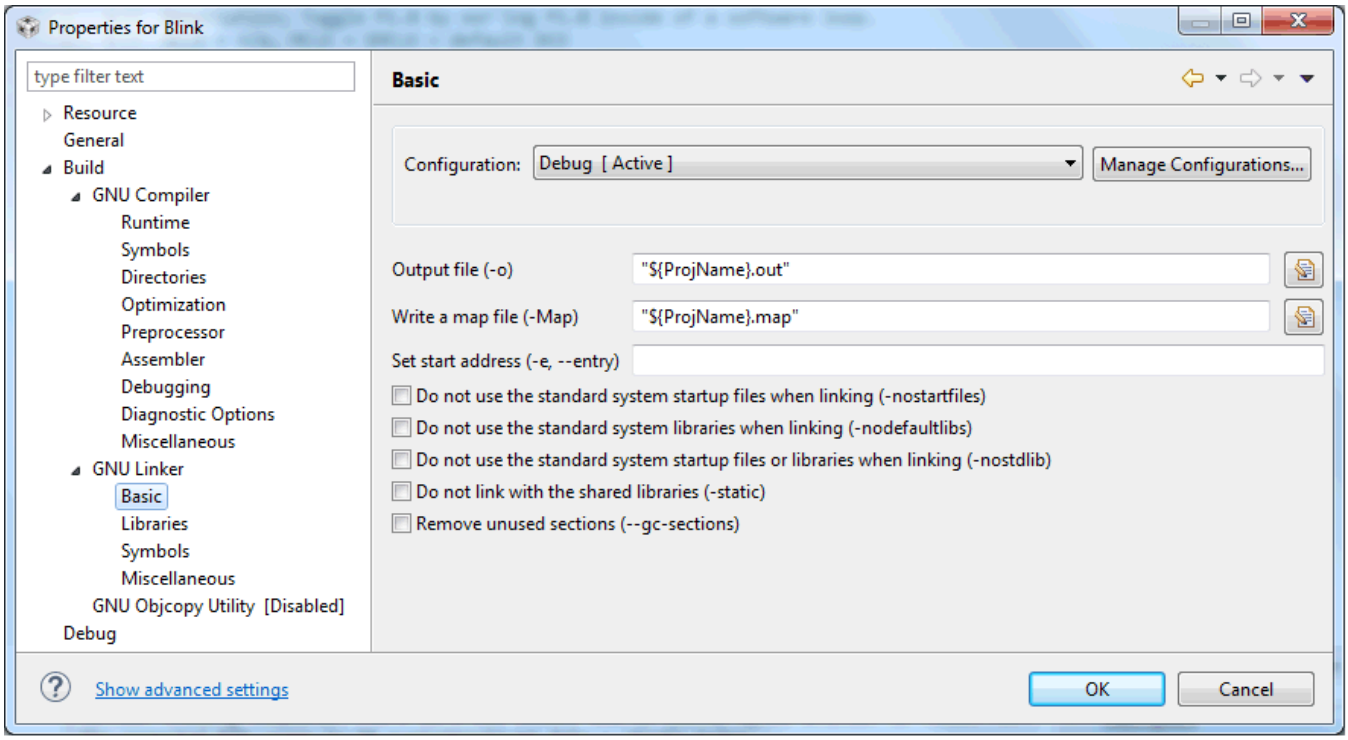

Figure 19. MSP430 GCC Linker Basic Settings

Table 13 describes the options that are available for MSP430 GCC Linker Basic settings.

Table 13. MSP430 GCC Linker Basic Settings

Option	Description
Output file (-o)	-o output Use output as the name for the file produced by ld; if this option is not specified, the name 'a.out' is used by default. The script command OUTPUT can also specify the output file name.
Write a map file (-Map)	Print to the file mapfile a link map, which contains diagnostic information about where symbols are mapped by ld and information on global common storage allocation.
Set start address (-e, --entry)	Use entry as the explicit symbol for beginning execution of the program, rather than the default entry point.
Do not use the standard system startup files when linking (-nostartfiles)	Do not use the standard system startup files when linking. The standard system libraries are used unless -nostdlib or -nodefaultlibs is used.
Do not use the standard system libraries when linking (-nodefaultlibs)	Do not use the standard system libraries when linking. Only the specified libraries are passed to the linker, and options specifying linkage of the system libraries, such as -static-libgcc or -shared-libgcc, are ignored. The standard startup files are used unless -nostartfiles is used. The compiler may generate calls to memcmp, memset, memcpy, and memmove. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.
Do not use the standard system startup files or libraries when linking (-nostdlib)	Do not use the standard system startup files or libraries when linking.
Do not link with the shared libraries (-static)	On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.
Remove unused sections (--gc-sections)	Enable garbage collection of unused input sections. It is ignored on targets that do not support this option.

3.3.13 GNU Linker: Libraries

Figure 20 shows the MSP430 GCC Linker Libraries settings window.

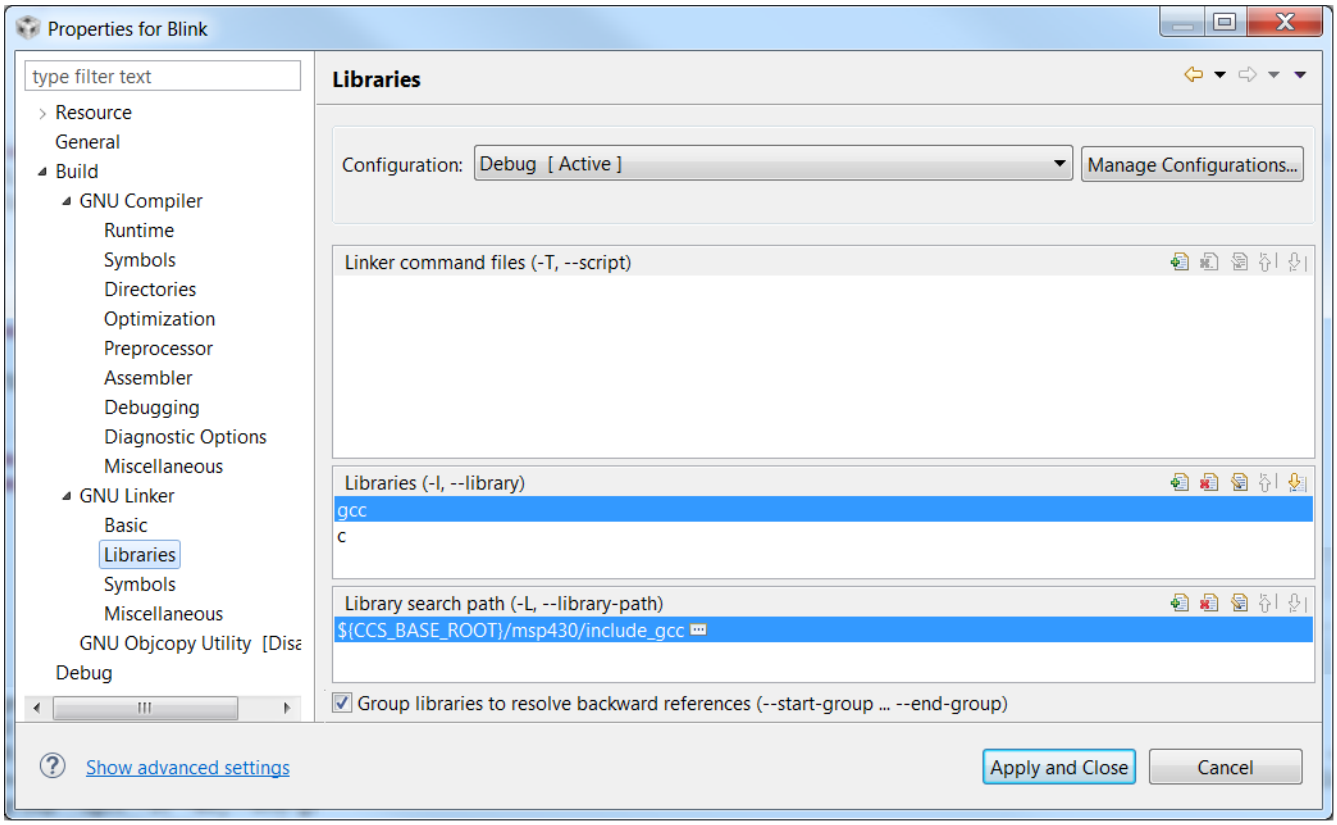

Figure 20. MSP430 GCC Linker Libraries Settings

Table 14 describes the options that are available for MSP430 GCC Linker Libraries settings.

Table 14. MSP430 GCC Linker Libraries Settings

Option	Description
Linker command files (-T, --script)	-T commandfile Read link commands from the file command file.
Libraries (-l, --library)	-l library Search the library named library when linking.
Library search path (-L, --library-path)	-L searchdir Add path searchdir to the list of paths that ld will search for archive libraries and ld control scripts.

3.3.14 GNU Linker: Symbols

Figure 21 shows the MSP430 GCC Linker Symbols settings window.

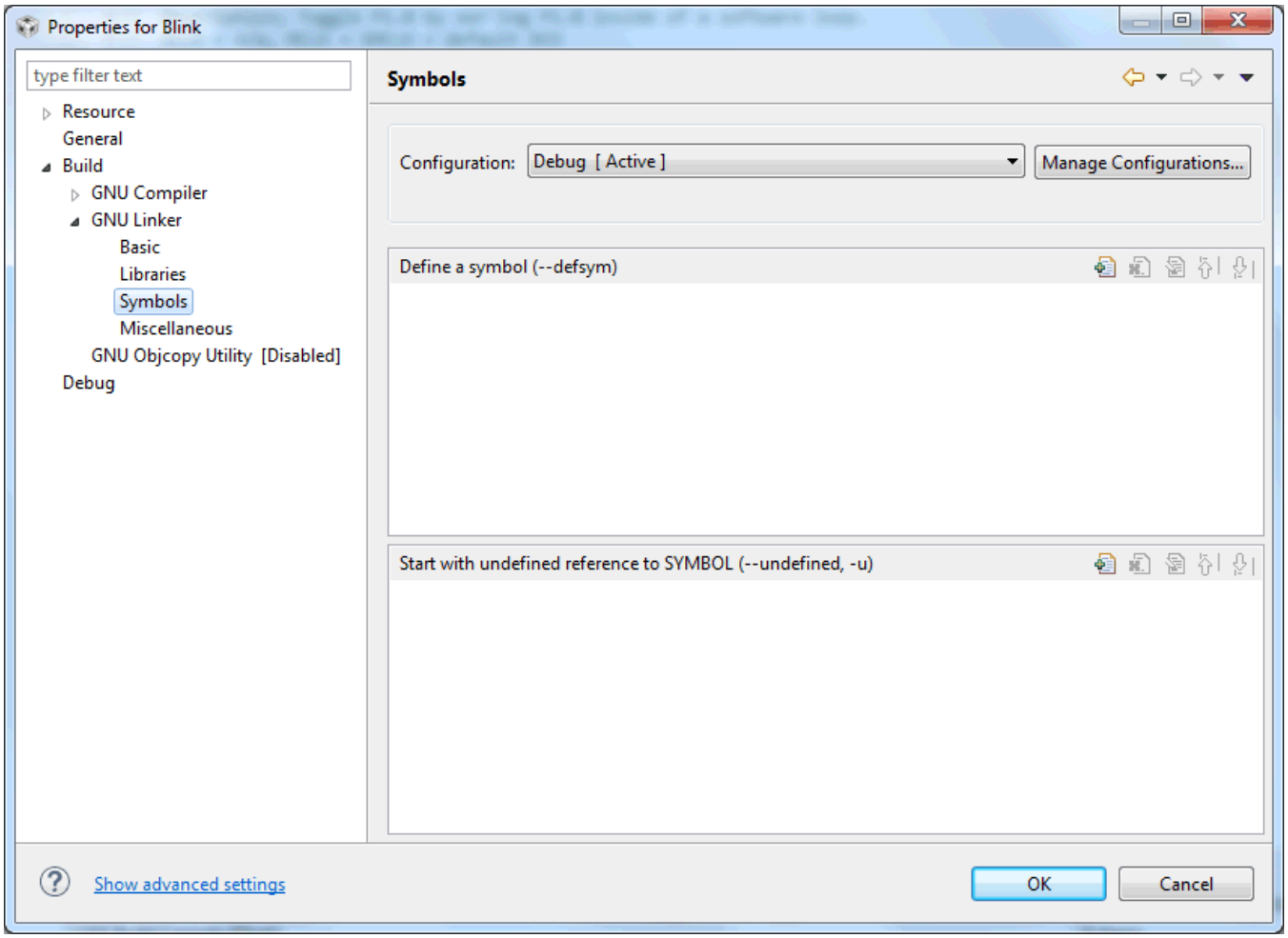

Figure 21. MSP430 GCC Linker Symbols Settings

Table 15 describes the options that are available for MSP430 GCC Linker Symbols settings.

Table 15. MSP430 GCC Linker Symbols Settings

Option	Description
Define a symbol (--defsym)	-defsym symbol=expression Create a global symbol in the output file, with the absolute address given by expression.
Start with undefined reference to SYMBOL (--undefined, -u)	Force symbol to be entered in the output file as an undefined symbol

3.3.15 GNU Linker: Miscellaneous

Figure 22 shows the MSP430 GCC Linker Miscellaneous settings window.

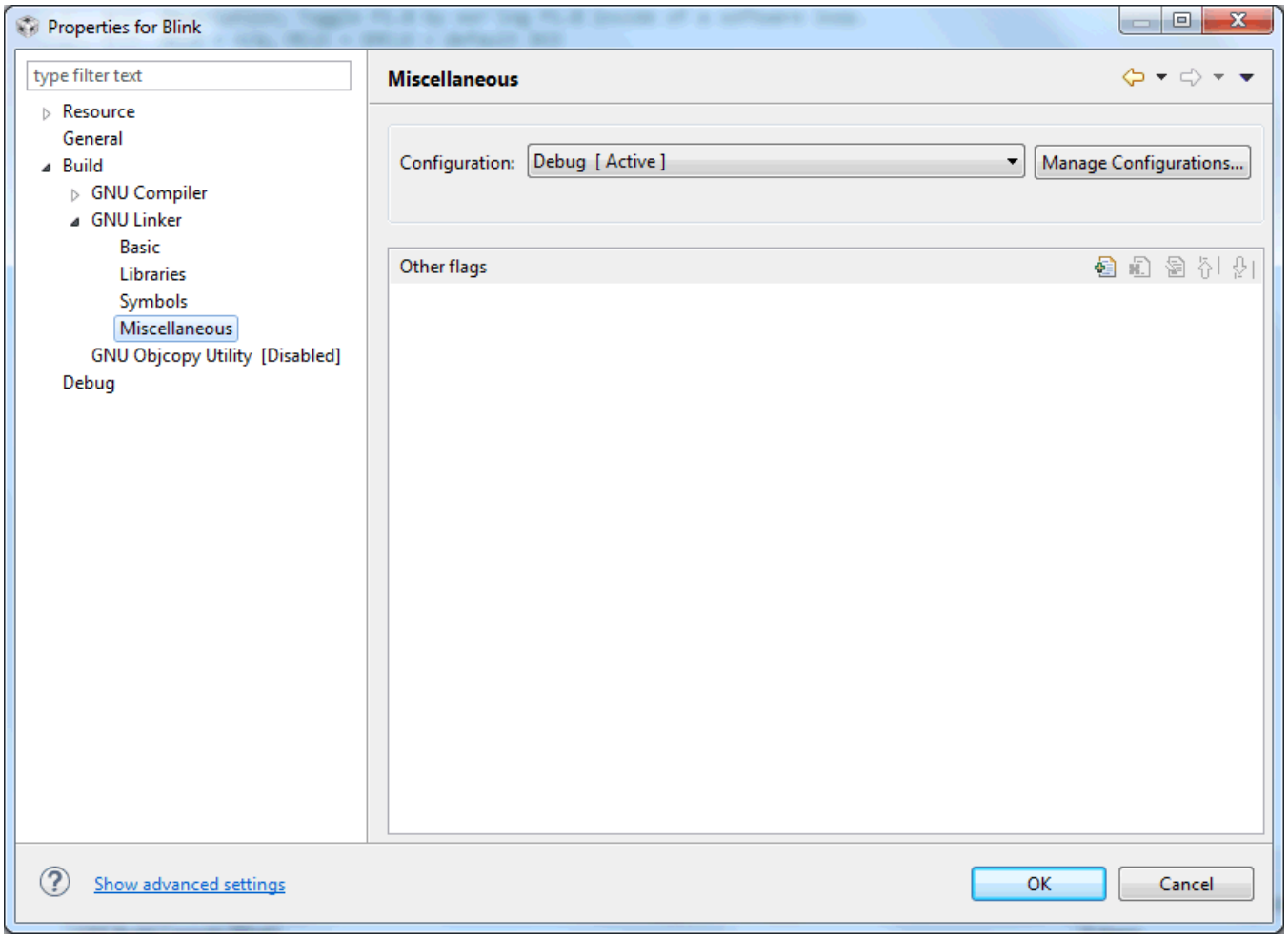

Figure 22. MSP430 GCC Linker Miscellaneous Settings

Table 16 describes the options that are available for MSP430 GCC Linker Miscellaneous settings.

Table 16. MSP430 GCC Linker Miscellaneous Settings

Option	Description
Other flags	Specifies individual flags based on the user requirements.

3.3.16 GNU Objcopy Utility

Figure 23 shows the MSP430 GCC GNU Objcopy Utility settings window.

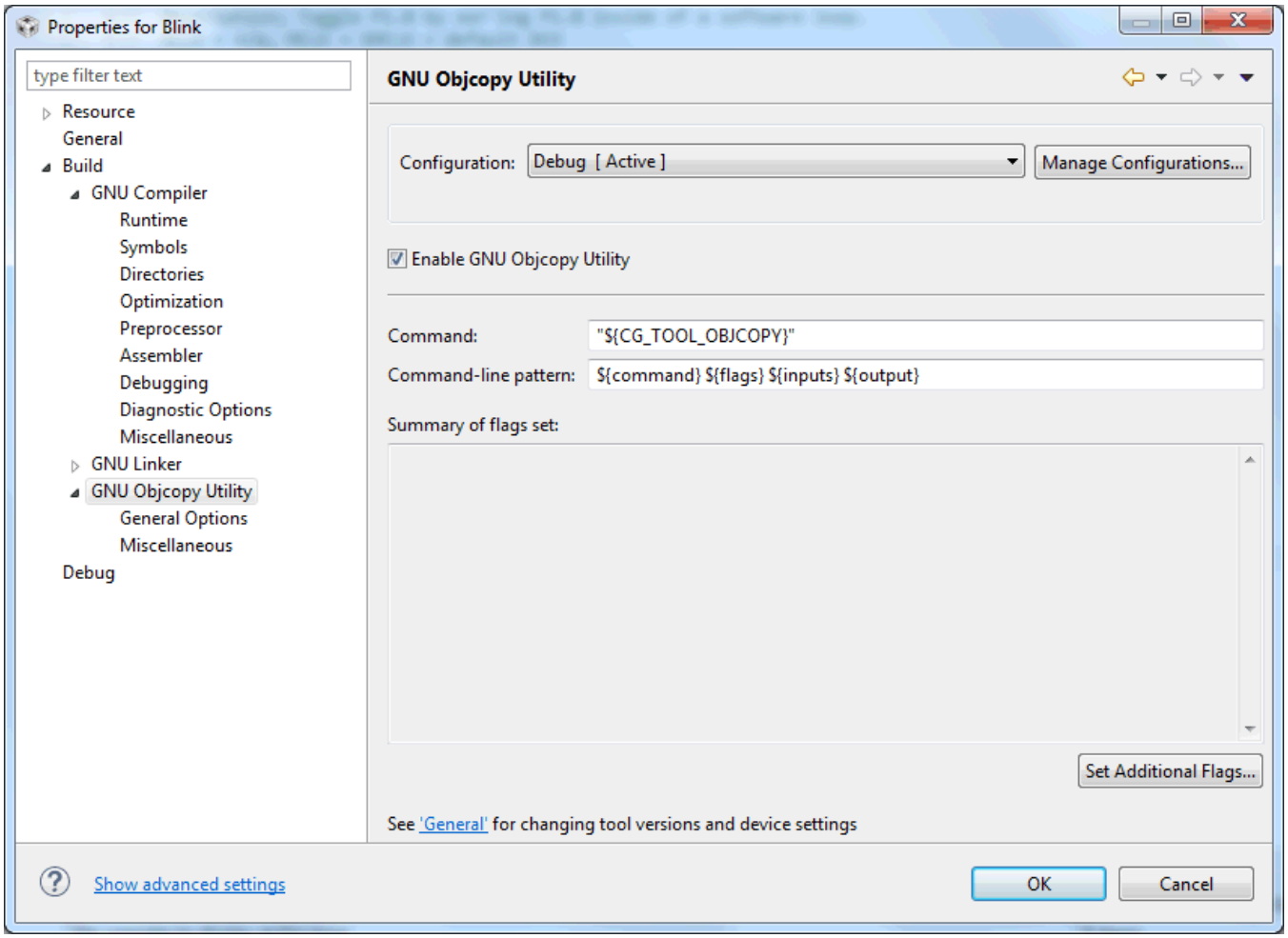

Figure 23. MSP430 GCC GNU Objcopy Utility Settings

Table 17 describes the options that are available for GNU Objcopy Utility.

Table 17. MSP430 GCC GNU Objcopy Utility Settings

Option	Description
Enable GNU Objcopy Utility	Enable this option to enable the GNU Objcopy Utility. It is disabled by default.
Command	GNU Objcopy location
Command-line pattern	Command line parameters
Summary of flags set	Command line with which the GNU Objcopy is called. Displays all the flags passed to the Objcopy command.

Figure 24 shows the MSP430 GCC GNU Objcopy Utility General Options settings window.

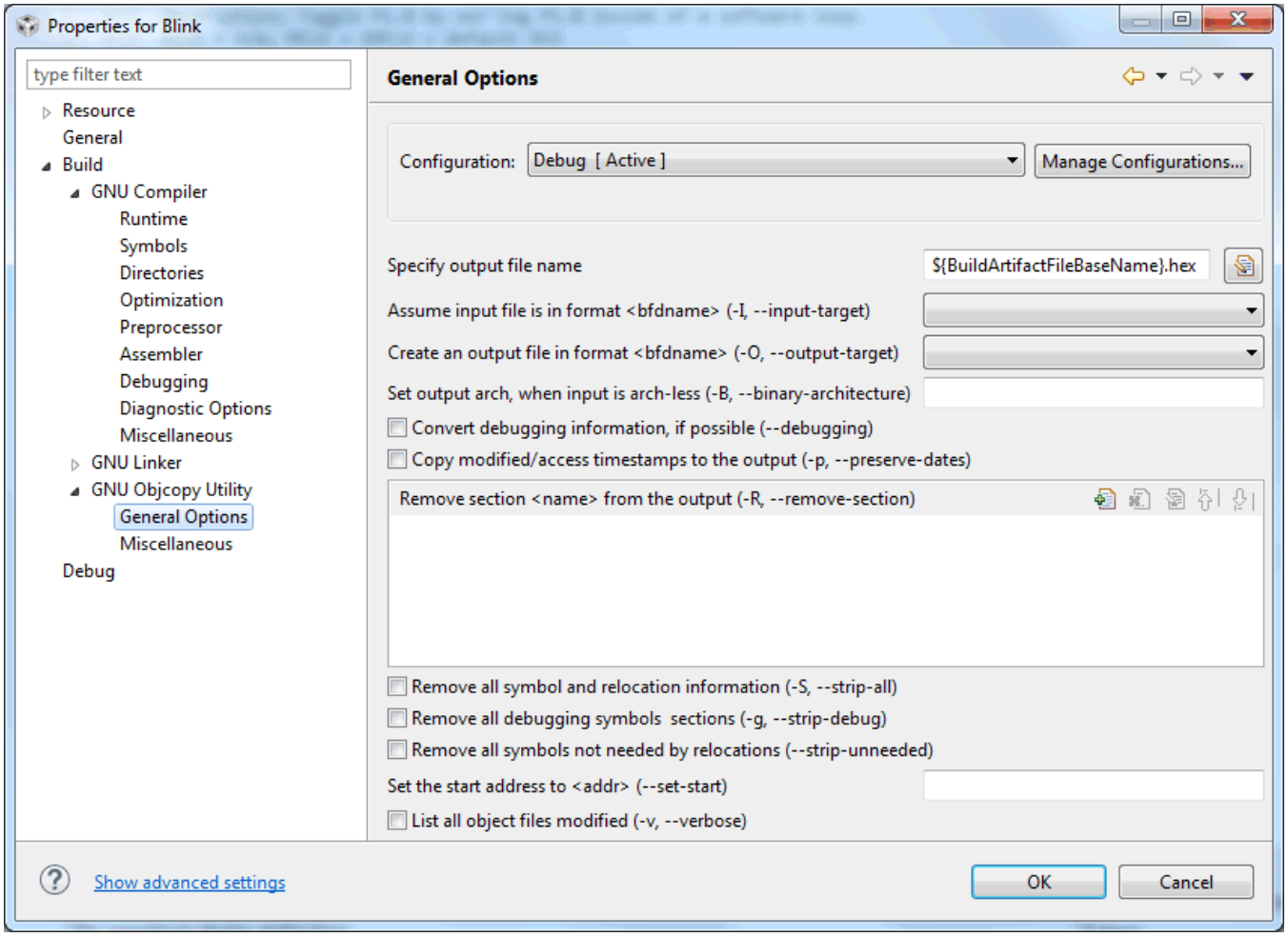

Figure 24. MSP430 GCC GNU Objcopy Utility General Options Settings

Table 18 describes the options that are available for GNU Objcopy Utility General Options.

Table 18. MSP430 GCC GNU Objcopy Utility General Options Settings

Option	Description
Specify output file name	Specifies the output file name
Assume input file is in format <bfdname> (-I, --input-target)	-I bfdname --input-target=bfdname Consider the source file's object format to be bfdname, rather than attempting to deduce it.
Create an output file in format <bfdname> (-O, --output-target)	-O bfdname --output-target=bfdname Write the output file using the object format bfdname.
Set output arch, when input is arch-less (-B, --binary-architecture)	-B bfdarch --binary-architecture=bfdarch Useful when transforming an architecture-less input file into an object file. In this case the output architecture can be set to bfdarch.
Convert debugging information, if possible (--debugging)	Convert debugging information, if possible. This is not the default because only certain debugging formats are supported, and the conversion process can be time consuming.
Copy modified/access timestamps to the output (-p, --preserve-dates)	Set the access and modification dates of the output file to be the same as those of the input file.

Table 18. MSP430 GCC GNU Objcopy Utility General Options Settings (continued)

Option	Description
Remove section <name> from the output (-R, --remove-section)	-R sectionpattern --remove-section=sectionpattern Remove any section matching sectionpattern from the output file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable. Wildcard characters are accepted in sectionpattern. Using the -j and -R options together results in undefined behavior.
Remove all symbol and relocation information (-S, --strip-all)	Do not copy relocation and symbol information from the source file.
Remove all debugging symbols sections (-g, --strip-debug)	Do not copy debugging symbols or sections from the source file.
Remove all symbols not needed by relocations (--strip-unneeded)	Strip all symbols that are not needed for relocation processing.
Set the start address to <addr> (--set-start)	Set the start address of the new file to the specified value. Not all object file formats support setting the start address.
List all object files modified (-v, --verbose)	Verbose output: list all object files modified. In the case of archives, 'objcopy -V' lists all members of the archive.

Figure 25 shows the MSP430 GCC GNU Objcopy Utility Miscellaneous settings window.

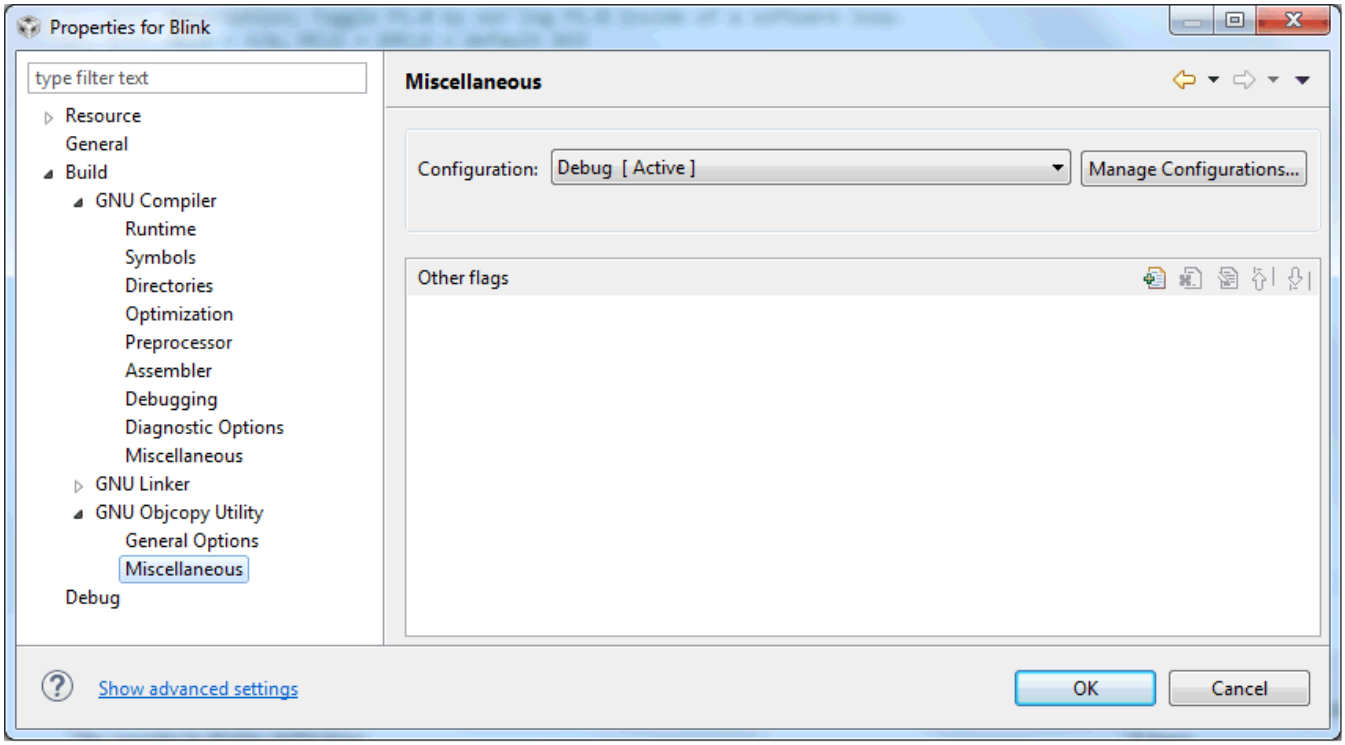

Figure 25. MSP430 GCC GNU Objcopy Utility Miscellaneous Settings

Table 19 describes the options that are available for GNU Objcopy Utility Miscellaneous.

Table 19. MSP430 GCC GNU Objcopy Utility Miscellaneous Settings

Option	Description
Other flags	Specifies individual flags based on the user requirements.

3.4 Change an Existing CCS project That Uses TI Compiler to MSP430 GCC

An existing CCS project that uses the TI compiler can be changed to use MSP430 GCC. Refer to Using MSP430-GCC with CCSv6 for more details.

3.5 Create a New CDT Project Using MSP430 GCC

A standard Eclipse C/C++ project (rather than a CCS project) can use the MSP430 GCC compiler. This would be necessary if you want to debug using GDB instead of the CCS debugger.

To create a new Eclipse C/C++ project that uses MSP430 GCC tools to build the executable, refer to Using MSP430-GCC with CCSv6.

3.6 GDB With MSP430 and CCSv6

CCSv6 and later can use GDB to debug MSP430 devices. To use the CCS GUI for things like setting and removing breakpoints, the project should be an Eclipse C/C++ project rather than a CCS project.

Refer to GDB with MSP430 and CCSv6 for more details.

3.7 CCS Compared to MSP430 GCC

Some CCS features are not supported in MSP430 GCC. These features are:

- Optimizer Assistant
- ULP Advisor
- Memory Protection Unit and Intellectual Property Encapsulation GUI configuration
- Memory allocation

To gain access to these features, the TI Compiler must be used.

4 MSP430 GCC Stand-Alone Package

4.1 MSP430 GCC Stand-Alone Packages

The MSP430 GCC stand-alone package is provided for users who prefer to use the MSP430 GCC compiler with other IDE or console-based solutions for compiling and debugging. This stand-alone package supports different operating systems and is provided in different formats:

- GCC, Binutils, and GDB binaries for Windows, Linux, and macOS
- MSP430 header and linker files
- MSP430 GCC source code
- GBD agent configuration

Table 20 lists all the available MSP430 GCC stand-alone packages.

Table 20. MSP430 GCC Stand-Alone Package

Software	Description
msp430-gcc-full-linux-installer-x.x.x.x.run	MSP430 GCC Linux installer including support files and debug stack and USB drivers. Apply **sudo chmod +x** <installer> before executing the package
msp430-gcc-full-windows-installer- x.x.x.x.exe	MSP430 GCC Windows installer including support files and debug stack and USB drivers
msp430-gcc-full-mac-installer- x.x.x.x.app	MSP430 GCC macOS installer including support files and debug stack and USB drivers
msp430-gcc-linux-installer- x.x.x.x.run	MSP430 GCC Linux installer. Compiler only. Apply **sudo chmod +x** <installer> before executing the package
msp430-gcc-windows-installer- x.x.x.x.exe	MSP430 GCC Windows installer. Compiler only.
msp430-gcc-mac-installer- x.x.x.x.app	MSP430 GCC macOS installer. Compiler only.
msp430-gcc-support-files.zip	Header files
msp430-gcc-source.tar.bz2	MSP430 GCC source files
md5sum.txt	MD5 checksums

4.1.1 MSP430 GCC Stand-Alone Package Folder Structure

The placeholder INSTALL_DIR refers to the directory where you installed the GCC MSP430 package.

- INSTALL_DIR
  - bin
    - MSP430 GCC Compiler binary
    - GDB binary
    - binutils
    - Tcl
    - GDB Agent
    - MSP430 Debug Stack
  - common
  - docs
  - emulation
    - Windows USB-FET Drivers
  - examples
  - include
    - MSP430 Support Files
  - lib
  - libexec
  - msp430-elf

- lib
  - libatomic
  - libgcc
  - libssp
  - libstdC++-v3
  - libbacktrace
  - libgcc-math
  - libgloss
  - libiberty
  - libsanitizer
  - newlib
- msp430.dat
- GCC_xx_manifest.pdf

4.2 Package Content

MSP430 GCC contains binary and sources software development tools for all TI MSP430 devices. The toolchain contains: compiler, assembler, linker, debugger, libraries, and other utilities.

These software development tools include:

- Compiler: MSP430 GCC (**msp430-elf-gcc**) is configured to compile C or C++.
- binutils: assembler, archiver, linker, librarian, and other programs.
- Newlib is the standard C library.
- Debugging: gdb_agent_console.exe and gdb_agent_gui.exe
- Source code: Compiler source code is available at http://www.ti.com/tool/msp430-gcc-opensource.

4.3 MSP430 GCC Options

The MSP430 GCC toolchain supports the options in Table 21 in addition to the standard GCC options.

For the most up-to-date information, refer to the GCC online documentation.

Table 21. MSP430 GCC Command Options

Option	Description
-masm-hex	Force assembly output to always use hex constants. Normally such constants are signed decimals, but this option is available for test suite or aesthetic purposes.
-mmcu=	Select the MCU to target. This is used to create a C preprocessor symbol based upon the MCU name, converted to upper case and prefixed and postfixed with '__'. This in turn is used by the 'msp430.h' header file to select an MCU-specific supplementary header file. The option also sets the ISA to use. If the MCU name is one that is known to only support the 430 ISA then that is selected, otherwise the 430X ISA is selected. A generic MCU name of 'msp430' can also be used to select the 430 ISA. Similarly, the generic 'msp430x' MCU name selects the 430X ISA. In addition an MCU-specific linker script is added to the linker command line. The script's name is the name of the MCU with '.ld' appended. Thus specifying '-mmcu=xxx' on the gcc command line defines the C preprocessor symbol __XXX__ and cause the linker to search for a script called 'xxx.ld'. This option is also passed on to the assembler.
-mwarn-mcu -mno-warn-mcu	This option enables or disables warnings about conflicts between the MCU name specified by the -mmcu option and the ISA set by the -mcpu option or the hardware multiply support set by the -mhwmult option. It also toggles warnings about unrecognized MCU names. This option is on by default.
-mcpu=	Specifies the ISA to use. Accepted values are 'msp430', 'msp430x' and 'msp430xv2'. This option is deprecated. The '-mmcu=' option should be used to select the ISA.
-msim	Link to the simulator runtime libraries and linker script. Overrides any scripts that would be selected by the '-mmcu=' option.

Table 21. MSP430 GCC Command Options (continued)

Option	Description
-mlarge	Use large-model addressing (20-bit pointers, 32-bit size_t).
-msmall	Use small-model addressing (16-bit pointers, 16-bit size_t).
-mrelax	This option is passed to the assembler and linker, and allows the linker to perform certain optimizations that cannot be done until the final link.
-mhwmult=	Describes the type of hardware multiply supported by the target. Accepted values: • 'none' for no hardware multiply • '16bit' for the original 16-bit-only multiply supported by early MCUs • '32bit' for the 16/32-bit multiply supported by later MCUs • 'f5series' for the 16/32-bit multiply supported by F5-series MCUs. • 'auto' can also be given. This tells GCC to deduce the hardware multiply support based upon the MCU name provided by the '-mmcu' option. If no '-mmcu' option is specified, then '32bit' hardware multiply support is assumed. 'auto' is the default setting. Hardware multiplies are normally performed by calling a library routine. This saves space in the generated code. When compiling at '-O3' or higher, however, the hardware multiplier is invoked inline. This makes for larger but faster code. The hardware multiply routines disable interrupts while running and restore the previous interrupt state when they finish. This makes them safe to use inside interrupt handlers as well as in normal code.
-minrt	Enable the use of a minimum runtime environment (no static initializers or constructors). This is intended for memory-constrained devices. The compiler includes special symbols in some objects that tell the linker and runtime which code fragments are required.
-mcode-region= -mdata-region=	These options tell the compiler where to place functions and data that do not have one of the lower, upper, either or section attributes. Possible values are: • lower • upper • either • any The first three behave like the corresponding section attribute; see Section 7.4 for placement information. The "any" value is the default. It leaves placement entirely up to the linker script and how the linker assigns the standard sections (.text, .data, and so on) to the memory regions.
-msilicon-errata=NAME[,NAME...]	Implements fixes for named silicon errata. Multiple silicon errata can be specified by multiple uses of the -msilicon-errata option or by including the errata names, separated by commas, on an individual -msilicon-errata option. Errata names currently recognized by the assembler are: cpu4 = PUSH #4 and PUSH #8 need longer encodings on the MSP430. This option is enabled by default, and cannot be disabled. cpu8 = Do not set the SP to an odd value. cpu11 = Do not update the SR and the PC in the same instruction. cpu12 = Do not use the PC in a CMP or BIT instruction. cpu13 = Do not use an arithmetic instruction to modify the SR. cpu19 = Insert NOP after CPUOFF.
-msilicon-errata-warn=NAME[,NAME...]	Like the -msilicon-errata option, except that instead of fixing the specified errata, a warning message is issued instead. This option can be used with -msilicon-errata to generate messages whenever a problem is fixed, or on its own to inspect code for potential problems.
-mdevices-csv-loc=PATH	Specify the full PATH to the devices.csv file. (This PATH should include the devices.csv filename.) This option should not be used directly. Instead, use one of the methods described in Section 4.6.
-mdisable-device-warnings	Disable warnings emitted when the devices.csv file cannot be found by the compiler.

4.4 MSP430 Built-in Functions

GCC provides special built-in functions to aid in the writing of interrupt handlers in C.

__bic_SR_register_on_exit (int mask)

This clears the indicated bits in the saved copy of the status register that currently resides on the stack. This only works inside interrupt handlers and the changes to the status register only take effect after the handler returns.

__bis_SR_register_on_exit (int mask)

This sets the indicated bits in the saved copy of the status register that currently resides on the stack. This only works inside interrupt handlers and the changes to the status register only take effect after the handler returns.

4.5 MSP430 GCC Interrupts Definition

To define an interrupt using MSP430 GCC, use the following syntax:

```
void __attribute__ ((interrupt(INTERRUPT_VECTOR))) INTERRUPT_ISR (void)
```

Example:

```
void __attribute__ ((interrupt(UNMI_VECTOR))) UNMI_ISR (void)
{ // isr }
```

You could also use the macro defined in the iomacros.h file:

```
#define __interrupt_vec(vec)__attribute__((interrupt(vec)))
```

Example:

```
void __interrupt_vec(UNMI_VECTOR) UNMI_ISR (void)
{}
```

4.6 Using MSP430 GCC Support Files

MSP430 GCC uses the `devices.csv` file that is included with the MSP430 GCC Support Files package to get the device data for the device specified with the -mmcu option. This causes the source code to be built for the correct ISA and hardware multiplier with any necessary symbols defined. This ensures the correct operation of the toolchain. When using the -mmcu option, the toolchain automatically selects the correct header files and linker scripts for the device specified.

MSP430 GCC uses a few different methods to find the support files (in the following precedence order):

1. **Command-line options for compiler include path and linker library path.** The compiler looks in the directories specified by the -I option, and the linker looks in the directories specified by the -L option. Pass the path to the "include" directory in the MSP430 GCC Support Files package to both of these options. CCS uses this method by default, so users of the CCS IDE should not have to make any changes.
2. **Directory specified via environment variable.** If the command line options described above are not provided, the toolchain examines the MSP430_GCC_INCLUDE_DIR environment variable. Set this environment variable to the full path to the "include" directory in the MSP430 GCC Support Files package. For example, on Linux:

   ```
   export MSP430_GCC_INCLUDE_DIR=/home/user/ti/gcc/include
   ```

3. **Default toolchain installation directory.** If neither a command line option nor the environment variable described above is provided, the toolchain checks the `msp430-elf/include/devices/` directory within the MSP430 GCC installation for the support files. Note that this "devices" directory does not exist in the latest toolchain installations, so the "include" directory from the support files package should be copied to this location. For example, on Linux:

   ```
   cp -r /home/user/ti/gcc/include/ /home/user/ti/gcc/msp430-elf/include/devices/
   ```

NOTE: The toolchain stops searching for support files once it finds `devices.csv`. The results may be different than expected if one of the higher-precedence methods finds out-of-date support files, despite newer support files being pointed to by one of the lower-precedence methods.

4.7 Quick Start: Blink the LED

This document assumes that a version of the GNU Make utility is installed on the system and that it is available on the system path. The placeholder **INSTALL_DIR** refers to the directory where the GCC MSP430 package is installed. The directory **INSTALL_DIR/bin** should be on the system path.

4.7.1 Building With a Makefile

1. In the command terminal, go to the **INSTALL_DIR\examples** directory.
2. There are examples for Windows, macOS, and Linux. They are located in the corresponding subdirectories. Choose one of the examples suitable for the operating system and MSP430 target device.
3. Change to the directory and type **make**.
4. The binary can now be downloaded and debugged on the target hardware.

4.7.2 Building Manually With gcc

To build one of the examples manually, open a terminal and change to the example for the target device and operating system. The compiler executable **msp430-elf-gcc** must be available on your system path.

```
msp430-elf-gcc -I <Path to MSP430 Support Files> -L <Path to MSP430 Support Files>
-T DEVICE.ld -mmcu=DEVICE -O2 -g blink.c -o blink.o
```

The placeholder **<Path to MSP430 Support Files>** is the directory that contains the MSP430 support files (header files and linker scripts to support the different MSP430 devices).

The placeholder **DEVICE** tells the compiler and linker to create code for the target device. The command line argument -T DEVICE.ld is optional, as the compiler automatically selects the correct linker script from the -mmcu=DEVICE argument.

Example

```
msp430-elf-gcc -I ../../../include -L ../../../include -T msp430fr5969.ld -mmcu=msp430fr5969 -O2
-g blink.c -o blink.o
```

4.7.3 Debugging

4.7.3.1 Starting GDB Agent

On Microsoft Windows, the GDB Agent is available as either as a small GUI application or on the command line. On GNU Linux, only the command line version is available.

4.7.3.1.1 Using the GUI

Open the **INSTALL_DIR/bin** directory and double-click gdb_agent_gui.

1. After the program starts, click the button **Configure**, select **msp430.dat**, and click **Open**.
2. Click on the button **Start** under the Panel Controls.
3. The "Log" window now contains the status message "Waiting for client".
4. Leave the window open until the end of the debugging process.

4.7.3.1.2 Using the Command Line

Open a command terminal, change to INSTALL_DIR and type:

Linux

./bin/gdb_agent_console msp430.dat

Windows

.\bin\gdb_agent_console msp430.dat

4.7.3.2 Debugging With GDB

4.7.3.2.1 Running a Program in the Debugger

1. In the command terminal, go to the INSTALL_DIR\examples\[Selected example], and type the command **make debug**.
2. This command starts GDB and waits for commands. This is indicated by the prompt <gdb>.
3. To connect GDB to the GDB Agent, type the command **target remote :55000** and press enter.
4. To load the program binary to the MSP430 target device, type **load**.
5. Type the command **continue** (short version: c) to tell GDB to run the loaded program.
6. The LED on the target board blinks.

4.7.3.2.2 Setting a Breakpoint

1. Connect GDB to the GDB Agent as previously described and load a program to the device.
2. To set a breakpoint on a function, type **break function name**.
3. To set a breakpoint on a source line, type **break filename:line**.
4. When you run the program, the program execution stops at the entry to the specified function or stops at the specified line.

4.7.3.2.3 Single Stepping

1. Connect GDB to the GDB Agent as previously described and load a program to the device.
2. After the debugger has stopped the program at a breakpoint, you can step through the code:
   - To execute the source line, type **next**. **next** does not step into functions, it executes the complete function and stops on the line following the function call.
   - To execute the next source line and step into functions, type **step**.
   - To execute the next instruction, type **nexti**.
   - To execute the next instruction and step into functions, type **stepi**.

4.7.3.2.4 Stopping or Interrupting a Running Program

1. Connect GDB to the GDB Agent as previously described and load a program to the device.
2. To stop a running program and get back to the GDB command prompt, type **Ctrl+C**. This currently applies only on Linux.

4.7.4 Creating a New Project

1. Create a directory for your project.
2. Copy one of the example project makefiles into the project directory.
3. Open the copied makefile and set the variable DEVICE to the target device.
4. Set the variable GCC_DIR to point to the directory where the GCC MSP430 package is installed.
5. Include all of the project source files (that is, the *.c files) as a dependency for the first target of the makefile.
6. Go to the project directory in a terminal and type **make** to build the project or **make debug** to start debugging the project.

```
OBJECTS=blink.o

GCC_DIR =  ../../../bin
SUPPORT_FILE_DIRECTORY = ../../../include

# Please set your device here
DEVICE  = msp430X
CC      = $(GCC_DIR)/msp430-elf-gcc
GDB     = $(GCC_DIR)/msp430-elf-gdb

CFLAGS = -I $(SUPPORT_FILE_DIRECTORY) -mmcu=$(DEVICE) -O2 -g
LFLAGS = -L $(SUPPORT_FILE_DIRECTORY) -T $(DEVICE).ld

all: ${OBJECTS}
    $(CC) $(CFLAGS) $(LFLAGS) $? -o $(DEVICE).out

debug: all
    $(GDB) $(DEVICE).out
```

4.8 GDB Settings

The GDB Agent is a tool to connect GDB with the target hardware to debug software. The GDB Agent uses the MSP430 debug stack to connect to the hardware and provides an interface to GDB. On Windows, both a console and a GUI application version of the GDB Agent are provided. Only the console application is supported on Linux.

4.8.1 Console Application

If you use the console application, run it from a command terminal using following syntax:

Linux

INSTALL_DIR/bin/gdb_agent_console INSTALL_DIR/msp430.dat

Windows

INSTALL_DIR\bin\gdb_agent_console INSTALL_DIR\msp430.dat

The console application opens a TCP/IP port on the local machine. It displays the port number in the console. By default, this port number is 55000.

4.8.2 GUI Application

After you start the GUI application, configure the GUI and then start the GDB server. For more information, refer to the XDS GDB Agent online documentation.

1. Click the **Configure** button and, in the Select board configuration file window, select the msp430.dat file. If successfully configured, an MSP430 device is displayed in the **<Targets>** list. The TCP/IP port for the GDB server is displayed when the MSP430 device is selected from the list.
2. To start the GDB Agent, click the **Start** button when the MSP430 device is selected.

4.8.3 Attaching the Debugger

After starting the debugger and to attach to the GDB server, use the target remote **[<host ip address>]:<port>** command, where **<port>** is the TCP/IP port from above. If the GDB Agent runs locally, omit the host IP address.

4.8.4 Configuring the Target Voltage

To configure the target voltage for the device, open the file msp430.dat in a text editor. To change the voltage, modify the key msp430_vcc. By default, this value is set to 3.3 V.

4.8.5 Resetting the Target

To reset the target, use the **monitor reset** command.

4.8.6 Halting the Target

To halt the target, use the **monitor halt** command.

5 Building MSP430 GCC From Sources

5.1 Required Tools

This document assumes that the required tools are installed on the system and that they are available on the system path.

- GNU make
- GCC and binutils
- bzip2 and tar
- curl, flex, bison, and texinfo

5.2 Building MSP430 GCC (Mitto Systems Limited)

The `README-build.sh` bash script included with the source-full package (and in the source-patches package) can be used to build the toolchain for Windows, Linux and macOS hosts. The script contains some distribution-specific instructions on how to install the pre-requisite tools from Section 5.1.

To build native Linux and macOS toolchains, follow the instructions in Section 5.2.1.

To build the toolchain for Windows hosts, follow both the instructions in Section 5.2.1 and then Section 5.2.2.

NOTE: If less than 2 GB of RAM is available during the build, the build may fail.

5.2.1 Building a Native MSP430 GCC Toolchain

Follow these steps to build Mitto MSP430 GCC for Linux and macOS:

1. Download the source-full tar archive (for example, msp430-gcc-7.3.0.9-source-full.tar.bz2) from the MSP430 GCC page.
2. Untar the file.
3. Change to the source-full directory.
4. Run `README-build.sh` to build the toolchain.
5. Build files are in the ./build folder.
6. Binaries/libs are in the ./install folder.

NOTE: An alternative to this process is to use the "source-patches" tar archive (for example, `msp430-gcc-7.3.0.9-source-patches.tar.bz2`) to apply patches to source tars as released by the upstream community.

Versions 7.3.0.9 and later include a script (`README-apply-patches.sh`), which downloads the upstream releases and applies the patches so the sources are ready for building. The `README-build.sh` script can then be used to build a native toolchain.

5.2.2 Building the MSP430 GCC Toolchain for Windows

Follow these instructions to build the toolchain for Windows hosts:

1. Begin by following the instructions in Section 5.2.1 to build a native toolchain.
2. Move the installation directory to a permanent location. (This is because the `README-build.sh` script deletes the "build" and "install" directories before starting the toolchain build.)
3. Install a cross-compiler for Windows:
   - On Ubuntu install the "mingw-w64" package as follows:
     ```
     > apt-get install mingw-w64
     ```
   - On Centos 7, first install the "Extra Packages for Enterprise Linux" (EPEL) repository, then install the mingw toolchain as follows:
     ```
     > yum install epel-release
     > yum install mingw64-gcc.x86_64 mingw64-gcc-c++.x86_64
     ```
4. Add the desired host platform to "configure_args_common" in `README-build.sh`. For 64-bit Windows this is usually "x86_64-w64-mingw32" and for 32-bit Windows it is "i686-w64-mingw32". For example:
   ```
   > configure_args_common='--target=msp430-elf --enable-languages=c,c++ --disable-nls
   --host=x86_64-w64-mingw32'
   ```

   NOTE: You can confirm that a cross-compiler is available for the target host by running `${HOST}-gcc --version`. For example:
   ```
   > x86_64-w64-mingw32-gcc --version
   ```
5. Make sure the native toolchain installed earlier is on the PATH.
6. Run `README-build.sh`.

5.3 Building MSP430 GCC Stand-Alone Full Package

- MSP430 GCC Compiler
  1. Download the MSP430 GCC Installer Compiler only from http://www.ti.com/tool/msp430-gcc-opensource.
  2. Use the generated MSP430 GCC version (see Section 5.2).
- USB driver package (Windows only)
  1. Download "Stand-alone Driver Installer for USB Low-Level Drivers" from http://www.ti.com/tool/mspds.
- MSPDS OS package
  1. Download "MSP Debug Stack Open Source Package" from http://www.ti.com/tool/mspds.
- Build MSPDebugStack
  1. Extract "MSP Debug Stack Open Source Package".
  2. Follow the instructions in "README-build.sh".
- GDB agent
  1. Download the GDB Agent from http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package.
- MSP430 support files for GCC
  1. Download "msp430-gcc-support-files.zip" from http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSPGCC/latest/index_FDS.html.

6 MSP430 GCC and MSPGCC

The new GCC compiler for MSP low-power microcontrollers conforms to the MSP Embedded Application Binary Interface (EABI) (see *MSP430 Embedded Application Binary Interface*). This allows GCC to interoperate with the proprietary TI compiler.

For example, assembly functions can be written in the same way, and libraries that are built with one compiler can be used as part of executables built with the other compiler. Aligning with the MSP EABI required breaking compatibility with the prior MSPGCC compiler. This document gives a brief overview of the ABI changes that are most likely to be noticed by and to affect a developer who is moving from MSPGCC to the newer GCC compiler for MSP.

6.1 Calling Convention

For developers writing assembly code, the most noticeable part of an ABI is the calling convention. Full specification of the calling convention is very detailed (see *MSP430 Embedded Application Binary Interface*), but developers writing assembly do not typically use most of it. There are three basic differences between MSPGCC and the GCC compiler for MSP in the calling convention that are important to be aware of:

- In MSPGCC, registers are passed starting with R15 and descending to R12. For example, if two integers are passed, the first is passed in R15 and the second is passed in R14. In contrast, the MSP430 EABI specifies that arguments are passed beginning with R12 and moving up to R15. So, in the same situation, registers R12 and R13 would hold the two arguments. In both cases, after the registers R12 through R15 are used, continued arguments are passed on the stack. If you are using stack-based arguments, you should consult the EABI specification.
- MSPGCC and the GCC compiler for MSP use different registers for the return value. MSPGCC places the return value in R15 (or R15 and consecutive lower registers if the value is larger than a word), while the EABI specifies that the return value is placed in R12.
- In MSPGCC, register R11 is considered a save on entry register and needs to be saved and restored by the callee if it is used in the called function. Conversely, the MSP EABI specifies that R11 is a save on call register, so it needs to be saved and restored by the calling function if its value will be needed after a function call. For comparison purposes, R4 to R10 are save on entry registers for both compilers, and R12 to R15 are save on call.

These are the key differences to be aware of when moving between the compilers. If you are writing assembly code that passes parameters on the stack or that passes structures by value, you should consult the MSP EABI document for additional information.

6.2 Other Portions of the ABI

Many other pieces make up the EABI, such as the object file format; debug information, and relocation information that is used when linking together files. However, in general, these pieces do not affect migration.

One other area to be aware of is that the details of data layout differ between ABIs. If you are relying on advanced data layout details such as layout of structures and bitfields, see *MSP430 Embedded Application Binary Interface*.

7 Appendix

7.1 GCC Intrinsic Support

The GCC Compiler supports the same intrinsics that the TI CGT for MSP430 does. These are:

- unsigned short __**bcd_add_short**(unsigned short op1, unsigned short op2);
- unsigned long __**bcd_add_long**(unsigned long op1, unsigned long op2);
- unsigned short __**bic_SR_register**(unsigned short mask); BIC mask, SR
- unsigned short __**bic_SR_register_on_exit**(unsigned short mask);
- unsigned short __**bis_SR_register**(unsigned short mask);
- unsigned short __**bis_SR_register_on_exit**(unsigned short mask);
- unsigned long __**data16_read_addr**(unsigned short addr);
- void __**data16_write_addr** (unsigned short addr, unsigned long src);
- unsigned char __**data20_read_char**(unsigned long addr);
- unsigned long __**data20_read_long**(unsigned long addr);
- unsigned short __**data20_read_short**(unsigned long addr);
- void __**data20_write_char**(unsigned long addr, unsigned char src);
- void __**data20_write_long**(unsigned long addr, unsigned long src);
- void __**data20_write_short**(unsigned long addr, unsigned short src);
- void __**delay_cycles**(unsigned long);
- void __**disable_interrupt**(void); AND __**disable_interrupts**(void);
- void __**enable_interrupt**(void); AND __**enable_interrupts**(void);
- unsigned short __**get_interrupt_state**(void);
- unsigned short __**get_SP_register**(void);
- unsigned short __**get_SR_register**(void);
- unsigned short __**get_SR_register_on_exit**(void);
- void __**low_power_mode_0**(void);
- void __**low_power_mode_1**(void);
- void __**low_power_mode_2**(void);
- void __**low_power_mode_3**(void);
- void __**low_power_mode_4**(void);
- void __**low_power_mode_off_on_exit**(void);
- void __**no_operation**(void);
- void __**set_interrupt_state**(unsigned short src);
- void __**set_SP_register**(unsigned short src);
- unsigned short __**swap_bytes**(unsigned short src);

7.2 GCC Function Attribute Support

- **interrupt** or **interrupt(x)**

  Make the function an interrupt service routine for interrupt "x". This attribute can also be used without an argument. If no argument is used, the function is not linked to an interrupt, but the function will have properties that are associated with interrupts.

- **wakeup**

  When applied to an interrupt service routine, wake the processor from any low-power state as the routine exits. When applied to other routines, this attribute is silently ignored.

- **naked**

  Do not generate a prologue or epilogue for the function.

- **critical**

  Disable interrupts on entry, and restore the previous interrupt state on exit.

- **reentrant**

  Disable interrupts on entry, and always enable them on exit.

7.3 GCC Data Attribute Support

- **noinit**

  Variables with the noinit attribute will not be initialized by the C runtime startup code or the program loader. Not initializing data in this way can reduce program startup times. A compiler warning will be provided if a variable marked with the noinit attribute is initialized to a constant value.

- **persistent**

  Variables with the persistent attribute will not be initialized by the C runtime startup code. Instead their value will be set once, when the application is loaded, and then never initialized again, even if the processor is reset or the program restarts. Persistent data is intended to be placed into Flash RAM, where its value will be retained across resets. The linker script used to create the application should ensure that persistent data is correctly placed. A compiler warning will be provided if a variable marked with persistent is not initialized to a constant value.

7.4 GCC Section Attribute Support

The following attributes can be applied to functions or data to specify whether to place them in high or low memory, or to let the linker decide where to place them.

- **upper**

  Place the function or data item in the .upper.text, .upper.data or .upper.bss section as appropriate for the type.

- **lower**

  Place the function or data item in the .lower.text, .lower.data or .lower.bss section as appropriate for the type.

- **either**

  Place the function or data item in the .either.text, .either.data or .either.bss section as appropriate for the type.

  The linker will decide whether to place the item in low memory (sections prefixed with .lower) or high memory (sections prefixed with .upper). Functions are placed in the appropriate .text section. Non-zero data is placed in the appropriate .data section. Data initialized to zero is placed in the appropriate .bss section.

8 References

1. Using the GNU Compiler Collection, Richard M. Stallman (http://gcc.gnu.org/onlinedocs/gcc.pdf). Refer to the *MSP430 Options* section.
2. GDB: The GNU Project Debugger, Free Software Foundation, Inc. (https://sourceware.org/gdb/current/onlinedocs/)

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from June 17, 2016 to May 31, 2018	Page
• CCS v8.x is the current version.	5
• Changed partner information from SOMNIUM to Mitto Systems Limited	7
• Changed OS X to macOS to match new Apple branding.	7
• Windows XP is no longer tested, and thus is not listed as a supported OS.	7
• Added information about use of -Og build option.	12
• Provided additional information about optimization options.	17
• Added -mdevices-csv-loc to Table 21, *MSP430 GCC Command Options*	34
• Added -mdisable-device-warnings to Table 21, *MSP430 GCC Command Options*	34
• Added Section 4.6, *Using MSP430 GCC Support Files*	35
• Listed additional required tools	40
• Added Section 5.2, *Building MSP430 GCC (Mitto Systems Limited)* and its subsections. Removed sections for building using previous partners.	40
• The interrupt function attribute can be used without an argument.	44
• Added Section 7.3, *GCC Data Attribute Support*	44
• Added Section 7.4, *GCC Section Attribute Support*	44

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265