LPCXpresso IDE User Guide

Rev. 8.1 — 15 February, 2016   **User guide**

15 February, 2016

Copyright © 2013-2016 NXP Semiconductors

1. Introduction to LPCXpresso

LPCXpresso is a low-cost microcontroller (MCU) development platform ecosystem from NXP, which provides an end-to-end solution enabling engineers to develop embedded applications from initial evaluation to final production.

The LPCXpresso platform ecosystem includes:

- The LPCXpresso IDE, a software development environment for creating applications for NXP's ARM-based "LPC" range of MCUs.

- The range of LPCXpresso development boards, each of which includes a built-in "LPC-Link", "LPC-Link2", or CMSIS-DAP debug probe. These boards are developed in collaboration with Embedded Artists.

- The standalone "LPC-Link2" debug probe.

This guide is intended as an introduction to using LPCXpresso, with particular emphasis on the LPCXpresso IDE. It assumes that you have some knowledge of MCUs and software development for embedded systems.

1.1 LPCXpresso IDE Overview of Features

The LPCXpresso IDE is a fully featured software development environment for NXP's ARM-based MCUs, and includes all the tools necessary to develop high-quality embedded software applications in a timely and cost effective fashion.

The LPCXpresso IDE is based on the Eclipse IDE and features many ease-of-use and MCU specific enhancements. The LPCXpresso IDE also includes the industry standard ARM GNU toolchain, providing professional quality development tools at low cost. The fully featured debugger supports both SWD and JTAG debugging, and features direct download to on-chip flash.

For the latest details on new features and functionality, visit http://www.lpcware.com/content/forum/lpcxpresso-latest-release

1.1.1 Summary of Features

- Complete C/C++ integrated development environment

  - Latest Eclipse-based IDE with many ease-of-use enhancements

    - Eclipse Mars (v4.5) and CDT (8.8)

  - The IDE can be further enhanced with Eclipse plugins

    - CVS source control built in; Subversion, TFS, Git, and others available for download

  - Command-line tools included for integration into build, test, and manufacturing systems

- Industry standard GNU toolchain (v5) including

  - C and C++ compilers, assembler, and linker

  - Converters for SREC, HEX, and binary

- Advanced project wizards

  - Simple creation of preconfigured applications for specific MCUs

- Device-specific support for NXP's ARM-based MCUs (including Cortex-M, ARM7 and ARM9 based parts)

- Automatic generation of linker scripts for correct placement of code and data into flash and RAM

- Automatic generation of MCU-specific startup and device initialization code

- No assembler required with Cortex-M MCUs

- Advanced multicore support

  - Provision for creating linked projects for each core in multicore MCUs

  - Debugging of multicore projects within a single IDE instance, with the ability to link various debug views to specific cores

- Fully featured debugger supporting JTAG and SWD connection

  - Built-in optimized flash programming for internal and SPI flash

  - High-level and instruction-level debug

  - Views of CPU registers and on-chip peripherals

  - Support for multiple devices on the JTAG scan-chain

- Library support

  - Redlib: a small-footprint embedded C library

  - Newlib: a complete C and C++ library

  - NewlibNano: a new small-footprint C and C++ library, based on Newlib

  - LPCOpen MCU software libraries

  - Cortex Microcontroller Software Interface Standard (CMSIS) libraries and source code

- Trace functionality

  - Instruction trace via Embedded Trace Buffer (ETB) on certain Cortex-M3/M4 based MCUs or via Micro Trace Buffer (MTB) on Cortex-M0+ based MCUs

    - Providing a snapshot of application execution with linkage back to source, disassembly and profile

  - SWO Trace on Cortex-M3/M4 based MCUs when debugging via LPC-Link2, providing functionality including:

    - Profile tracing

    - Interrupt tracing

    - Datawatch tracing

    - Printf over ITM

- Power Measurement

- On enabled boards, sample power usage at adjustable rates of up to 200 ksps; average power usage display option

- Explore detailed plots of collected data in the IDE

- Export data for analysis with other tools

- RedState — state machine designer and code generator

  - Graphically design your state machines

  - Generates standard C code

  - Configures NXP State Configurable Timer (SCT) as well as supporting software state machines

1.1.2 Supported Debug Probes

The following debug probes are supported by the LPCXpresso IDE for general debug connections:

- LPC-Link (the original LPCXpresso board)

- LPC-Link2 (with "CMSIS-DAP" firmware) - either the standalone debug probe or the version built into LPCXpresso V2 and V3 boards

- CMSIS-DAP-enabled debug probes, such as LPC800-MAX, LPCXpresso824-MAX, LPCXpresso1769/CD, Keil ULINK-ME etc.

- Older debug probes:

  - Red Probe / Red Probe+

  - RDB1768 development board built-in debug connector (RDB-Link)

  - RDB4078 development board built-in debug connector

For more information on debug probe support in the LPCXpresso IDE, visit http://www.lpcware.com/content/faq/lpcxpresso/supported-debug-probes-mcus

Support for GDB server-based debug connections is also provided. This feature enables support for third-party debug probes, such as Segger J-Link. When debugging with GDB server connections, some functionality may be reduced.

- For more information on using Segger J-Link with LPCXpresso, visit http://www.segger.com/nxp-lpcxpresso.html

1.1.3 LPCXpresso Development Boards

A major part of the LPCXpresso platform is the range of LPCXpresso boards that work seamlessly with the LPCXpresso IDE. These boards provide practical and easy-to-use development hardware to use as a starting point for your LPC Cortex-M MCU based projects.

LPCXpresso IDE User Guide -

User Guide Rev. 8.1 — 15 February, 2016 3
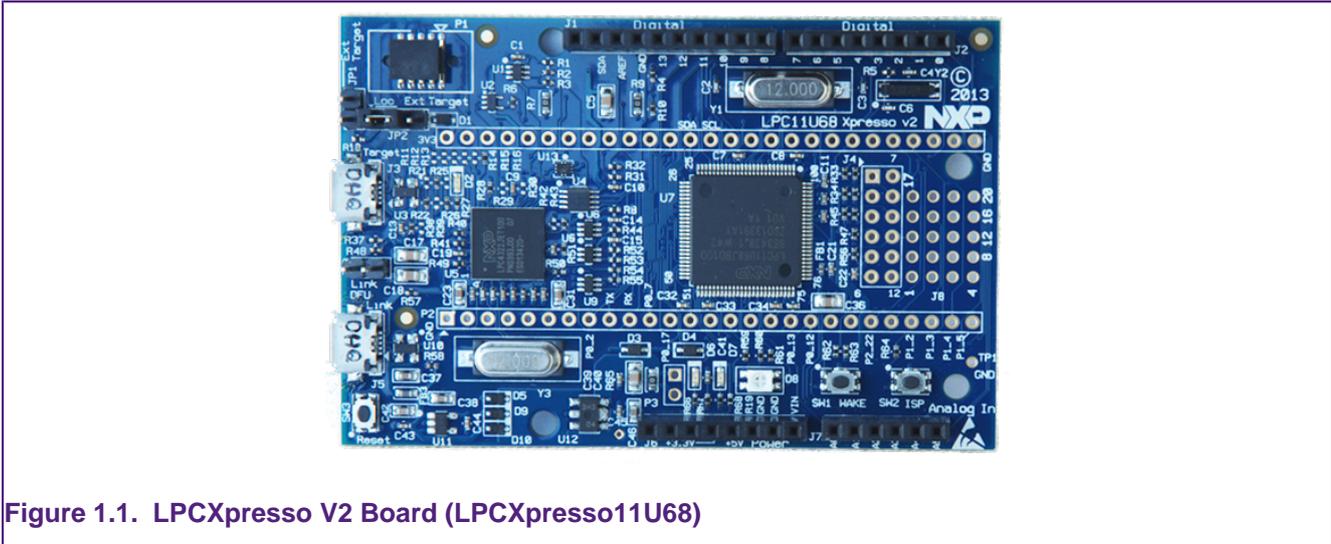
Figure 1.1. LPCXpresso V2 Board (LPCXpresso11U68)
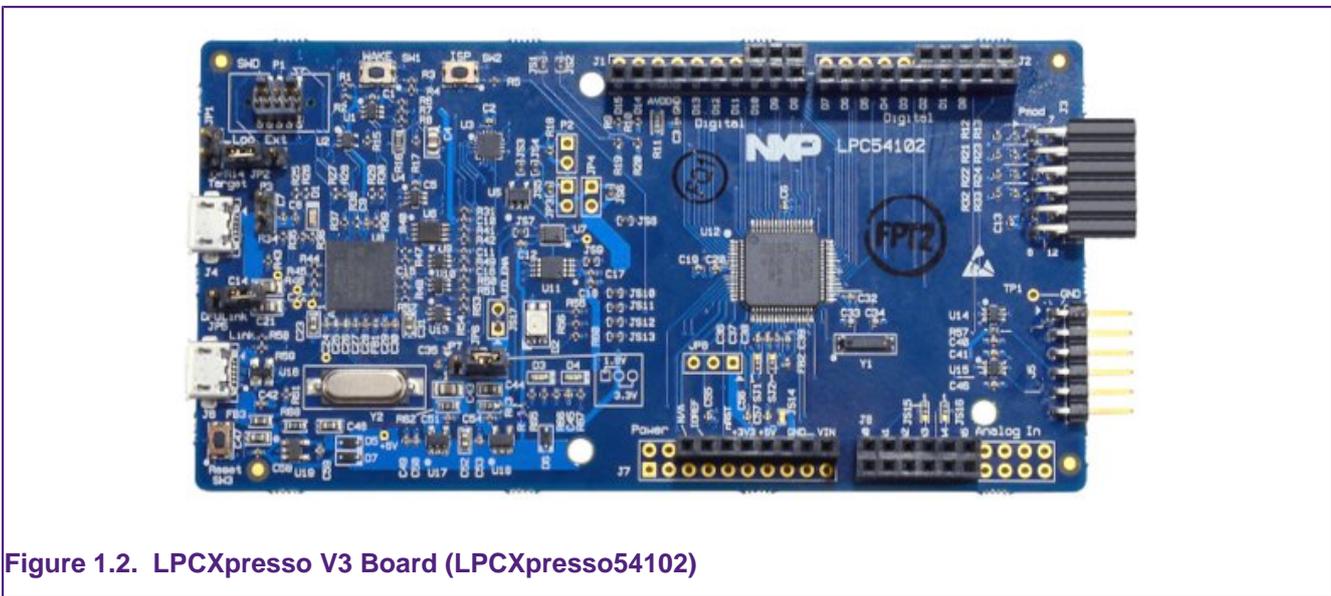
Figure 1.2. LPCXpresso V3 Board (LPCXpresso54102)

For more information, visit:

http://www.lpcware.com/LPCXpressoBoards

2. LPCXpresso IDE Overview

2.1 Documentation and Help

The LPCXpresso IDE is based on the Eclipse IDE framework, and many of the core features are described well in generic Eclipse documentation and in the help files to be found on the LPCXpresso IDE's **Help -> Help Contents** menu. That also provides access to the LPCXpresso User Guide (this document), as well as the documentation for the compiler, linker, and other underlying tools.

LPCXpresso IDE documention comprises a suite of documents including:

- LPCXpresso IDE Installation and Licensing Guide

- LPCXpresso IDE User Guide

- LPCXpresso IDE SWO Trace Guide

- LPCXpresso IDE Instruction Trace Guide

- LPCXpresso IDE Power Measurement Guide

- LPCXpresso IDE Red State Guide

To obtain assistance on using LPCXpresso, visit

http://lpcware.com/lpcxpresso/support

2.2 Workspaces

When you first launch LPCXpresso IDE, you will be asked to select a Workspace, as shown in Figure 2.1.
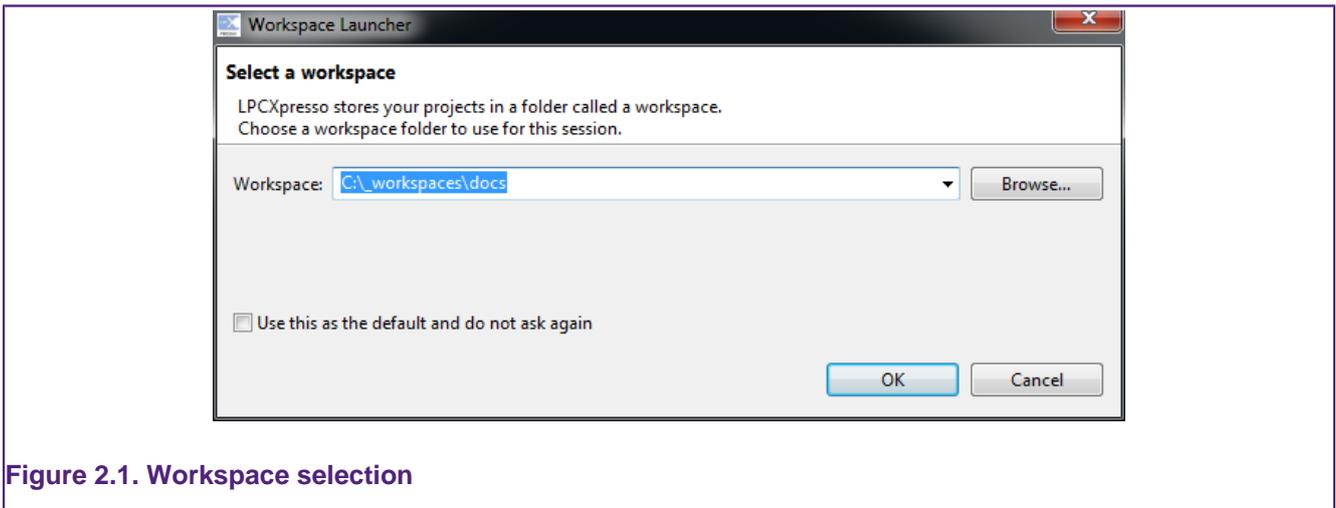
Figure 2.1. Workspace selection

A Workspace is simply a directory that is used to store the projects you are currently working on. Each Workspace can contain multiple projects, and you can have multiple Workspaces on your computer. The LPCXpresso IDE can only access a single Workspace at a time, although it is possible to run multiple instances in parallel — with each instance accessing a different Workspace.

If you tick the **Use this as the default and do not ask again** option, then the LPCXpresso IDE will always start up with the chosen Workspace opened; otherwise, you will always be prompted to choose a Workspace.

It is possible to change the Workspace that the LPCXpresso IDE is using, via the **File -> Switch Workspace** option.

2.3 Perspectives and Views

The overall layout of the main LPCXpresso IDE window is known as a Perspective. Within each Perspective are many sub-windows, called Views. A View displays a particular set of data in the LPCXpresso environment. For example, this data might be source code, hex dumps, disassembly, or memory contents. Views can be opened, moved, docked, and closed, and the layout of the currently displayed Views can be saved and restored.

Typically, the LPCXpresso IDE operates using the single **Develop Perspective**, under which both code development and debug sessions operate as shown in Figure 2.3. This single perspective simplifies the Eclipse environment, but at the cost of slightly reducing the amount of information displayed on screen.

Alternatively the LPCXpresso IDE can operate in a "dual Perspective" mode such that the **C/C++ Perspective** is used for developing and navigating around your code and the **Debug Perspective** is used when debugging your application.

You can manually switch between Perspectives using the Perspective icons in the top right of the LPCXpresso IDE window, as shown in Figure 2.2.
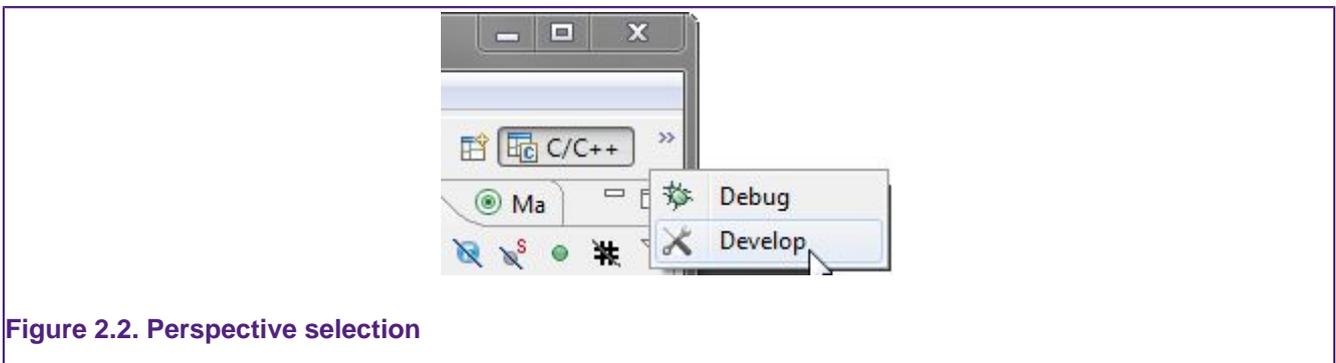
Figure 2.2. Perspective selection

All Views in a Perspective can also be rearranged to match your specific requirements by dragging and dropping. If a View is accidentally closed, it can be restored by selecting it from the **Window -> Show View** dialog. The default layout for a perspective can be restored at any time via **Window -> Reset Perspective**.

2.4 Major Components of the Develop Perspective
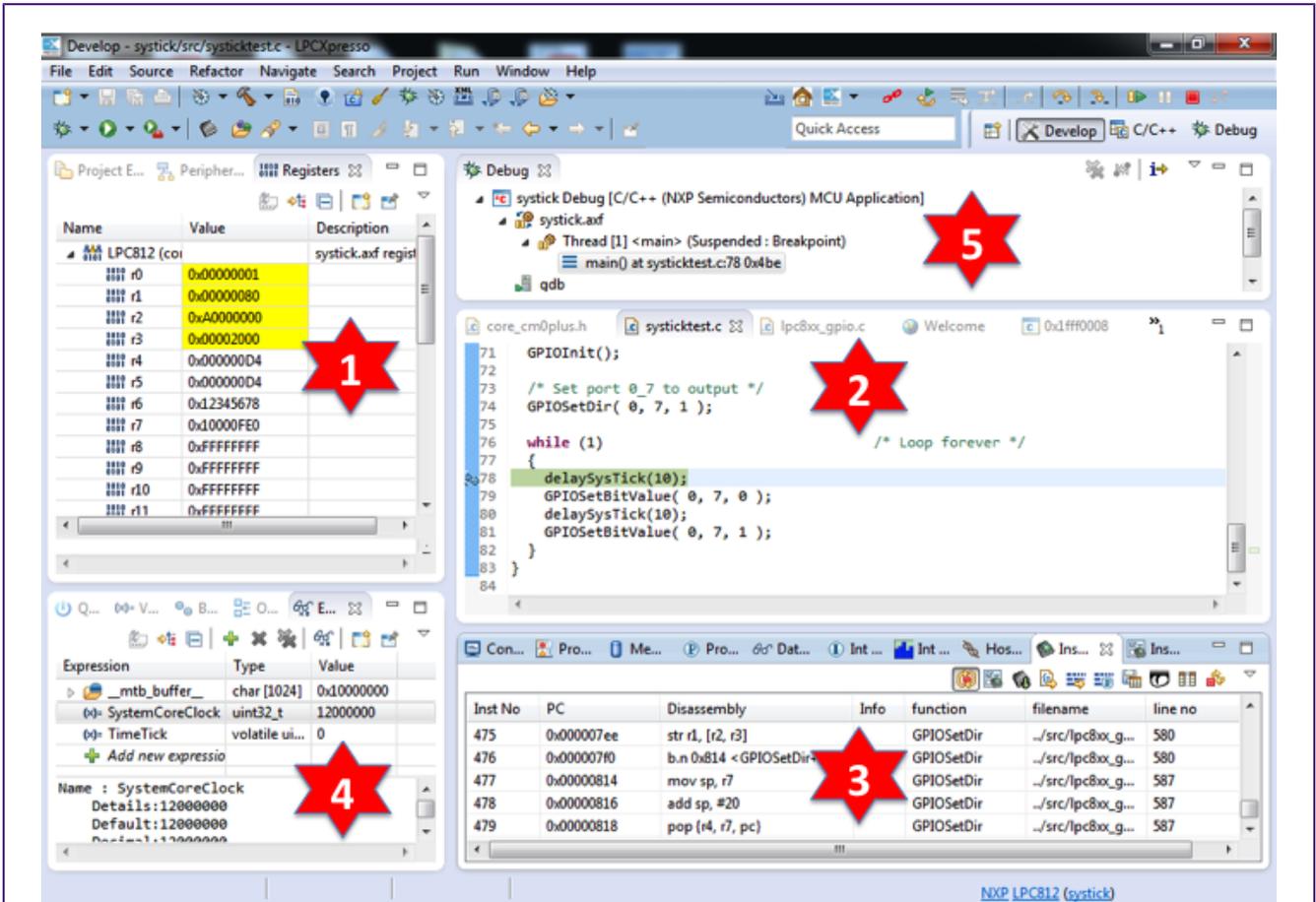
Figure 2.3. Develop Perspective (whilst debugging)

1. Project Explorer / Peripherals / Registers Views

   • The **Project Explorer** gives you a view of all the projects in your current Workspace.

   • When debugging, the **Peripherals** view allows you to display the registers within Peripherals.

   • When debugging, the **Registers** view allows you to display the registers within the CPU of your MCU.

   • Not visible here is the **Symbol Viewer**; this view displays symbolic information from a referenced .axf file.

2. Editor

   • On the upper right is the editor, which allows modification and saving of source code. When debugging, this is where you can see the code you are executing and can step from line to line. By pressing the `'i->'` icon at the top of the Debug view, you can switch to stepping by assembly instruction. Clicking in the left margin will set and delete breakpoints.

3. Console / Problems / Trace Views / Power Measurement

- On the lower right are the Console and Problems Views. The Console View displays status information on compiling and debugging, as well as semihosted program output.

- The Problems View (available by changing tabs) shows all compiler errors and will allow easy navigation to the error location in the Editor View.

- Sitting in parallel with the Console View are the various Views that make up the Trace functionality of LPCXpresso IDE. For more information on Trace functionality, please see the LPCXpresso IDE SWO Trace Guide and/or the LPCXpresso IDE Instruction Trace Guide.

  - The SWO trace Views allow you to gather and display runtime information using the SWV technology that is part of Cortex-M3/M4 based parts.

  - On some MCUs, you can also view instruction trace data downloaded from the MCU's Embedded Trace Buffer (ETB) or Micro Trace Buffer (MTB). The example here shows instruction trace information downloaded from an LPC812's MTB.

- Sitting in parallel with the Console View is the Power Measurement View, a dedicated trace View capable of displaying real-time target power usage. For more information please see the LPCXpresso IDE Power Measurement Guide.

4. Quickstart / Variables / Breakpoints / Outlines / Expressions Views

- On the lower left of the window, the **Quickstart Panel View** has fast links to commonly used features. This is the best place to go to find options such as Build, Debug, and Import.

- Sitting in parallel to the Quickstart Panel, the **Variable** View allows you to see the values of local variables.

- Sitting in parallel to the Quickstart Panel, the **Breakpoint** View allows you to see and modify currently set breakpoints.

- Sitting in parallel to the Quickstart Panel, the **Outline** View allows you to quickly find components of the source file with input focus within the editor.

- Sitting in parallel to the Quickstart Panel, the **Expressions** View allows you to add global variables and other expressions so that you can see and modify their values.

5. Debug View

- The Debug View appears when you are debugging your application. This shows you the stack trace. In the "stopped" state, you can click on any particular function and inspect its local variables in the Variables tab (which is located parallel to the **Quickstart Panel View**).

LPCXpresso IDE User Guide -

User Guide Rev. 8.1 — 15 February, 2016 8

3. Importing and Debugging Example Projects

3.1 Software Drivers and Examples

LPCOpen is now the preferred software platform for most NXP Cortex-M based MCUs, replacing the various CMSIS / Peripheral Driver Library / code bundle software packages made available in the past. LPCOpen has been designed to allow you to quickly and easily utilize an extensive array of software drivers and libraries in order to create and develop multifunctional products. Amongst the features of LPCOpen are:

- MCU peripheral device drivers with meaningful examples

- Common APIs across device families

- Commonly needed third party and open source software ports

- Support for Keil, IAR and LPCXpresso toolchains

LPCOpen is thoroughly tested and maintained. The latest LPCOpen software now available provides:

- MCU family-specific download package

- Support for USB ROM drivers

- Improved code organization and drivers (efficiency, features)

- Improved support for the LPCXpresso IDE

CMSIS / Peripheral Driver Library / code bundle software packages are still available, both within your LPCXpresso IDE install directory in **\lpcxpresso\Examples\NXP** and also downloadable from the NXP LPCware website. But generally these should only be used for existing development work. When starting a new evaluation or product development, we would recommend the use of LPCOpen.

More information on LPCOpen together with package downloads can be found at:

http://www.lpcware.com/lpcopen

3.2 Importing an Example Project

The **Quickstart Panel View** provides rapid access to the most commonly used features of the LPCXpresso IDE. Using the Quickstart Panel, you can quickly import example projects, create new projects, build projects, and debug projects.
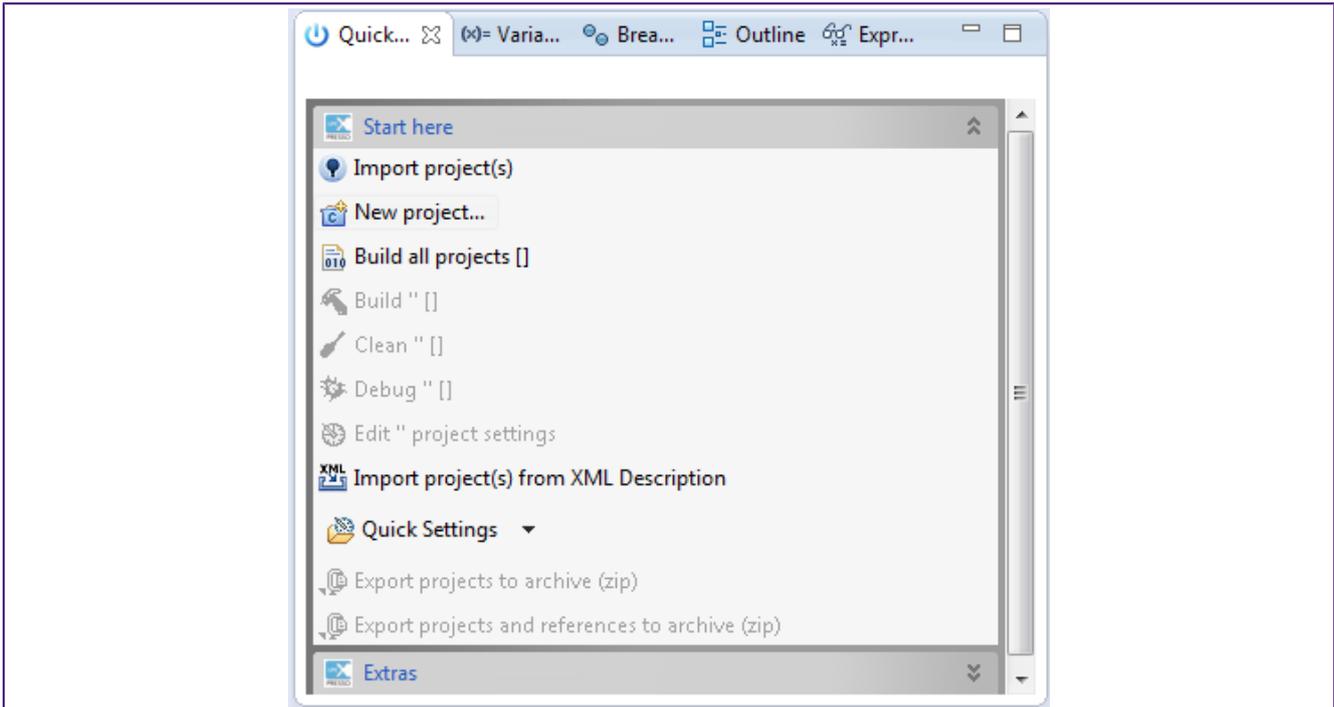
Figure 3.1. Importing project(s)

On the **Quickstart Panel**, click on the "Start here" sub-panel, and click on Import project(s).
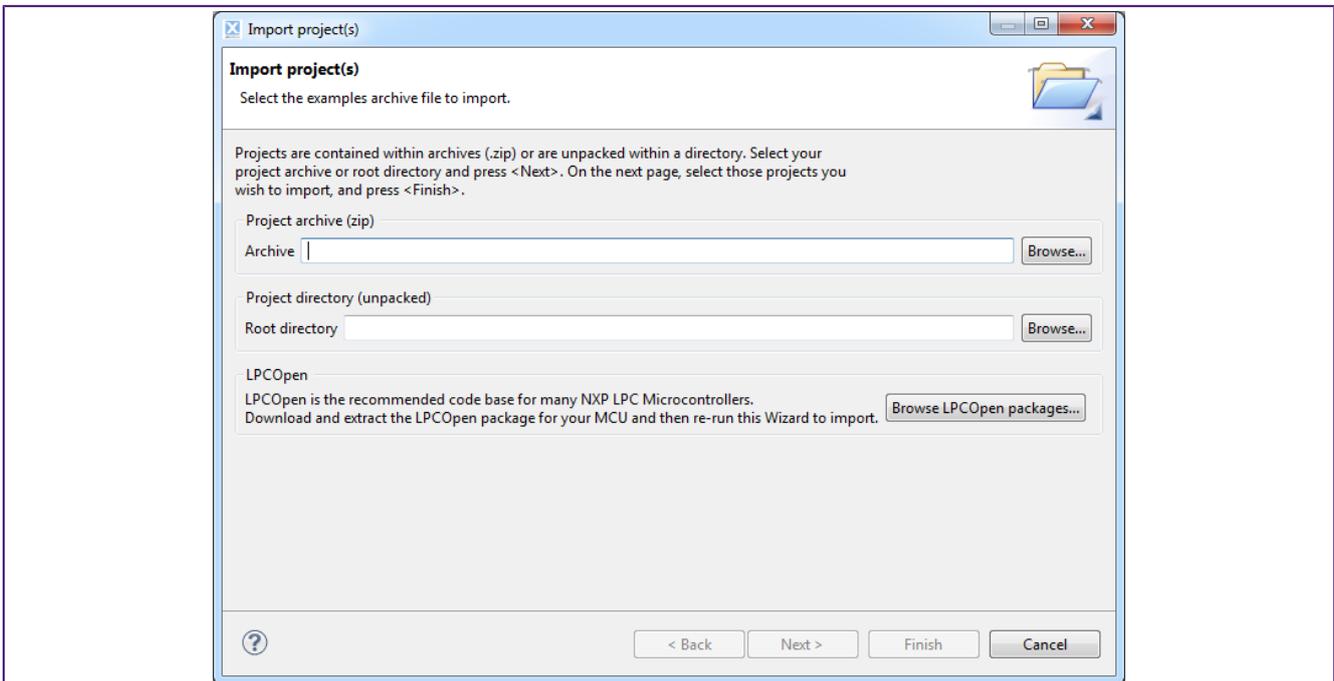
Figure 3.2. Importing examples

As shown in Figure 3.2, from the first page of the wizard you can:

- **Browse** to locate Examples stored in zip archive files on your local system. These could be archives that you have previously downloaded (for example LPCOpen packages from the NXP LPCware website or the supplied, but deprecated, sample code bundles located within the Examples subdirectory of your LPCXpresso IDE installation).

- **Browse** to locate projects stored in directory form on your local system (for example, you can use this to import projects from a different Workspace into the current Workspace).

- **Browse LPCOpen** packages to visit LPCware and download an appropriate LPCOpen package for your target MCU. This option will automatically open a web browser onto a suitable links page on the NXP LPCware website.

To demonstrate how to use the Import Project(s) functionality, we will now import the LPCOpen examples for the LPCXpresso4337 development board.

3.2.1 Importing Examples for the LPCXpresso4337 Development Board

First of all, assuming that you have not previously downloaded the appropriate LPCOpen package, click on **Browse LPCOpen Packages**, which will open a web browser window. Click on **Download LPCOpen Packages**, and then the link to **LPCOpen v2.xx for LPC43xx family devices**, and then choose the download for the LPCXpresso4337 board.

Once the package has downloaded, return to the Import Project(s) dialog and click on the **Browse** button next to **Project archive (zip)**; then locate the LPCOpen LPCXpresso4337 package archive previously downloaded. Select the archive, click **Open** and then click **Next**. You will then be presented with a list of projects within the archive, as shown in Figure 3.3.
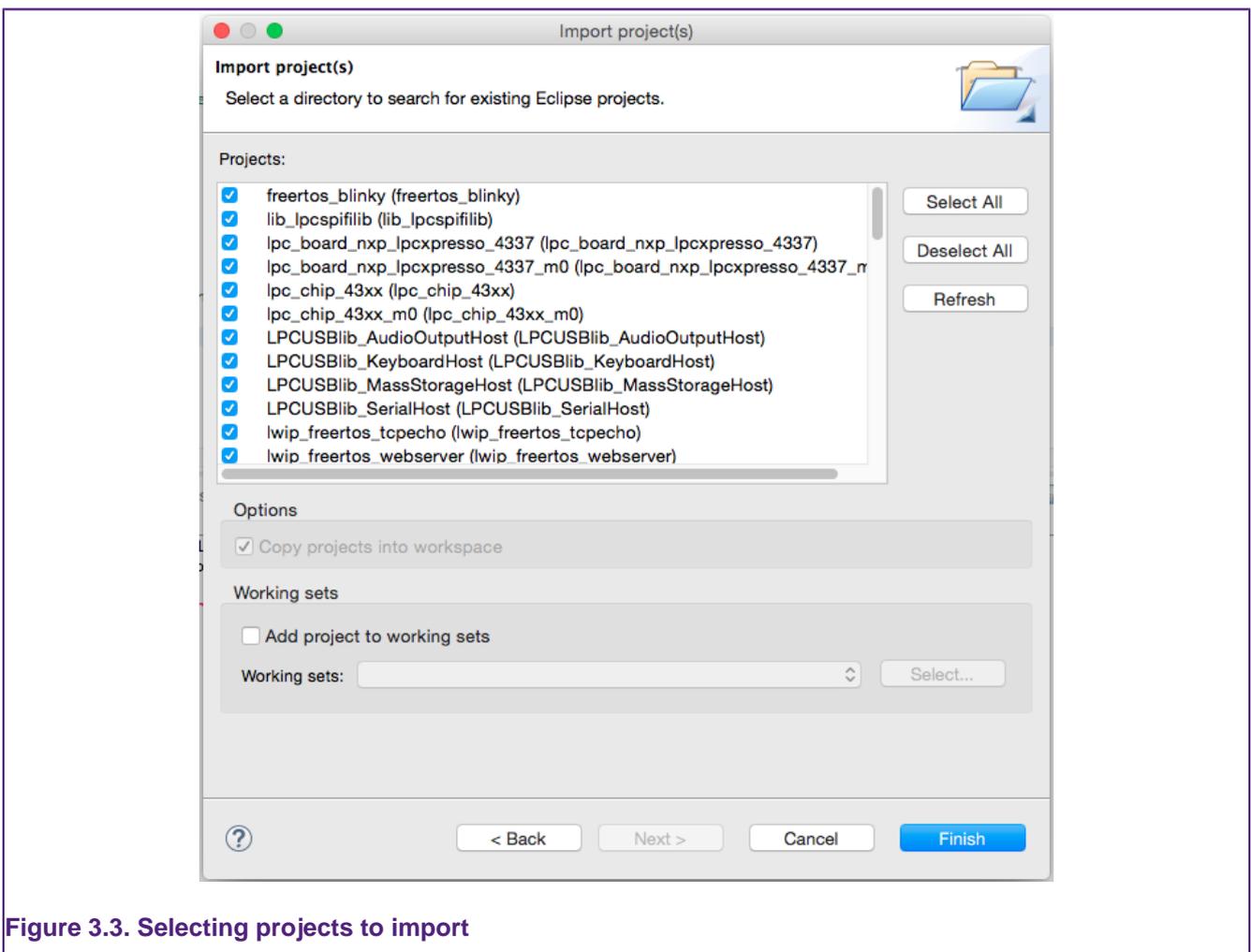
Figure 3.3. Selecting projects to import

Select the projects you want to import and then click **Finish**. The examples will be imported into your Workspace.

Note that generally it is a good idea to leave all projects selected when doing an import from a zip archive file of examples. This is certainly true the first time you import an example set, when you will not necessarily be aware of any dependencies between projects. In most cases, an archive of projects will contain one or more library projects, which are used by the actual application projects within the examples. If you do not import these library projects, then the application projects will fail to build.

3.3 Building Projects

Building the projects in a workspace is a simple case of using the **Quickstart Panel** to "Build all projects". Alternatively a single project can be selected in the Project Explorer View and built. **Note** that building a single project may also trigger a build of any associated library projects.

3.3.1 Build Configurations

By default, each project will be created with two different "build configurations": **Debug** and **Release**. Each build configuration will contain a distinct set of build options. Thus a **Debug** build will typically compile its code with optimizations disabled (`-O0`) and **Release** will compile its code optimizing for minimum code size (`-Os`). The currently selected build configuration for a project will be displayed after its name in the Quickstart Panel's Build/ Clean/Debug options.

For more information on switching between build configurations, see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/change-build-config

3.4 Debugging a Project

This description shows how to run the LPCOpen `periph_blinky` example application for the LPCXpresso4337 development board. The same basic principles will apply for other examples and boards.

First of all you need to ensure that your LPCXpresso development board is connected to your computer. **Note** that some LPCXpresso development boards have two USB connectors fitted. Make sure that you have connected the lower connector marked DFU-Link.

To start debugging `periph_blinky` on your target, simply highlight the project in the Project Explorer, and then in the **Quickstart Panel** click on **Start Here** and select **Debug 'periph_blinky'**, as in Figure 3.4. Click on **OK** to continue with the download and debugging of the "Debug" build of your project. The project binary will then be built if required, and then automatically downloaded to the target and programmed into flash memory. A default breakpoint will be set on the first instruction in `main()`, the application will be started (by simulating a processor reset), and code will be executed until the default breakpoint is hit.
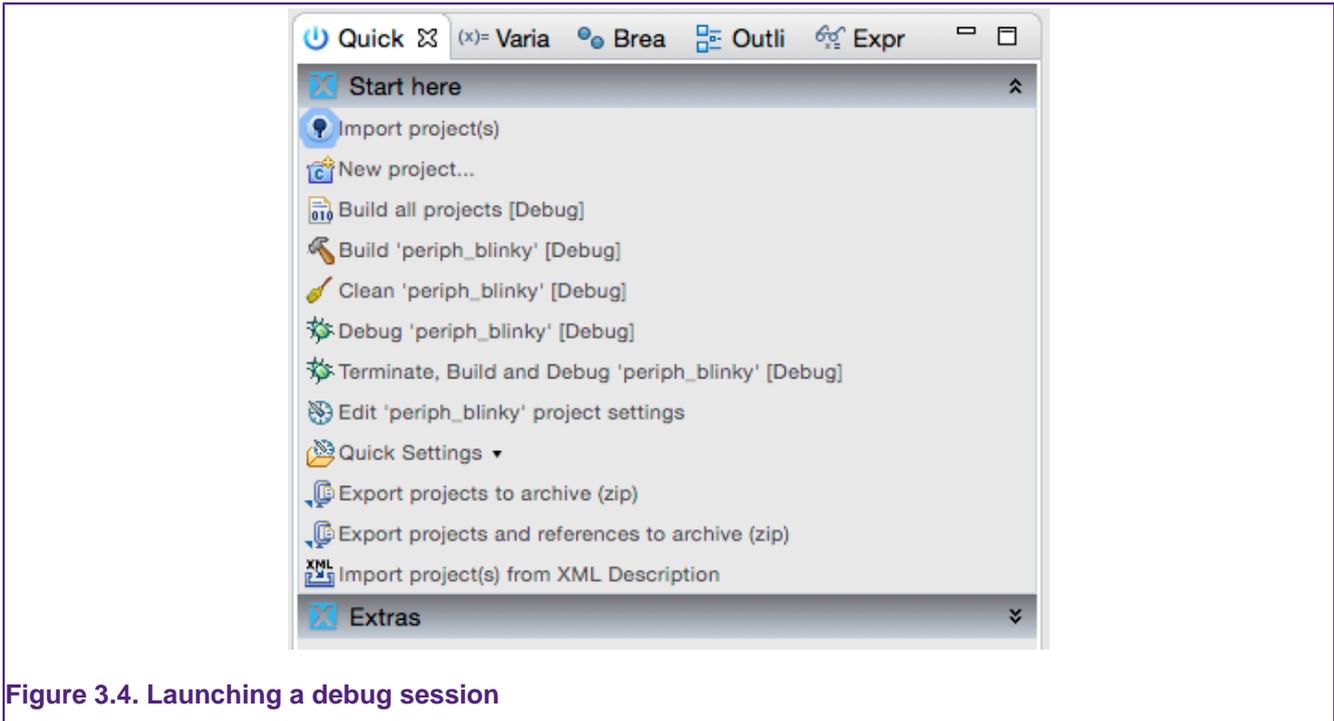
Figure 3.4. Launching a debug session

3.4.1 Debug Emulator Selection Dialog

The first time you debug a project, the Debug Emulator Selection Dialog will be displayed, showing all supported probes that are attached to your computer. In the example shown in Figure 3.5, an LPC-Link2 and an LPCXpressoCD board have been found.
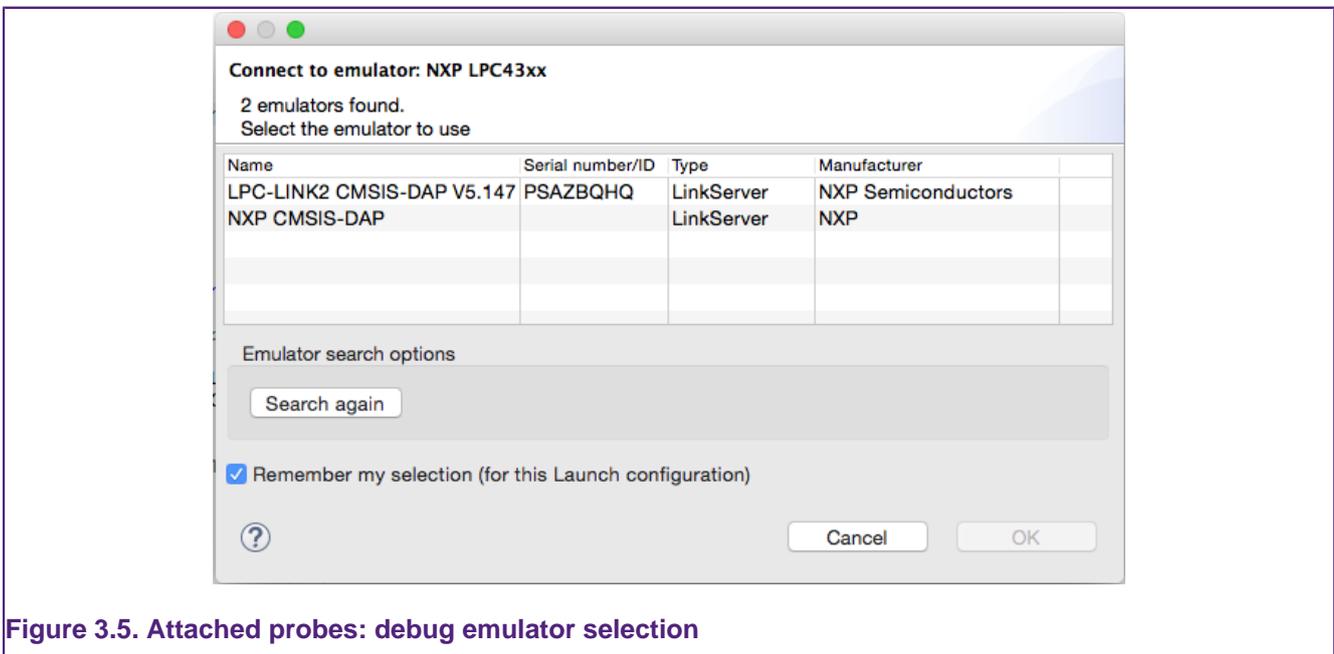
Figure 3.5. Attached probes: debug emulator selection

Note New from LPCXpresso IDE version 8.1 is support for unique LPC-Link2 probe identification. When using the latest LPC-Link2 CMSIS-DAP firmware, each debug probe will report its ID, in this example **PSAZBQHQ**. Using this scheme a particular probe can be recognized as associated with a particular project.

You now need to select the probe through which you wish to debug. In Figure 3.6 the LPC-Link2 has been selected, which is the correct option to debug an LPCXpresso4337 board.
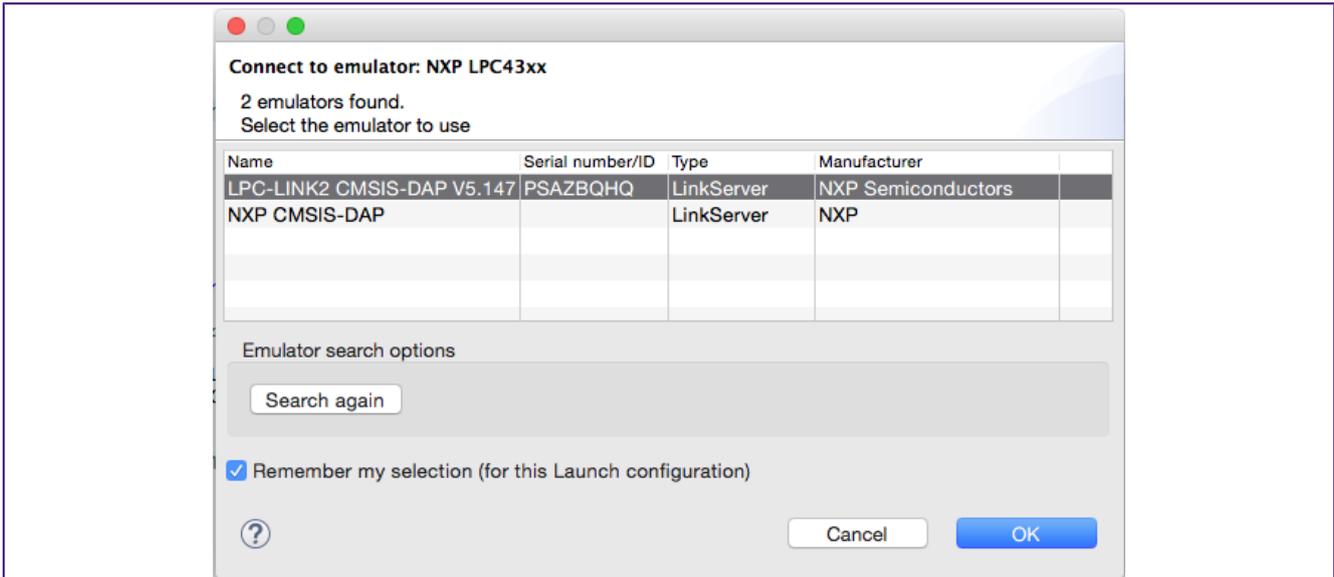
Figure 3.6. LPC-Link2 selected

For any future debug sessions, the stored probe selection will be automatically used, unless the probe cannot be found. In Figure 3.7 the previously selected LPC-Link2 is no longer connected.
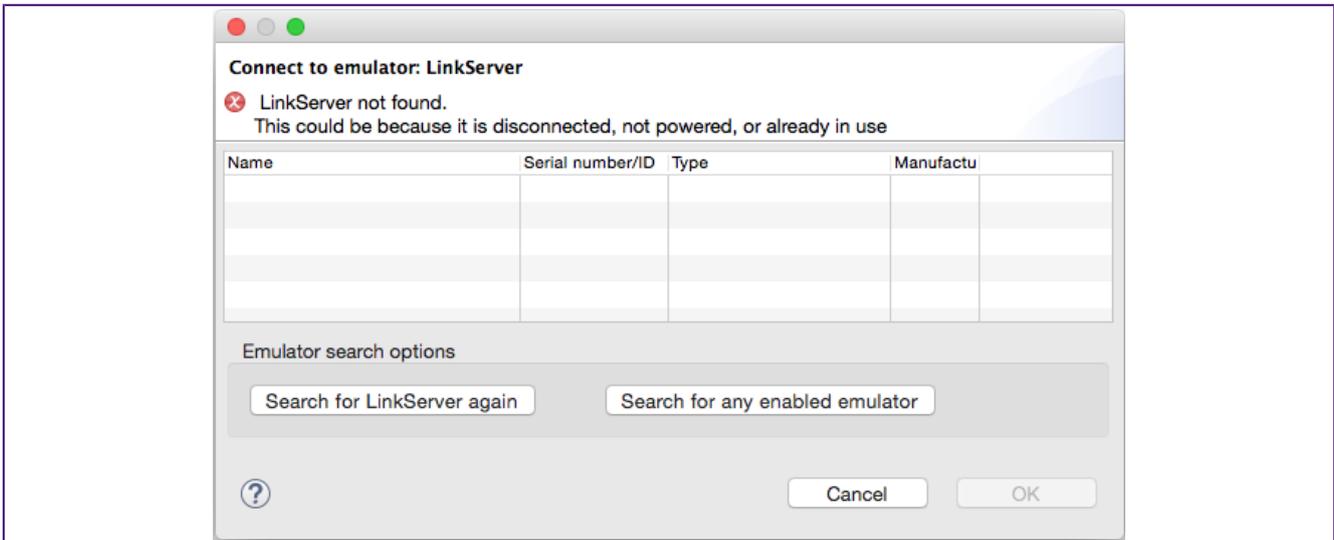
Figure 3.7. LPC-Link2 no longer connected

This might have been because you had forgotten to connect the probe, in which case connect it to your computer and select **Search again**.

The tools will then go and search for appropriate emulators. In Figure 3.8 LPC-Link2 has been detected again.
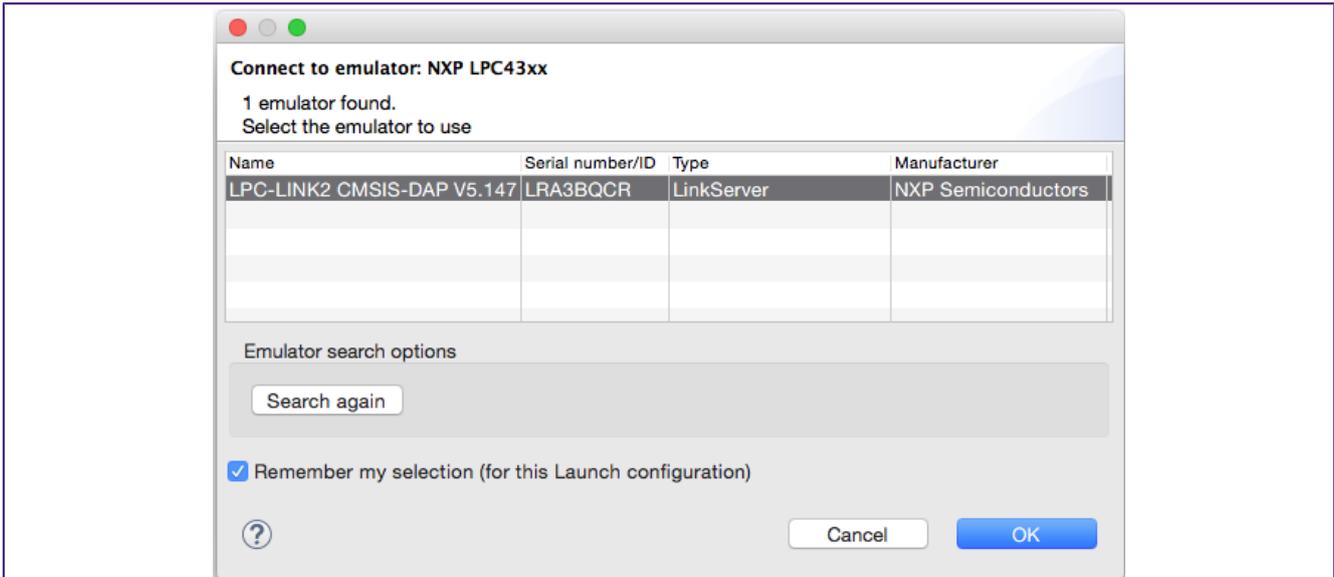
Figure 3.8. Attached probes — LPC-Link2

Notes:

- The "Remember my selection" option is enabled by default in the Debug Emulator Selection Dialog, and will cause the selected probe to be stored in the launch configuration for the current configuration (typically Debug or Release) of the current project. You can thus remove the probe selection at any time by simply deleting the launch configuration.

- You will need to select a probe for each project that you debug within a Workspace (as well as for each configuration within a project).

- Storing the selected emulator (probe) in the debug launch configuration helps to improve debug startup time.

3.4.2 Controlling Execution

When you have started a debug session and (if necessary) selected the appropriate probe to connect to, your application is automatically downloaded to the target, a default breakpoint is set on the first instruction in `main()`, the application is started (by simulating a processor reset), and code is executed until the default breakpoint is hit.

Program execution can now be controlled using the common debug control buttons, as listed in Table 3.1, which are displayed on the global toolbar. The call stack is shown in the Debug View, as in Figure 3.9.
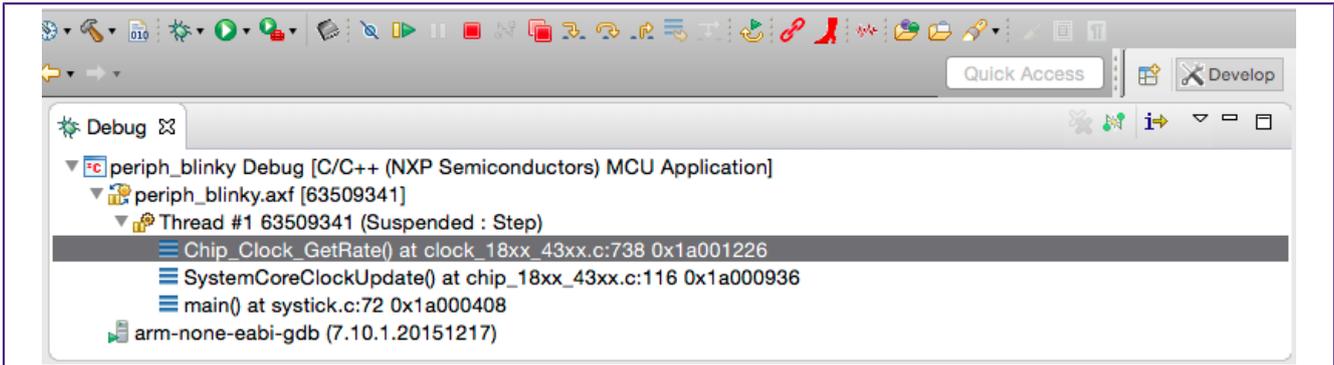
Figure 3.9. Debug controls

Table 3.1. Program execution controls

Button	Description	Keyboard Shortcut
	Restart program execution (from reset)	
	Run/Resume the program	F8
	Step over a C/C++ line	F6
	Step into a function	F5
	Return from a function	F7
	Stop the debugger	Ctrl + F2
	Pause Execution of the running program	
	Show disassembled instructions	

Setting a breakpoint

To set a breakpoint, simply double-click on the margin area of the line on which you wish to set the breakpoint (before the line number).

Restarting the application

If you hit a breakpoint or pause execution and want to start execution of the application from the beginning again, you can do this using the **Restart** button.

Stopping debugging

To stop debugging just press the **Stop** button.

If you are debugging using the **Debug Perspective**, then to switch back to the **C/C++ Perspective** when you stop your debug session, just click on the **C/C++** tab in the upper right area of the LPCXpresso IDE (as shown in Figure 2.2).

4. Creating Projects using the Wizards

The LPCXpresso IDE includes many project templates to allow the rapid creation of correctly configured projects for specific MCUs.

4.1 Creating a Project using a Wizard

Click on the **New project...** option in the **Start here** tab of the **Quickstart Panel** to open the Project Creation Wizard. Now select the MCU family for which you wish to create a new project. **Note** that in some cases a number of families of MCUs are grouped together at a top level – just open the appropriate "expander" to get to the specific part family that you require. For example, the top-level group " LPC11 / LPC12" is used to hold all LPC11 and LPC12 part families, as shown in Figure 4.1.
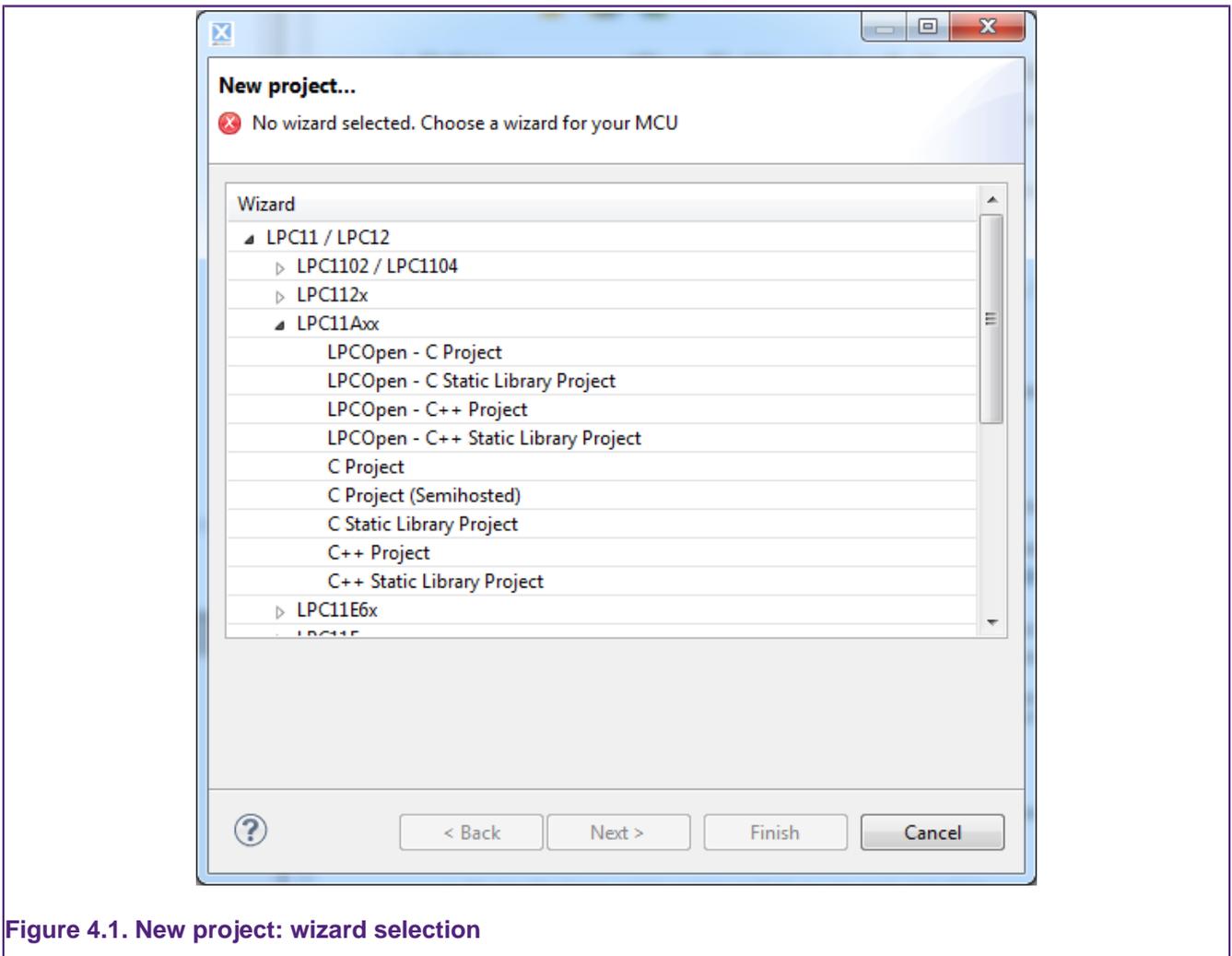
Figure 4.1. New project: wizard selection

You can now select the type of project that you wish to create.

4.1.1 Selecting the Wizard Type

For most MCU families the LPCXpresso IDE provides wizards for two forms of project: LPCOpen and non-LPCOpen. For more details on LPCOpen, see Software drivers and examples [9] . For both kinds, the main wizards available are:

C Project

- Creates a simple C project, with the `main()` routine consisting of an infinite `while(1)` loop that increments a counter.

- For LPCOpen projects, code will also be included to initialize the board and enable a LED.

C++ Project

- Creates a simple C++ project, with the `main()` routine consisting of an infinite `while(1)` loop that increments a counter.

- For LPCOpen projects, code will also be included to initialize the board and enable a LED.

C Static Library Project

- Creates a simple static library project, containing a source directory and, optionally, a directory to contain include files. The project will also contain a "liblinks.xml" file, which can be used by the smart update wizard on the context-sensitive menu to create links from application projects to this library project. For more details, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/creating-linking-library-projects

C++ Static Library Project

- Creates a simple (C++) static library project, like that produced by the C Static Library Project wizard, but with the tools set up to build C++ rather than C code.

The non-LPCOpen wizard families also include a further wizard:

Semihosting C Project

- Creates a simple "Hello World" project, with the `main()` routine containing a `printf()` call, which will cause the text to be displayed within the Console View of the LPCXpresso IDE. This is implemented using "semihosting" functionality. For more details, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/using-printf

4.1.2 Configuring the Project

Once you have selected the appropriate project wizard, you will be able to enter the name of your new project.

Then you will need to select the actual MCU that you will be targeting for this project. It is important that the MCU you select matches the MCU that you will be running your application on. This ensures that appropriate compiler and linker options will be used for the build, as well as correctly setting up the memory description, the flash driver (when possible), and the debug connection.

Finally you will be presented with one or more "Options" pages that provide the ability to set a number of project-specific options. The choices presented will depend upon which MCU you are targeting and the specific wizard you selected, and may also change between versions of the LPCXpresso IDE. **Note** that if you have any doubts over any of the options, then we would normally recommend leaving them set to their default values.

The following sections detail some of the options that you may see when running through a wizard.

4.2 Wizard Options

4.2.1 LPCOpen Library Project Selection

When creating an LPCOpen-based project, the first option page that you will see is the LPCOpen library selection page.
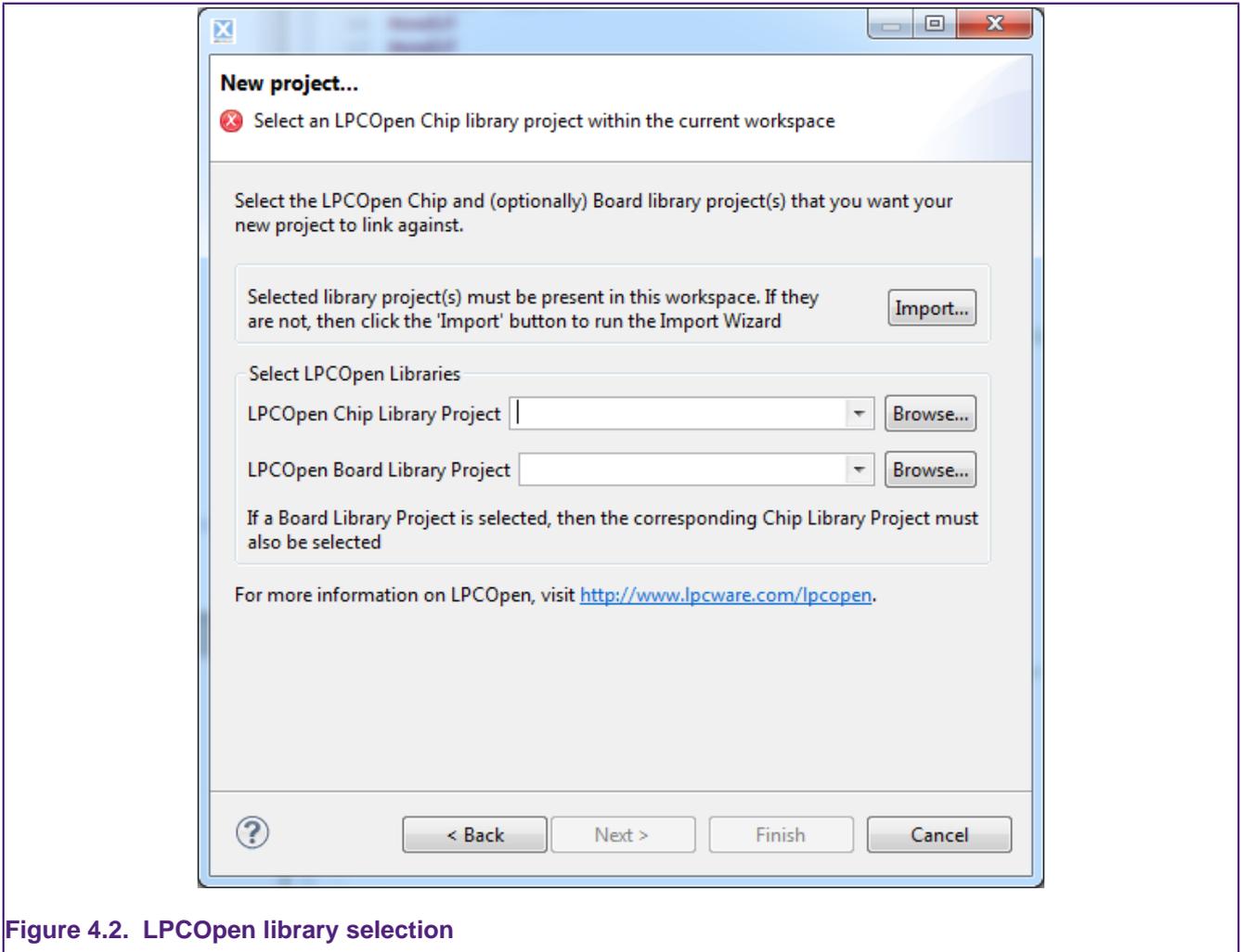
Figure 4.2. LPCOpen library selection

This page allows you to run the "Import wizard" to download the LPCOpen bundle for your target MCU/board from the NXP LPCware website and import it into your Workspace, if you have not already done so.

You will then need to select the LPCOpen Chip library for your MCU using the Workspace browser (and for some MCUs an appropriate value will also be available from the dropdown next to the Browse button). **Note** that the wizard will not allow you to continue until you have selected a library project that exists within the Workspace.

Finally, you can optionally select the LPCOpen Board library for the board that your MCU is fitted to, using the Workspace browser (and again, in some cases an appropriate value may also be available from the dropdown next to the Browse button). Although selection of a board library is optional, it is recommended that you do this in most cases.

4.2.2 CMSIS-CORE Selection

For backwards compatibility reasons, the non-LPCOpen wizards for many parts provide the ability to link a new project with a CMSIS-CORE library project. The CMSIS-CORE portion

of ARM's **Cortex Microcontroller Software Interface Standard** (or **CMSIS**) provides a defined way of accessing MCU peripheral registers, as well as code for initializing an MCU and accessing various aspects of functionality of the Cortex CPU itself. The LPCXpresso IDE typically provides support for CMSIS through the provision of CMSIS library projects. CMSIS-CORE library projects can be found in the Examples directory of your LPCXpresso IDE installation.

Generally, if you wish to use CMSIS-CORE library projects, you should use `CMSIS_CORE_<partfamily>` (these projects use components from ARM's CMSIS v3.20 specification). The LPCXpresso IDE does in some cases provide libraries based on early versions of the CMSIS specification with names such as `CMSISv1p30_<partfamily>`, but these are not recommended for use in new projects.

The CMSIS library option within the LPCXpresso IDE allows you to select which (if any) CMSIS-CORE library you want to link to from the project you are creating. **Note** that you will need to import the appropriate CMSIS-CORE library project into the workspace before the wizard will allow you to continue.

For more information on CMSIS and its support in the LPCXpresso IDE, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/cmsis-support

Note - The use of LPCOpen instead of CMSIS-CORE library projects is recommended in most cases for new projects. (In fact LPCOpen actually builds on top of many aspects of CMSIS-CORE.) For more details see Software drivers and examples [9]

4.2.3 CMSIS DSP Library Selection

ARM's **Cortex Microcontroller Software Interface Standard** (or **CMSIS**) specification also provides a definition and implementation of a DSP library. The LPCXpresso IDE provides prebuilt library projects for the CMSIS DSP library for Cortex-M0/M0+, Cortex-M3 and Cortex-M4 parts, although a source version of it is also provided within the LPCXpresso IDE Examples.

Note - The CMSIS DSP library can be used with both LPCOpen and non-LPCOpen projects.

4.2.4 Peripheral Driver Selection

For some parts, one or more peripheral driver library projects may be available for the target MCU from within the Examples area of your LPCXpresso IDE installation. The non-LPCOpen wizards allow you to create appropriate links to such library projects when creating a new project. You will need to ensure that you have imported such libraries from the Examples before selecting them in the wizard.

Note - The use of LPCOpen rather than these peripheral driver projects is recommended in most cases for new projects.

4.2.5 Code Read Protect

NXP's Cortex and ARM7 based MCUs provide a "Code Read Protect" (CRP) mechanism to prevent certain types of access to internal flash memory by external tools when a specific memory location in the internal flash contains a specific value. The LPCXpresso IDE provides support for setting this memory location. For more details see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/code-read-protect-crp

4.2.6 Enable use of Floating Point Hardware

Certain MCUs may include a hardware floating point unit (for example NXP LPC32xx, LPC407x_8x, and LPC43xx parts). This option will set appropriate build options so that code is built to use the hardware floating point unit and will also cause startup code to enable the unit to be included.

4.2.7 Enable use of `Romdivide` Library

Certain NXP Cortex-M0 based MCUs, such as LPC11Axx, LPC11Exx, LPC11Uxx, and LPC12xx, include optimized code in ROM to carry out divide operations. This option enables the use of these Romdivide library functions. For more details see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/rom-divide

4.2.8 Disable Watchdog

Unlike most MCUs, NXP's LPC12xx MCUs enable the watchdog timer by default at reset. This option disables that default behavior. For more details, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/lpc12-watchdog

4.2.9 LPC1102 ISP Pin

The provision of a pin to trigger entry to NXP's ISP bootloader at reset is not hardwired on the LPC1102, unlike other NXP MCUs. This option allows the generation of default code for providing an ISP pin. For more information, please see NXP's application note, AN11015, "Adding ISP to LPC1102 systems".

4.2.10 Redlib Printf Options

The "Semihosting C Project" wizard for some parts provides two options for configuring the implementation of printf family functions that will get pulled in from the Redlib C library:

• Use non-floating-point version of printf

  • If your application does not pass floating point numbers to `printf()` family functions, you can select a non-floating-point variant of printf. This will help to reduce the code size of your application.

  • For MCUs where the wizard does not provide this option, you can cause the same effect by adding the symbol `CR_INTEGER_PRINTF` to the project properties.

• Use character- rather than string-based printf

  • By default `printf()` and `puts()` make use of `malloc()` to provide a temporary buffer on the heap in order to generate the string to be displayed. Enable this option to switch to using "character-by-character" versions of these functions (which do not require additional heap space). This can be useful, for example, if you are retargeting printf() to write out over a UART – since in this case it is pointless creating a temporary buffer to store the whole string, only to print it out over the UART one character at a time.

  • For MCUs where the wizard does not provide this option, you can cause the same effect by adding the symbol `CR_PRINTF_CHAR` to the project properties.

Note that if you only require the display of fixed strings, then using `puts()` rather than `printf()` will noticeably reduce the code size of your application.

4.2.11 Project Created

Having selected the appropriate options, you can then click on the Finish button, and the wizard will create your project for you, together with appropriate startup code and a simple `main.c` file. Build options for the project will be configured appropriately for the MCU that you selected in the project wizard.

You should then be able to build and debug your project, as described in Section 3.3 and Section 3.4.

5. Memory Editor and User-Loadable Flash Driver Mechanism

5.1 Introduction

By default, the LPCXpresso IDE provides a standard memory layout for known MCUs. This works well for parts with internal flash and no external memory capability, as it allows linker scripts to be automatically generated for use when building projects, and gives built-in support for programming flash as well as other debug capabilities.

In addition, the LPCXpresso IDE supports the editing of the target memory layout used for a project. This allows for the details of external flash to be defined or for the layout of internal RAM to be reconfigured. In addition, it allows a flash driver to be allocated for use with parts with no internal flash, but where an external flash part is connected.

5.2 New in LPCXpresso IDE v8.x — Support for Multiple Flash Regions

It is now possible to specify a flash driver for each region of defined flash memory. Now if a project is created that makes use of more than one flash region, including regions of different flash types, then LPCXpresso can automatically program the regions specified in a single operation.

For MCUs such as the LPC1800 and LPC4300 series, a typical use case could be to create an application to run from the MCU's internal flash that makes use of collateral in external SPI flash.

5.3 Memory Editor

The Memory Editor is accessed via the MCU settings dialog, which can be found at

Project Properties -> C/C++ Build -> MCU settings

This lists the memory details for the selected MCU, and will, by default, display the memory regions that have been defined by the LPCXpresso IDE itself.

LPCXpresso IDE User Guide -

User Guide   Rev. 8.1 — 15 February, 2016   23
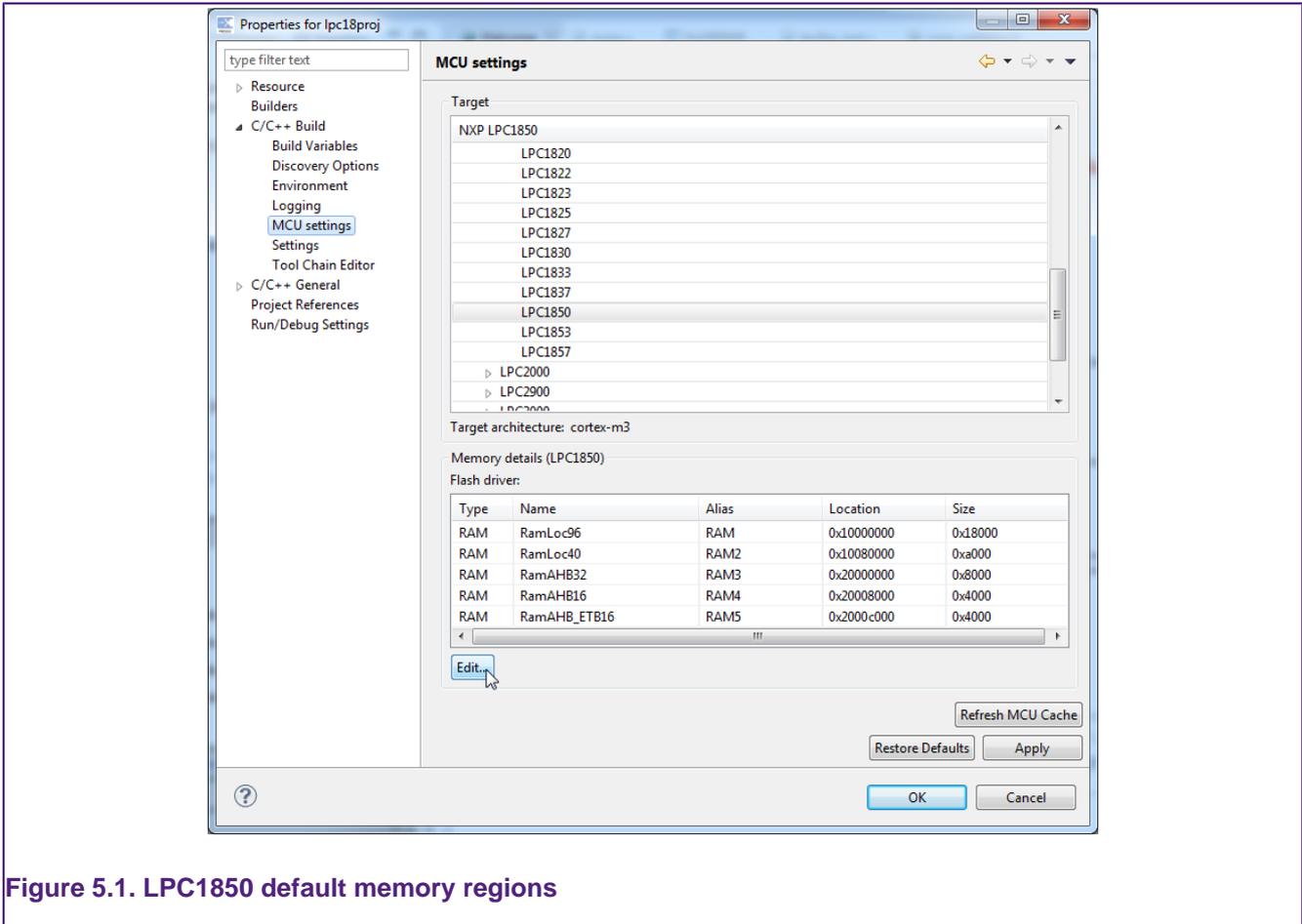
Figure 5.1. LPC1850 default memory regions

5.3.1 Editing a Memory Configuration

In the example below, we will show how the default memory configuration for an LPC4337 can be changed. Selecting the **Edit...** button will launch the **Memory configuration editor** dialog — see Figure 5.2.
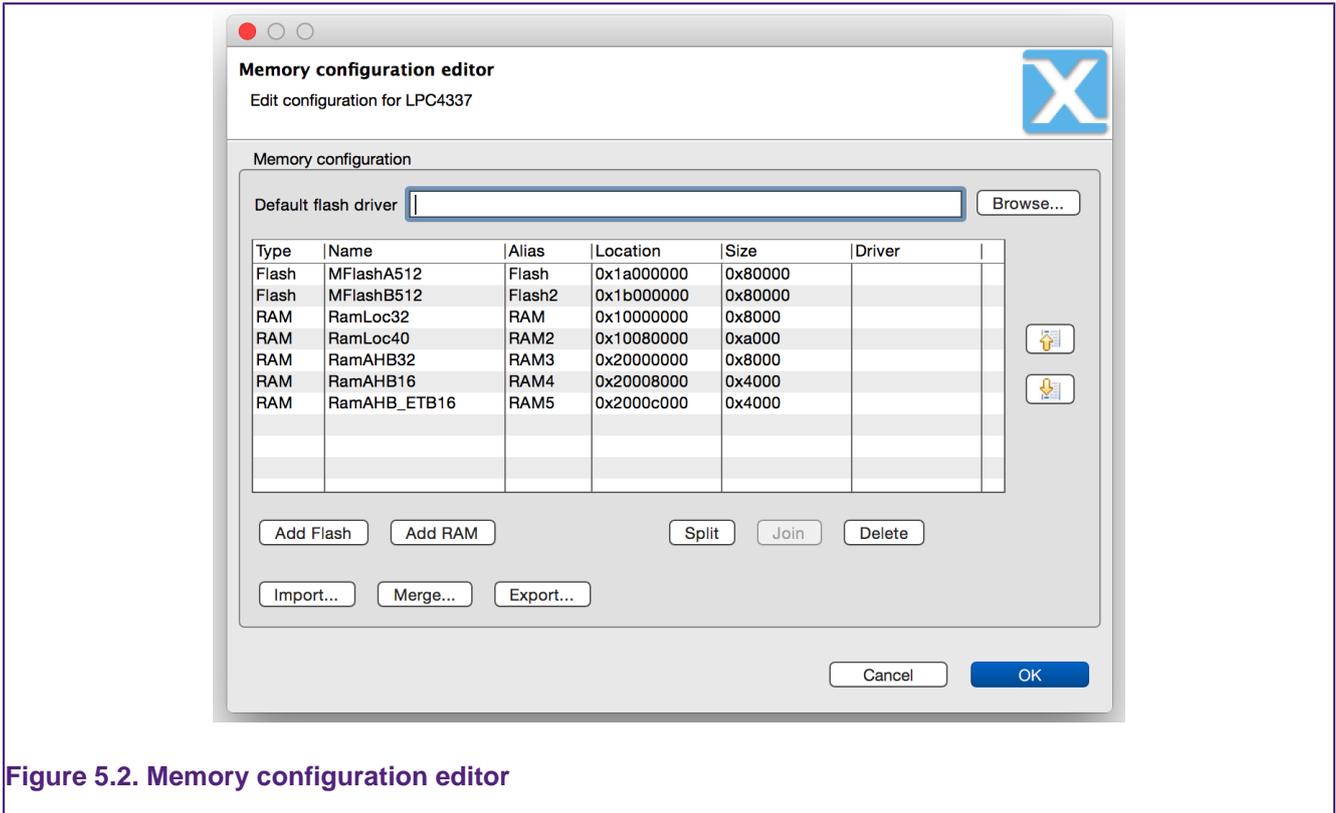
Figure 5.2. Memory configuration editor

> Known blocks of memory, with their type, base location, and size are displayed. Entries can be created, deleted, etc by using the provided buttons — see Figure 5.2.

Table 5.1. Memory editor controls

Button	Details
Add Flash	Add a new memory block of the appropriate type.
Add RAM	Add a new memory block of the appropriate type.
Split	Split the selected memory block into two equal halves.
Join	Join the selected memory block with the following block (if the two are contiguous).
Delete	Delete the selected memory block.
Import	Import a memory configuration that has been exported from another project, overwriting the existing configuration.
Merge	Import a partial memory configuration from a file, merging it with the existing memory configuration. This allows you, for example, to add an external flash bank definition to an existing project.
Export	Export a memory configuration for use in another project.
Up / Down	Reorder memory blocks. This is important: if there is no flash block, then code will be placed in the first RAM block, and data will be placed in the block following the one used for the code (regardless of whether the code block was RAM or Flash).
Browse(Flash driver)	Select the appropriate driver for programming the flash memory specified in the memory configuration. This is only required when the flash memory is external to the MCU. Flash drivers for external flash must have a ".cfx" file extension and must be located in the \bin\flash subdirectory of the LPCXpresso IDE installation. For more details see User loadable flash drivers .

> The name, location, and size of this new region can be edited in place. **Note** that when entering the size of the region, you can enter full values in decimal or in hex (by prefixing with `0x`), or by specifying the size in kilobytes or megabytes. For example:

> • To enter a region size of 32KB, enter `32768`, `0x8000` or `32k`.

- To enter a region size of 1MB, enter `0x100000` or `1m`.

Note that memory regions must be located on four-byte boundaries, and be a multiple of four bytes in size.

The screenshot in Figure 5.3 shows the dialog after the "Add Flash" button has been clicked.
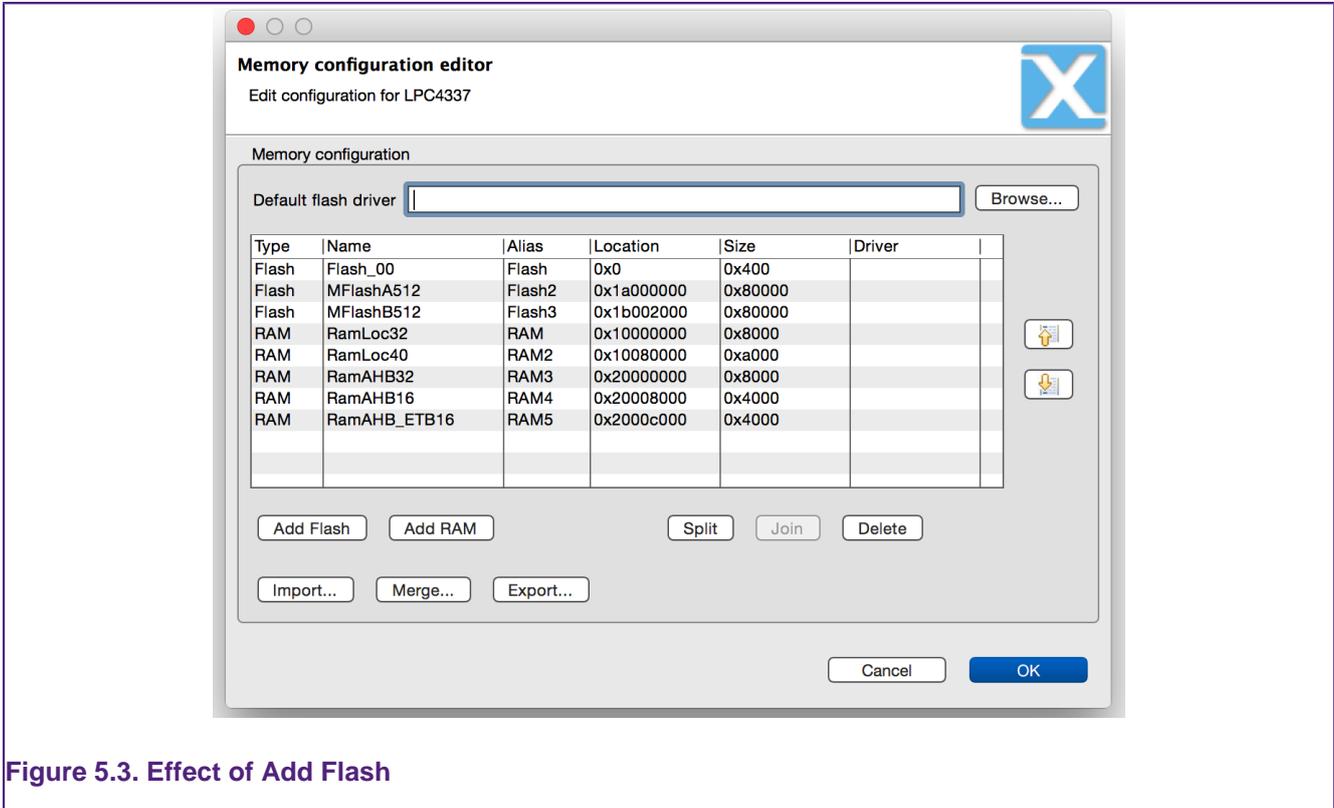
Figure 5.3. Effect of Add Flash

After updating the memory configuration, click **OK** to return to the MCU settings dialog, which will be updated to reflect the new configuration — see Figure 5.4.
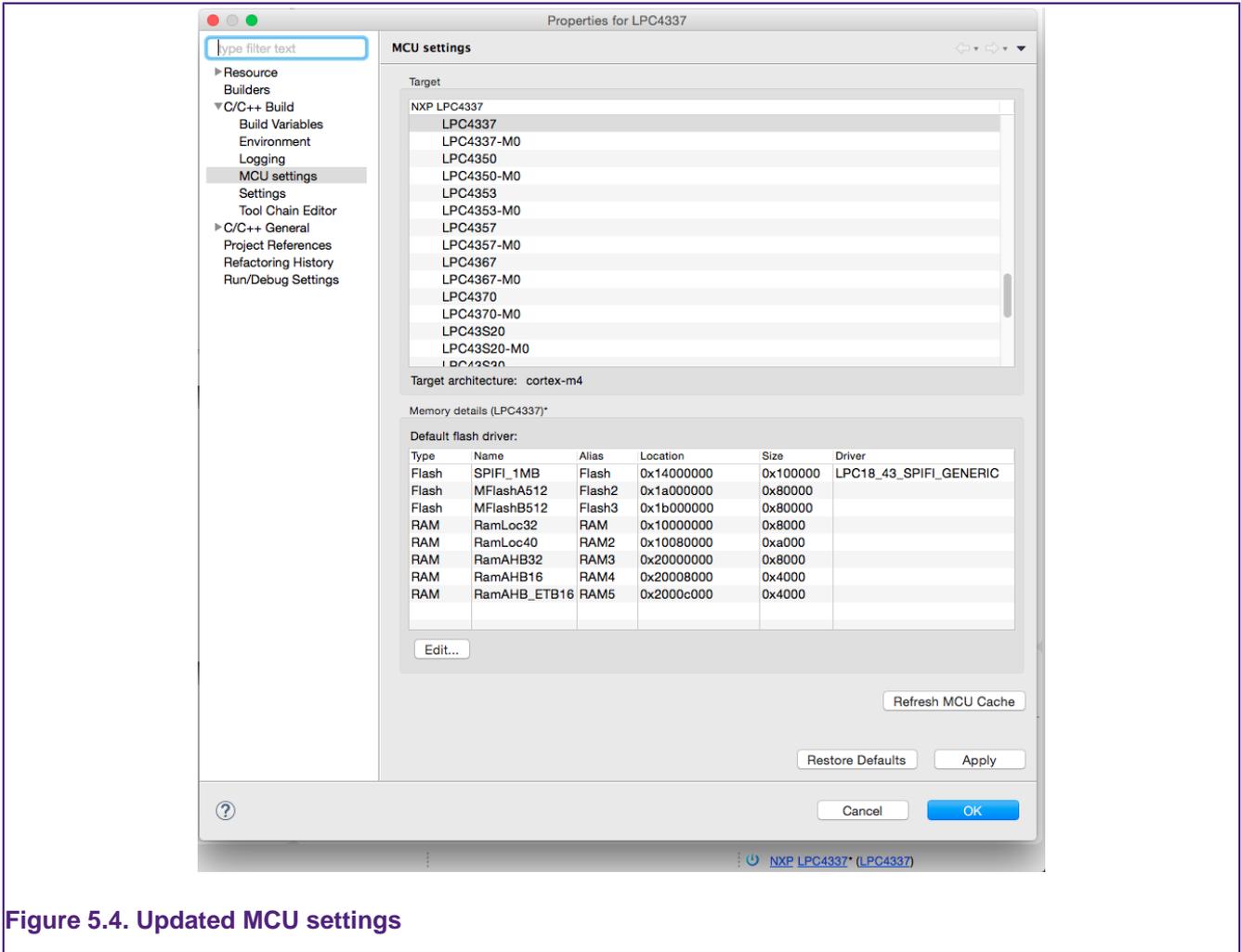
Figure 5.4. Updated MCU settings

Here you can see that the region has been named SPIFI_1MB, and the default flash driver has been deleted and the Generic SPIFI driver selected for the newly created SPIFI_1MB region.

Note that once the memory details have been modified, the selected MCU as displayed on the LPCXpressoIDE "Status Bar" (at the bottom of the IDE window) will be displayed with an asterisk (*) next to it. This provides an indication that the MCU memory configuration settings for the selected project have been modified.

5.3.2 Restoring a Memory Configuration

To restore the memory configuration of a project back to the default settings, simply reselect the MCU type, or use the "Restore Defaults" button, on the MCU Settings properties page.

5.3.3 Copying Memory Configurations

Memory configurations can be exported for import into another project. Use the Export and Import buttons for this purpose.

5.4 User-Loadable Flash Drivers

Flash drivers for external flash must have a `.cfx` file extension and must be located in the `/bin/Flash` subdirectory of the LPCXpresso IDE installation.

Many flash drivers supplied with the LPCXpresso IDE are for SPIFI flash devices. SPIFI memory is located in the MCU's memory map at `0x14000000`.

Earlier versions of the LPCXpresso IDE provided a number of dedicated SPIFI flash drivers (targeted at one or more specific SPIFI devices). This meant that it was the user's responsibility to select the correct SPIFI flash driver to match the device fitted to their target hardware.

LPCXpresso IDE v7.9.0 introduced an improved flash driver mechanism for SPIFI flash. Now a project targeting SPIFI flash only has to specify a single Generic flash driver: "LPC18_43_SPIFI_GENERIC.cfx". When a flash programming operation is performed, this driver will first interrogate the SPIFI device and determine its type, size, and configuration, which are then reported back to the host debug driver. Using this information the correct optimized programming routines for the SPIFI device detected can be used.

For a more complete list of supplied SPIFI flash drivers for the NXP LPC1800 and LPC4300 series of MCUs, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/lpc18-lpc43-spifi-flash-drivers

A small number of example flash drivers are also provided for parallel flash (connected to the MCU via the EMC). Such devices are located in the MCU's memory map at `0x1c000000`.

For example, the `LPC18_43_Diolan_S29AL016J70T.cfx` driver is for use in programming the external 2MB parallel flash fitted to the Diolan LP1850-DB1 and LPC4350-DB1 boards.

Source code for external flash drivers is also provided in the `/Examples/FlashDrivers` subdirectory of the LPCXpresso IDE installation. These projects can be used as the basis for writing your own flash drivers for other devices.

Note that the `/bin/Flash` subdirectory may also contain some drivers for the built-in flash on some MCUs. It should be clear from the filenames which these are. Do not try to use these drivers for external flash on other MCUs!

5.5 Projects and Multiple Flash Regions

As we have seen earlier, the LPCXpresso IDE will automatically create memory maps to match the target MCU and select an appropriate flash driver for MCUs containing internal flash.

From LPCXpresso IDE 8.0, there is extended support for the creation and programming of projects that span multiple flash devices. Previously only a single (default) flash driver could be specified for a project; now it is possible to specify a flash driver for each region of flash memory.

In Figure 5.5 we can see that a memory description can be created that not only describes the addresses and sizes of the memories but also specifies the flash drivers needed to program each flash region.
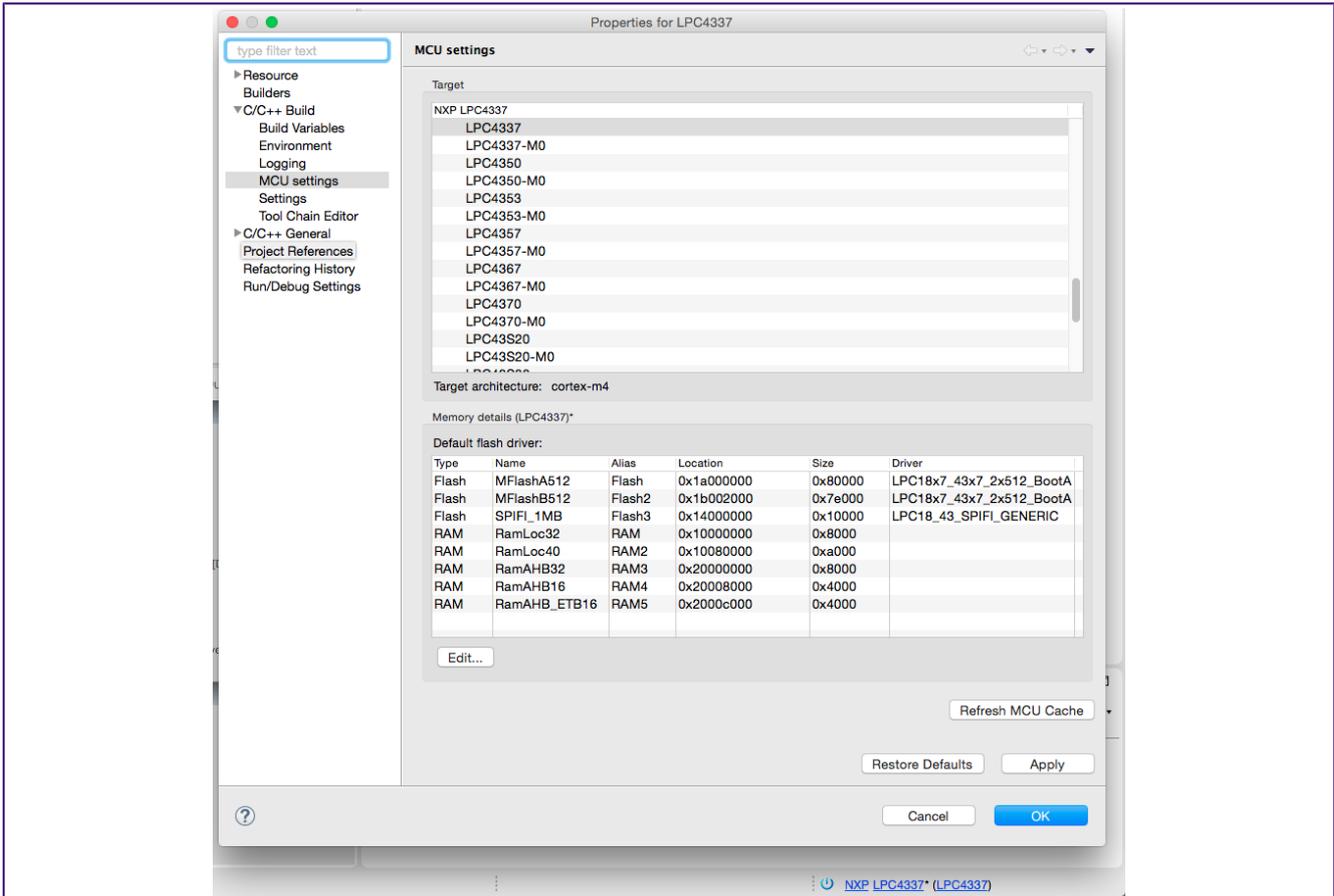
Figure 5.5. Multiflash MCU settings

These per-region flash drivers override any default flash driver that may have been specified.

Note that some additional care may be required when creating projects of this type. In this example, we have chosen to boot from the flash memory at 0x1A000000 (BankA) and have offset the second flash bank from its base address of 0x1B000000 by 8KB (where 8KB is the smallest block that can be programmed in the first 64KB of the flash device). This is to avoid a small risk that code or data located at the start of this flash may be misinterpreted by the MCU's bootloader.

For further information on creating projects to make use of multiple banks of flash memory, please see the FAQ at

https://www.lpcware.com/content/faq/lpcxpresso/coderodata-different-flash-blocks

5.6 Modifying Memory Configurations within the New Project Wizards

The New Project Wizards for LPC18xx, LPC43xx, and LPC541xx parts allow the project's memory configuration to be edited from within the wizard itself. This allows you to add, delete, or import memory blocks as required, as detailed in Editing a memory configuration, from within the project wizard itself.
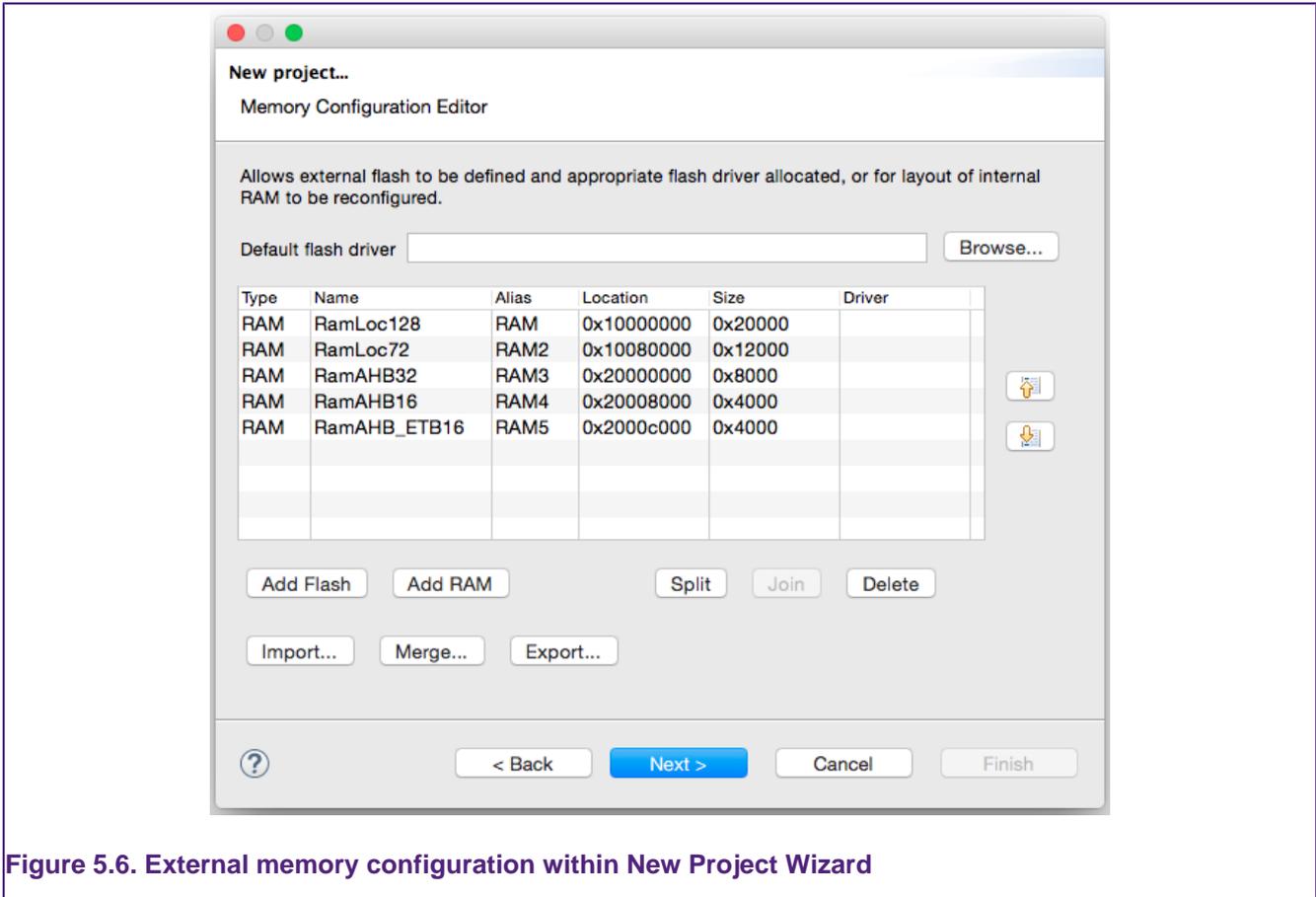
Figure 5.6. External memory configuration within New Project Wizard

In addition, a number of memory configurations containing LPC18/43 external flash and flash driver driver definitions suitable for use with the Merge option of the memory configuration editor can be found at:

```
<install_dir>\lpcxpresso\Wizards\MemConfigs
```

Also, for backwards compatibility reasons, a number of full memory configurations, suitable for use with the Import option, can be found in the directory:

```
<install_dir>\lpcxpresso\Wizards\MemConfigs\NXP
```

However, we would generally recommend the use of the "merge" files instead.

6. Multicore Projects

6.1 LPC43xx Multicore Projects

The LPC43xx family of MCUs contain a Cortex-M4 "master" core and one or more Cortex-M0 "slave" cores. After a power-on or Reset, the master core boots and is then responsible for booting the slave core(s). However, this relationship only applies to the booting process; after boot, your application may treat any of the cores as the master or a slave.

The LPCXpresso IDE allows for the easy creation of "linked" projects that support the targeting of LPC43xx Multicore MCUs. For more information on creating and using such multicore projects, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/lpc43xx-multicore-apps

6.2 LPC541xx Multicore Projects

Some members of the LPC541xx family of MCUs contain a Cortex-M4 core and a Cortex-M0+ core (with the Cortex-M4 being the master, and the M0+ the slave). After a power-on or Reset, the master core boots and is then responsible for booting the slave core. However, this relationship only applies to the booting process; after boot, your application may treat either of the cores as the master or the slave.

The LPCXpresso IDE allows for the easy creation of "linked" projects that support the targeting of LPC541xx Multicore MCUs. For more information on creating and using such multicore projects, please see the FAQ at

http://www.lpcware.com/content/faq/lpcxpresso/lpc541xx-multicore-apps