

Developing Ember Applications

For the EM250 SoC Platform

xIDE for EM250, the Ember integrated development environment for the EM250, is part of the InSight development environment. xIDE for EM250 features project and build management, a code editor, and a full source-level debugger.

This document describes how to use xIDE for EM250 to create, compile, and debug EmberZNet applications for the EM250. It assumes that you have already followed the steps in the *Quick Start Guide* to install xIDE for EM250 and have obtained a license file.

xIDE for EM250 provides its own documentation through online help and links to PDF documents. Refer to these for detailed information about how to use xIDE that is beyond the scope of this document.

Contents

Opening an Existing Project	2
Project Workspace and Files	3
Building an Application	3
Uploading an Application Image	5
Creating a Custom Project	5
Configuring Preprocessor Settings	7
Building a Debug Image	10
Building a Debug Off Image	11
Building an Image for Use with the Application Bootloader	11
Generating an .ebl Image	11
Reducing Flash Usage	12
Debugging an Application	12



ember

Ember Corporation
47 Farnsworth Street
Boston, MA 02210
+1 (617) 951-0200
www.ember.com



wireless semiconductor solutions

Opening an Existing Project

EmberZNet includes several sample projects for the EM250 that you can use to start your own development. Examples in this guide refer to the Sink application, but other projects are similar. The *Ember Developer Kit User's Guide* describes all available sample applications.

To open the Sink sample project files:

1. Select the xIDE menu option Project | Open Workspace.
2. Browse to the directory where the EmberZNet stack is installed.
3. Select the `sink.xiw` workspace.

As shown in Figure 1, xIDE for EM250 displays the Sink application workspace in the Project Navigator.

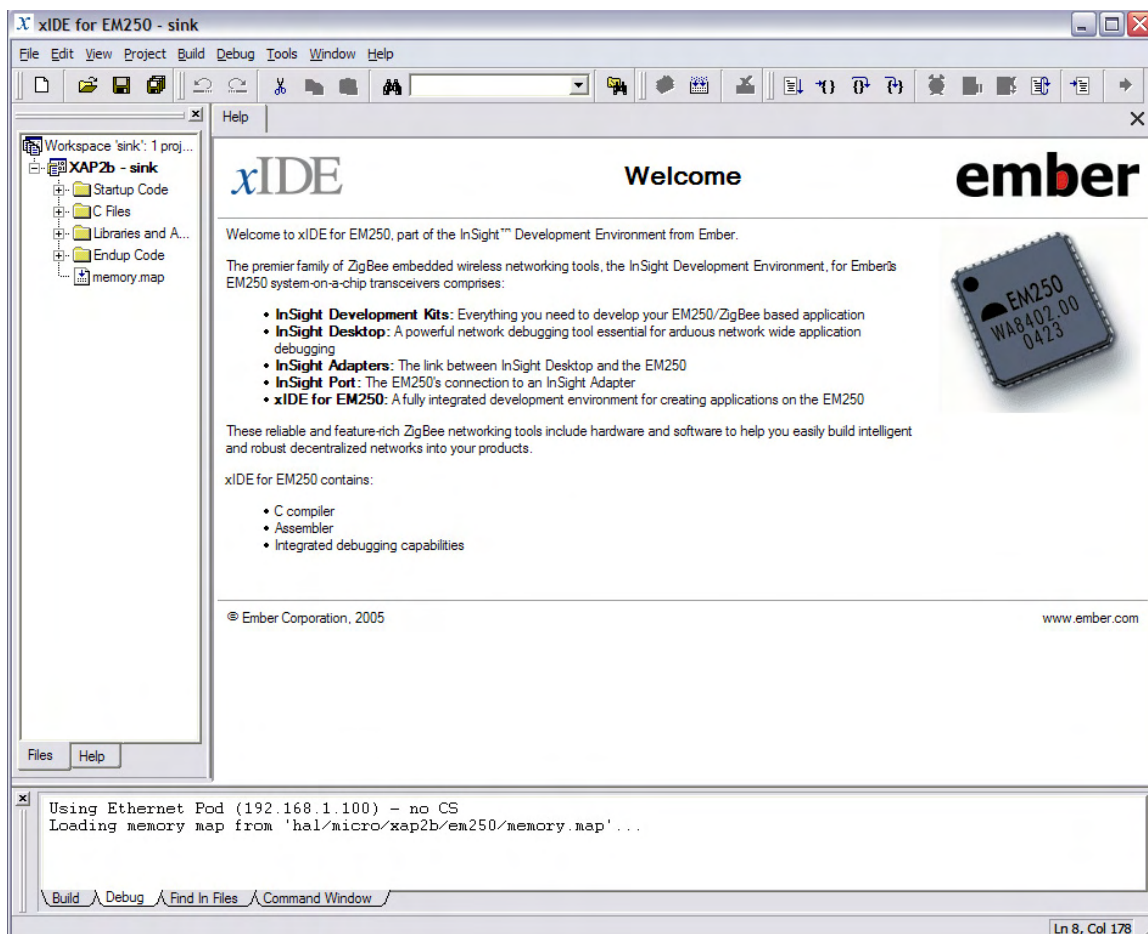


Figure 1. The Sink application workspace

Project Workspace and Files

Workspaces can include one or more projects. Each EmberZNet project contains the following folders and top-level items, which must appear in this sequence:

- **Startup code:** Assembly source code for low-level initialization of the EM250 and the core interrupt handler.
- **C files:** All C source files for the application, including the EmberZNet Hardware Abstraction Layer (HAL).
- **Libraries and assembler files:** The EmberZNet stack and auxiliary libraries required to build the application and HAL assembler files.
- **Endup code:** Low-level assembler utilities that can detect build-time error conditions such as unaligned packet buffers.
- **memory.map:** The internal memory map of the EM250 for the xIDE debugger. Do not modify this file.

Note: Because the order of folders defines the link order of the project, it must be maintained as shown in the preceding list.

Building an Application

To build an application, choose the menu option Build | Build (F7). Build progress and errors display in the lower output pane (see Figure 2). Double-click on an error to open its source file and go to its line.

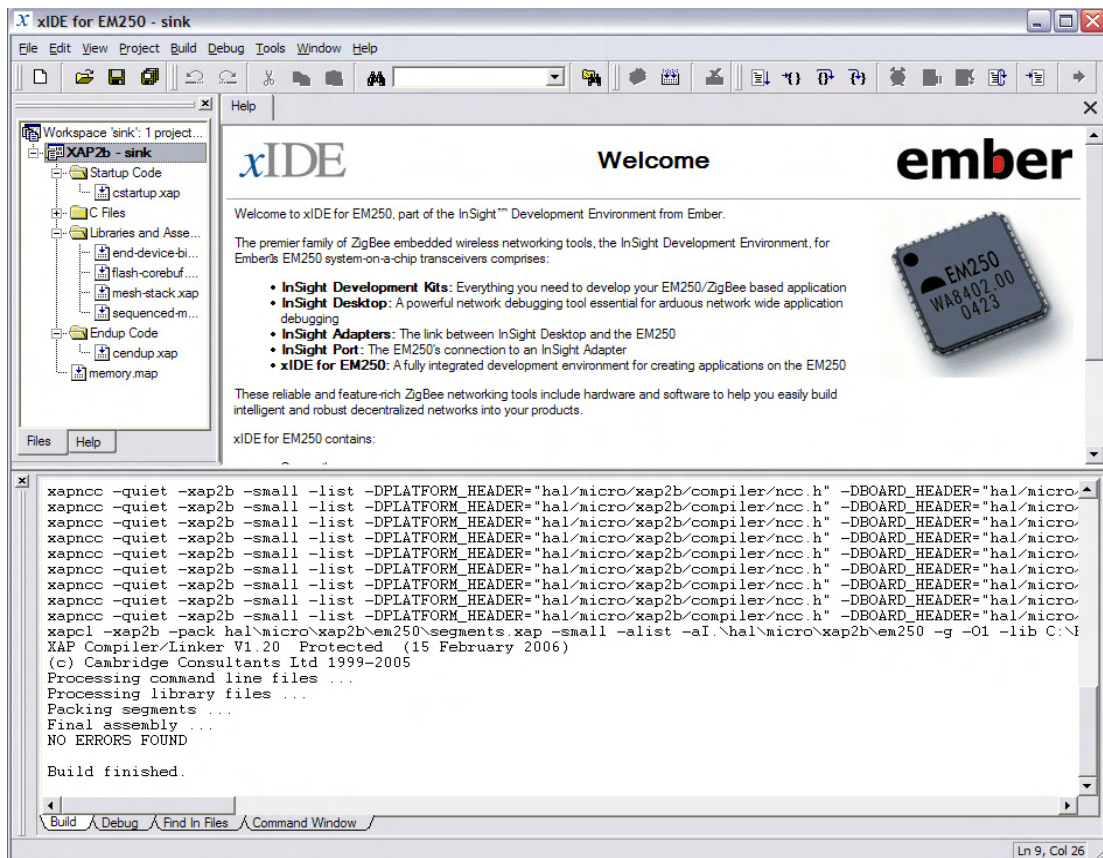


Figure 2. Build progress and error report pane

Build output

The build output of an application, along with the associated list files and debug information, are in a folder beneath the EmberZNet installation's `build` directory. The build directory is configured in the General configuration pane of the project properties. For example, the sample Sink build output is in:

```
install-dir/build/sink-xap2b-em250-em250-dev0455
```

The final built image of an EM250 application consists of two files with the following extensions:

- `xpv` contains the application's code image.
- `xdv` contains the application's constant data and any static initializers.

An `.eb1` image is also automatically generated as a post-build step for use with a bootloader. This file consolidates all code and constant data from the `.xpv` and `.xdv` files. More information about building an `.eb1` file is available on page 11.

Uploading an Application Image

You can upload an application image to an EM250 node with InSight Desktop or with the command-line utility `em2xx_load.exe`.

`em2xx_load.exe` utility

The `em2xx_load.exe` utility is located under the xIDE installation, in the subdirectory `SIF/bin`. This utility uploads an application to the specified node and starts execution.

You execute `em2xx_load.exe` with the following arguments:

```
em2xx_load.exe
  -sid [IP-address] path/base-filename[.ext] [-Run | -Reset]
```

where:

- `IP-address` specifies the IP address or hostname of the target node's InSight Adapter.
- `path` and `base-filename` specifies the path and base name of the `xpv` and `xdv` files to upload. Both files must be in the same folder. You can also specify an `.ebl` image, but the specification must include its file extension.
- `ext` optionally specifies the extension of the files to upload: `xpv`, `xdv`, or `ebl`. If no extension is specified, `em2xx_load.exe` looks for the `.xpv` and `.xdv` files.
- `-Run` tells `em2xx_load` to reset the EM250 and run the application after loading is complete.
- `-Reset` resets the EM250 and stops the application, allowing a debugger to be used from the beginning of execution.

More information about `em2xx_load` can be found in the *EmberZNet Utilities Guide*.

Creating a Custom Project

xIDE for EM250 includes project templates that you can use to build new applications. These templates include all the standard settings and common source files required to build EmberZNet applications. The templates only require the addition of main application source files.

To create a project from a template, choose the menu option Project | New. The New Project dialog (see Figure 3) displays, where you choose one of these templates:

- The **EM250 Template** creates a template for standard application builds. This template includes basic debug capability, such as the Virtual UART and assert tracking with InSight Desktop.
- The **EM250 Debug Template** creates a template for builds that are linked to the EmberZNet debug stack. This stack provides sophisticated integration with InSight Desktop, including verbose output such as API traces, which let you debug problems across an entire network.

You can specify any name for a new project. Other options must be configured so all the paths in the template project that reference the HAL sources and EmberZNet stack libraries line up properly:

- Set the location of the project to be located in the base directory of your EmberZNet installation. xIDE does not allow source files to be in a directory above the project file itself.
- Deselect the option **Create project folder**.
- Choose the option **Create new workspace**.

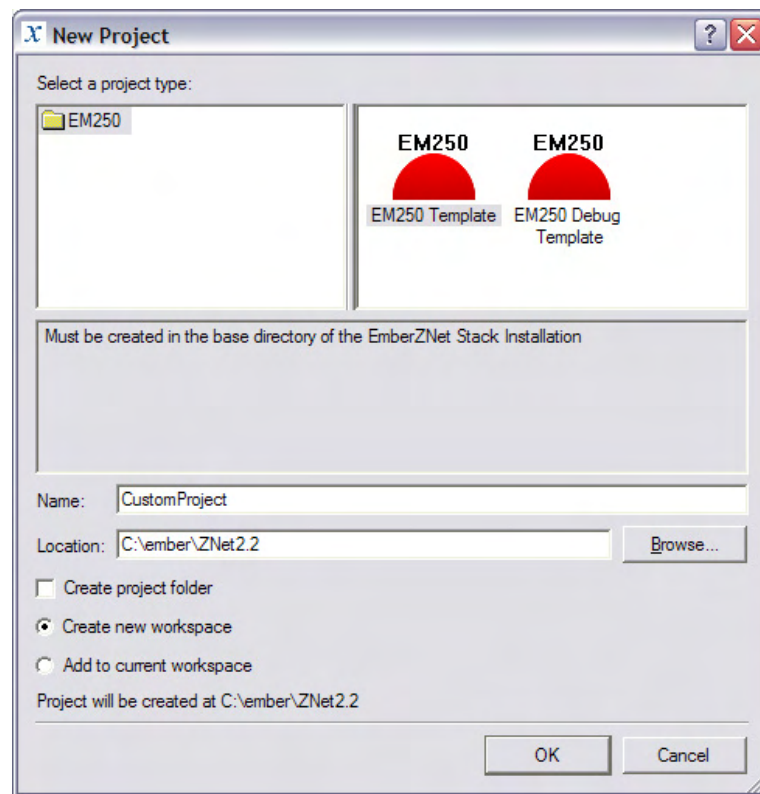


Figure 3. New Project dialog

The new project file contains all required startup code, endup code, HAL sources, and a default set of EmberZNet stack libraries. Several options are available for adding application source files to the project:

- Create application source files and add them to the project by choosing **File | New**.
- Add existing source files to the project by right-clicking on the desired folder in the Project Navigator and, from the context menu, choosing **Add Files to Folder**.
- Create folders by right-clicking on the project name in the Project Navigator and, from the context menu, choosing **New Folder**.

Note: After modifying the list of source files in your project, you might see build errors indicating improper alignment of packet buffers. These occur because the build order for `ember-configuration.c` was not maintained. To solve this, create a folder between the folders Startup Code and C Files, and move `ember-configuration.c` into it.

Depending on your application, you may also need to change the set of EmberZNet stack libraries to include different features such as enhanced security or end device binding support. When changing these libraries, be sure to include the proper version of the library that matches the debug configuration of the project. For more details about the different library options available, please refer to the *EmberZNet Application Developer's Guide*.

You may also need to change various preprocessor configuration settings for your application (see the *EmberZNet Application Developer's Guide* for details of all the possible options). The following settings are typically unique for each application:

- `BOARD_HEADER` and `BOARD_NAME`
- `CONFIGURATION_HEADER`
- `APPLICATION_TOKEN_HEADER`
- Serial configurations

Many additional configuration settings are available in the project properties. This document focuses on preprocessor settings that are important for developing EmberZNet applications. For full descriptions of all project settings, see the xIDE online help.

Configuring Preprocessor Settings

This section first describes how to configure the preprocessor settings and then discusses some of the common settings.

Preprocessor definitions are used to configure various operations of the EmberZNet stack and HAL. These definitions are accessible as project properties.

To access C preprocessor settings:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Compiler | Preprocessor Pane.
3. From the Preprocessor pane, select the option **Define Symbols**.
4. Click [...].

This invokes the Define Symbols dialog (see Figure 4), where you can:

- Add new symbols
- Remove old settings

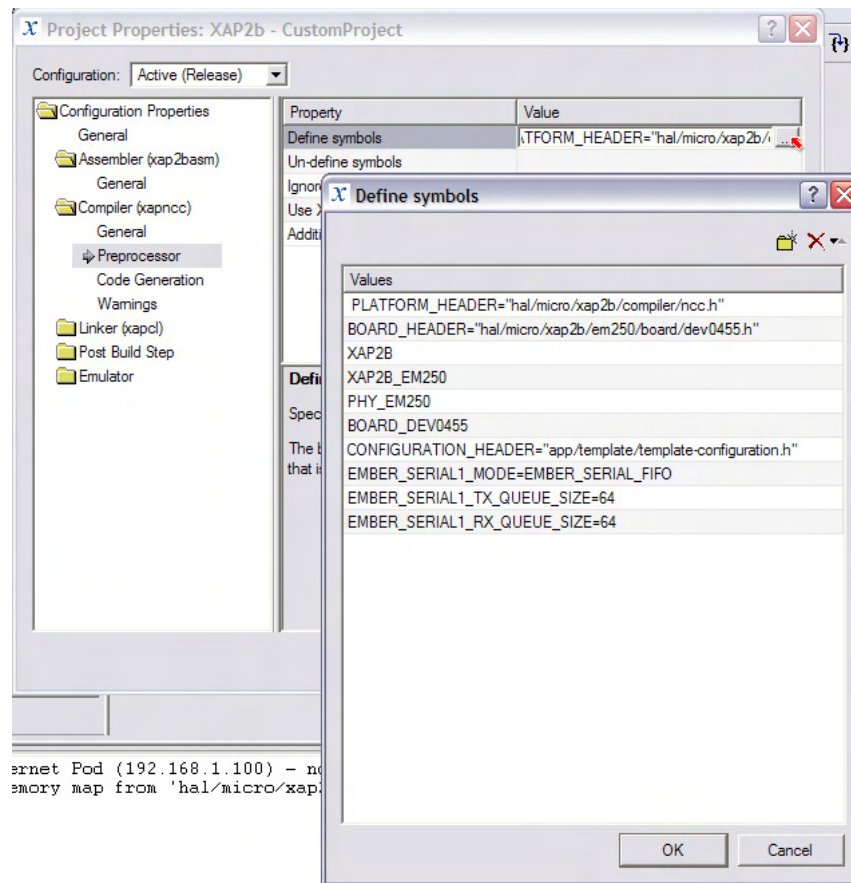


Figure 4. Define Symbols dialog

Some preprocessor settings must also be changed for the assembly source files. To change these settings:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Assembler | General Pane.
3. From the General pane, select the Define Symbols field.
4. Click [...].

These settings can now be manipulated in the same manner as the C preprocessor settings.

Header file configuration

The following preprocessor symbols define the header files that are automatically included to configure the EmberZNet stack and HAL.

Variable	Setting
PLATFORM_HEADER	Must be set to the header file <code>hal/micro/xap2b/compiler/ncc.h</code> . This file contains standard typedefs and macros used to build EmberZNet applications, which are specific to the EM250 and the xIDE for EM250 compiler.
BOARD_HEADER	Must be set to the path of a board header file that configures the EmberZNet HAL for use with custom hardware. For example, <code>hal/micro/xap2b/em250/board/dev0455.h</code> is the setting for the development kit breakout board. When changing the header, also be sure to change the HAL configuration symbol <code>BOARD_NAME</code> (described in the next section).
CONFIGURATION_HEADER	Set to the path of a header file that configures memory allocations for a particular application, such as the number of packet buffers and binding table sizes. This definition is common but not required.
APPLICATION_TOKEN_HEADER	Set to the path of a header file that includes definitions for any custom application tokens. This definition is optional.
ZCL_CONFIGURATION_HEADER	Set to the path of a header file that includes application-specific configurations for the ZCL. This definition is optional, and is only needed if using the ZCL utilities.

HAL configuration

These settings are required:

Variable	Setting
XAP2B	Defines the platform in use. Do not change this setting.
XAP2B_EM250	Defines the specific microcontroller in use. Do not change this setting.
PHY_EM250	Defines the radio physical layer in use. Do not change this setting.
BOARD_NAME	Defines the board to use with a value that contains the base filename of the <code>BOARD_HEADER</code> configuration variable. For example, if using the development kit breakout board, set to <code>BOARD_DEV0455</code> . Be sure to also set the <code>BOARD</code> variable in order to properly generate an <code>.ebl</code> image file (see page 11).

Serial configuration

The following variables are optional, but when used the first three must always be defined together. When using an EmberZNet debug stack, the same settings are also available for use with the Virtual UART as serial port 0.

Variable	Setting
EMBER_SERIAL1_MODE	EMBER_SERIAL_FIFO or EMBER_SERIAL_BUFFER.
EMBER_SERIAL1_TX_QUEUE_SIZE	Any power of 2 between 2–128, inclusive.
EMBER_SERIAL1_RX_QUEUE_SIZE	Any power of 2 between 2–128, inclusive.
EMBER_SERIAL1_BLOCKING	Optional, but use with caution as it can interfere with the timing of EmberZNet stack operation.

Other configuration options

These optional settings configure various behaviors of the EmberZNet stack and HAL.

Variable	Setting
DISABLE_WATCHDOG	Used for applications that do not require the watchdog, or when debugging an application within xIDE.
DEBUG DEBUG_OFF	Configures the EmberZNet HAL for use with a debug stack, enabling the advanced debug features of InSight Desktop, or a debug off stack for the smallest code size possible. Other project changes as described below must also be made when changing DEBUG modes.

Building a Debug Image

Debug builds of an application let you exploit the advanced network-wide debug features of InSight Desktop. If you did not start with a debug template, you can convert your application to a debug build by following these steps:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Assembler | General | Define Symbols.
3. In the Define Symbols dialog, add the \$DEBUG symbol.
4. From Project Properties, select Compiler | Preprocessor | Define Symbols.
5. In the Define Symbols field, add DEBUG.
6. Replace the stack and auxiliary libraries with their DEBUG variants.
7. Rebuild the application.

Building a Debug Off Image

Debug off builds allow for the smallest possible application images, but severely limit the debug features of InSight Desktop. You can convert your application to a debug off build by following these steps:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Assembler | General | Define Symbols.
3. In the Define Symbols dialog, add the `$DEBUG_OFF` symbol.
4. From Project Properties, select Compiler | Preprocessor | Define Symbols.
5. In the Define Symbols field, add `DEBUG_OFF`.
6. Replace the stack and auxiliary libraries with their `DEBUG_OFF` variants.
7. Rebuild the application.

Building an Image for Use with the Application Bootloader

By default, EmberZNet applications are built for use with the Standalone Bootloader, but the Application Bootloader may be preferred for certain applications. You can convert your application to use the Application Bootloader by following these steps:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Assembler | General | Define Symbols.
3. In the Define Symbols dialog, add the `$APP_BTL` symbol.
4. From Project Properties, select Compiler | Preprocessor | Define Symbols.
5. In the Define Symbols field, add `APP_BTL`.
6. Rebuild the application.

Generating an .eb1 Image

Project properties are configured to generate an `.eb1` image as a post-build step. The `.eb1` image is generated for use with a bootloader. If you change the board type used by your application, you must also adjust the `BOARD` variable in the post-build step as follows, or the `.eb1` image might not be correctly generated.

1. Choose Project | Properties from the main menu.
2. From Project Properties, click **Post Build Step** and select the Build Command pane.
3. Select Additional Variables and click [...].
4. Set the `BOARD` variable to the name of the board to use. For example, to use the Ember development kit breakout board, set this variable as follows:

```
BOARD=dev0455
```

Reducing Flash Usage

Dead Stripping

xIDE for EM250 supports automatic removal of unused sections of code from a built application, also known as *dead stripping*. If dead stripping is enabled, xIDE for EM250 works with the linker to find unused function symbols and their descendants. These sections of code are removed before the linker makes its final pass.

To enable dead stripping:

1. Choose Project | Properties from the main menu.
2. From Project Properties, select Assembler | General | Define Symbols.
3. In the Define Symbols dialog, add the `$DEADSTRIP` symbol.

Note: Project templates enable dead stripping by default.

Peephole optimizations

xIDE for EM250 also supports a number of peephole optimizations which can be enabled to further improve code size. These optimizations make small adjustments to the assembly generated by the compiler to yield more efficient sequences of instructions.

To enable the peephole optimizations:

1. From Project Properties, select Assembler | General | Define Symbols.
2. Add the symbol `$PEEPHOLE` to the list in the Define Symbols dialog.

Note: Project templates enable the peephole optimizations by default.

Detailed results of the peephole optimizations used are displayed at the end of the build progress information.

Debugging an Application

xIDE offers complete integrated debugging capability for solving problems in your application. It is best suited for diagnosing problems that affect only one node. To debug network problems across multiple devices, use InSight Desktop.

Note: When the XAP2b CPU core inside the EM250 is halted with the debugger, all peripherals continue to run normally, including the watchdog timer. To keep the watchdog from resetting the EM250 two seconds after any breakpoints have been hit in the debugger, the application being debugged should be built with the watchdog disabled. This is easily done by including the preprocessor setting `DISABLE_WATCHDOG` in the project properties (see Figure 5), or by adding `#define DISABLE_WATCHDOG` to the application's `CONFIGURATION_HEADER`.

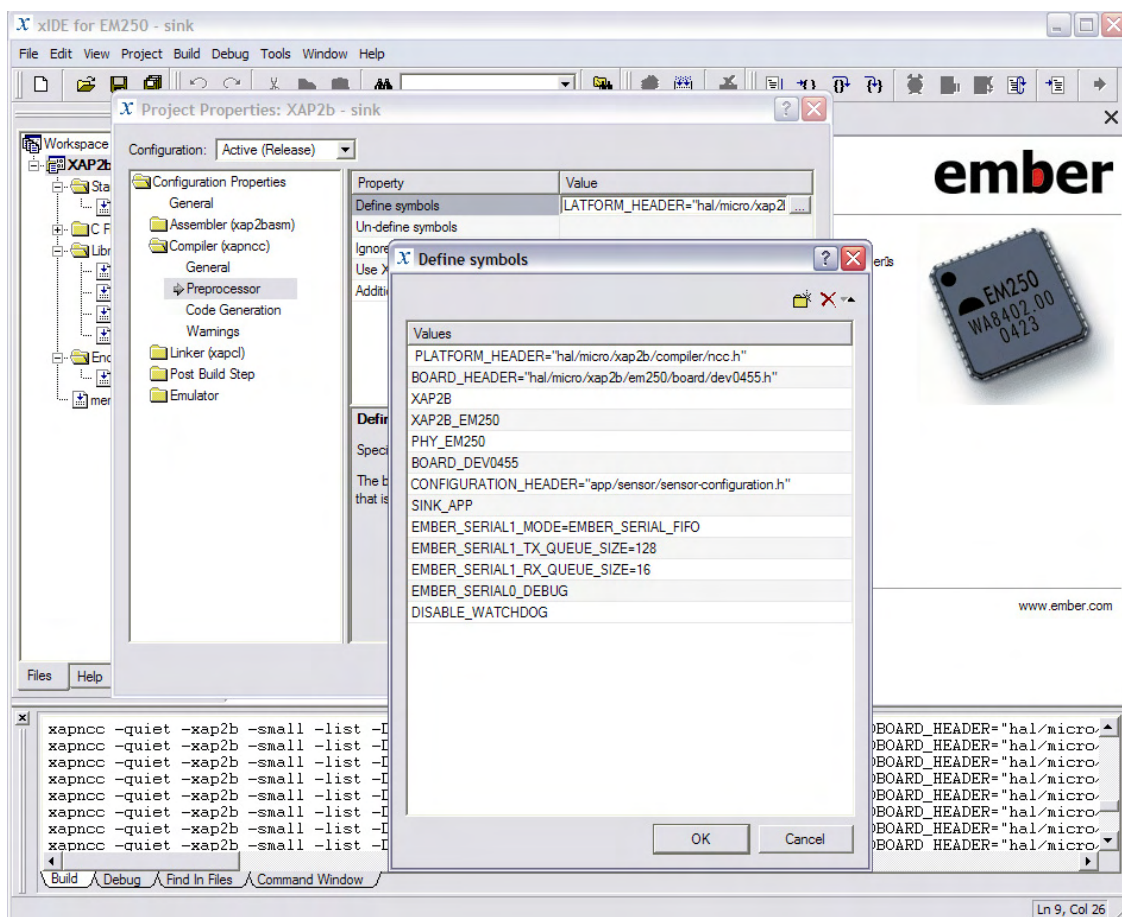


Figure 5. Including the preprocessor symbol `DISABLE_WATCHDOG` in the project properties

Configuring a project for the xIDE debugger

To interactively debug a node, you must configure the Emulator pane of the project properties to connect to the InSight Adapter attached to the node. To do this, from the SIF Slave drop-down menu select the appropriate Ethernet Pod IP address (see Figure 6).

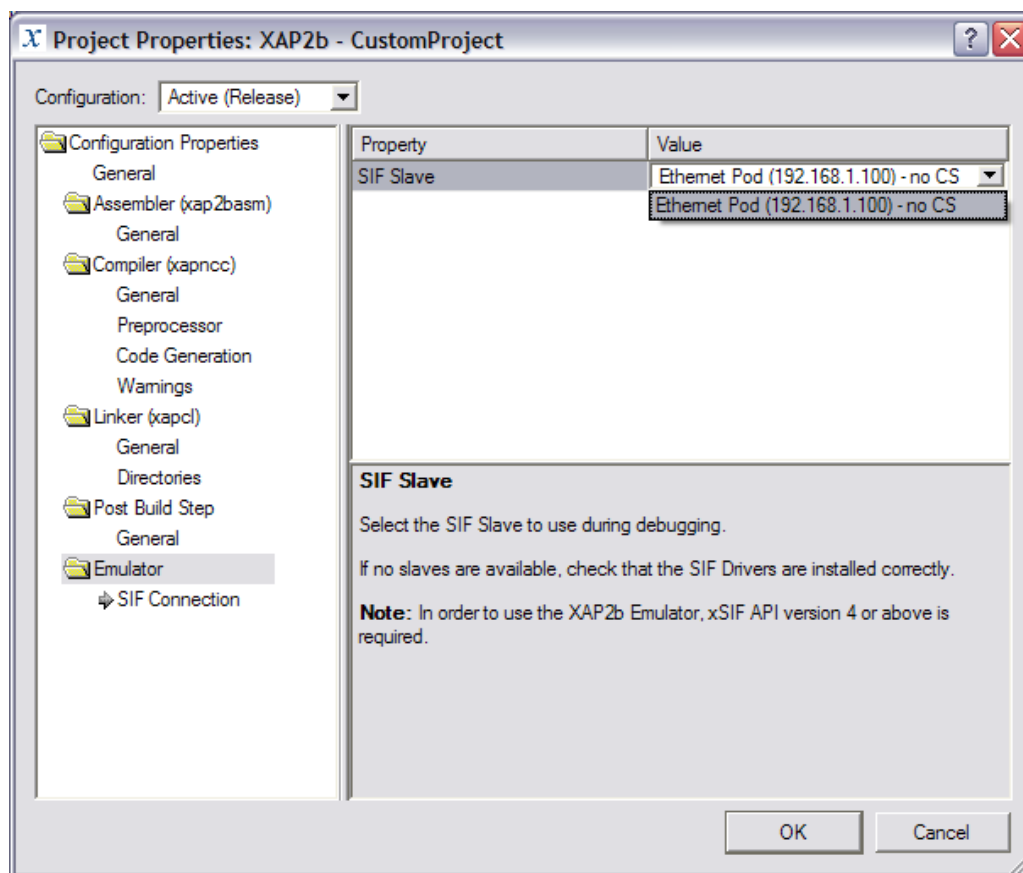


Figure 6. Selecting the Ethernet Pod IP address

Troubleshooting

If the menu does not list the desired IP address, try one or both of the following actions:

- Load an image to the desired node using either `em2xx_load` or InSight Desktop.
- Restart xIDE for EM250.

If the InSight Adapter is still not listed, you can add it manually:

1. From the main menu, choose Tools | Auto-Detect SIF. This starts the SIF Auto-detect wizard.
2. Choose **Add SIF Pods** from the wizard, then click **Next**.
3. Choose **No**, manually enter details of a new SIF Pod, then click **Next**.
4. Deselect **Use Network Name**.
5. Enter the IP address of your InSight Adapter.

6. Click **Next** twice.
7. Your InSight Adapter should be automatically detected and configured. Click **Finish**.

If the debugger does not seem to operate properly, you might also need to manually set the SIF Slave Type for your InSight Adapter:

1. From the main menu, choose **Tools | Configure SIF**.
2. Find the InSight Adapter listed in Pod View.
3. Switch to Slave View in the SIF Configuration dialog.
4. Set **Slave type** to **XAP2** (see Figure 7).
5. Click **OK** and try using your InSight Adapter again.

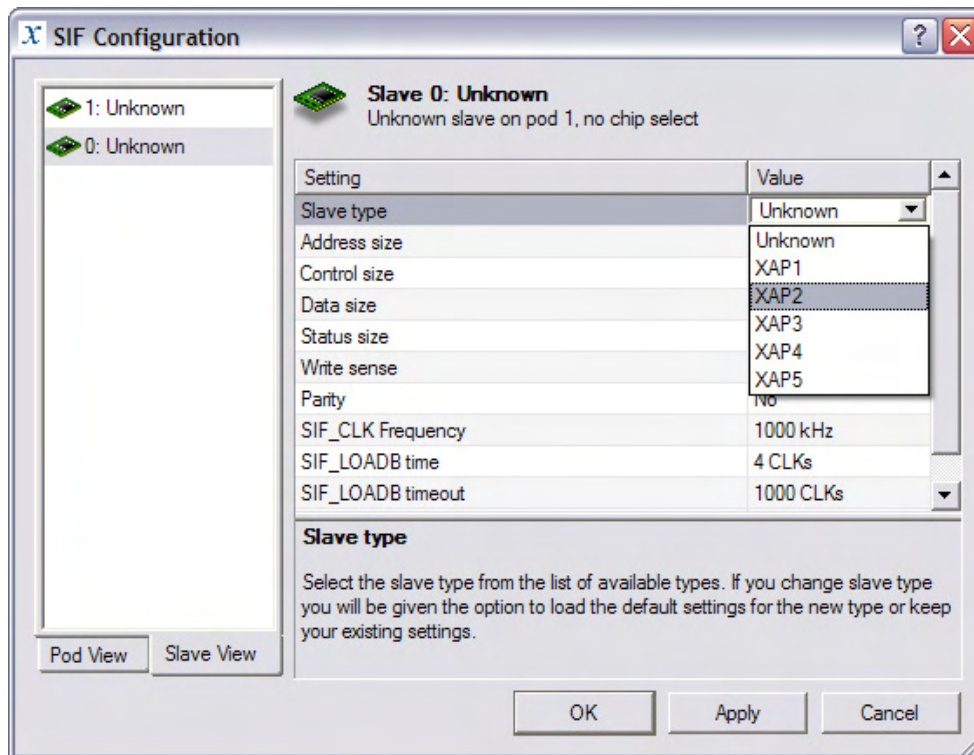


Figure 7. SIF Configuration dialog

Starting a debug session

When the project is properly configured, start a debug session by choosing **Debug | Attach** from the main menu. For details on using the debugger, see the xIDE for EM250 online help.

Note: The EM250 only supports a single hardware breakpoint.

The `Restart` option of the xIDE debugger only restarts the CPU core itself, leaving the EM250 peripherals configured and operating as they were. If this behavior is not desired and a full reset is needed, use the reset options in InSight Desktop or run `em2xx_load` without specifying a new image to load:

```
em2xx_load.exe -sid [IP Address] [-Run | -Reset]
```

Setting code optimization

The code optimization setting for an application build can determine how much debug information is available to xIDE. The optimization setting is configured in the Project Properties dialog (see Figure 8) with one of the following options:

- **Off:** Full source-level debugging capabilities; global and local variable watches.
- **Normal:** Source-level debug capability; only global variable watches, no local variables.
- **High:** Limited source-level debugging; only global variable watches, no local variables.

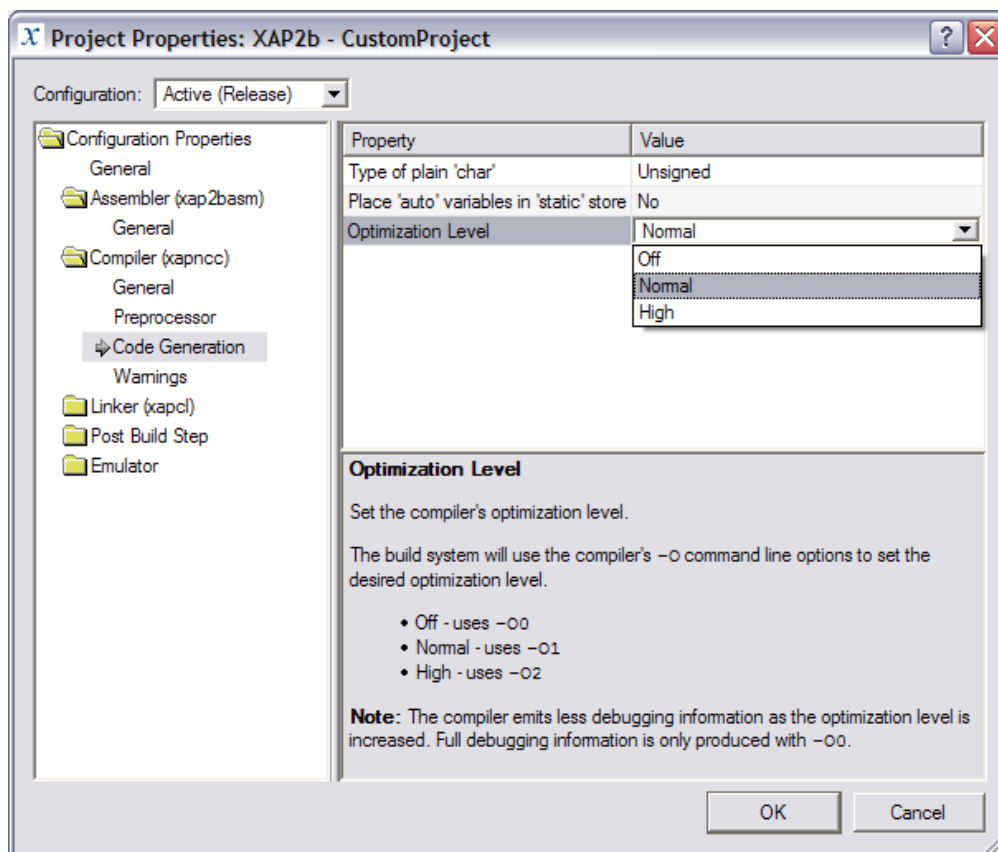


Figure 8. Project Properties dialog

After you change a project's optimization settings, you must rebuild the application and upload it again to the EM250 nodes. Note that the EmberZNet stack and other libraries are pre-built with a High optimization level.

After Reading This Document

If you have questions or require assistance with the procedures described in this document, please contact an Ember support representative at support@ember.com.

Copyright © 2008 by Ember Corporation

All rights reserved.

The information in this document is subject to change without notice. The statements, configurations, technical data, and recommendations in this document are believed to be accurate and reliable but are presented without express or implied warranty. Users must take full responsibility for their applications of any products specified in this document. The information in this document is the property of Ember Corporation.

Title, ownership, and all rights in copyrights, patents, trademarks, trade secrets and other intellectual property rights in the Ember Proprietary Products and any copy, portion, or modification thereof, shall not transfer to Purchaser or its customers and shall remain in Ember and its licensors.

No source code rights are granted to Purchaser or its customers with respect to all Ember Application Software. Purchaser agrees not to copy, modify, alter, translate, decompile, disassemble, or reverse engineer the Ember Hardware (including without limitation any embedded software) or attempt to disable any security devices or codes incorporated in the Ember Hardware. Purchaser shall not alter, remove, or obscure any printed or displayed legal notices contained on or in the Ember Hardware.

Ember, Ember Enabled, EmberZNet, InSight, and the Ember logo are trademarks of Ember Corporation.

All other trademarks are the property of their respective holders.

ember

Ember Corporation
47 Farnsworth Street
Boston, MA 02210
+1 (617) 951-0200
www.ember.com



wireless semiconductor solutions