MCUXpresso IDE User Guide

Rev. 11.8.0 — 24 July, 2023 User guide

24 July, 2023

Copyright © 2023 NXP Semiconductors

All rights reserved.

1. Introduction to MCUXpresso IDE 1
1.1. MCUXpresso IDE Overview of Features 1
1.1.1. Summary of Features 2
1.1.2. Supported Debug Probes 3
1.1.3. Development Boards 4
2. New Features in MCUXpresso IDE version 11.8.0 8
2.1. Feature Highlights from previous releases of MCUXpresso IDE 8
3. IDE Overview 14
3.1. Workspaces 14
3.2. Welcome View 14
3.3. Documentation and Help 15
3.4. Perspectives and Views 16
3.5. Major Components of the Develop Perspective 18
3.5.1. Dark Theme 20
3.6. The Quickstart Panel 21
3.7. Project Explorer and New Projects 23
3.8. Updating MCUXpresso IDE 25
3.8.1. Locating IDE Components 26
4. Part Support Overview (Preinstalled and via SDKs) 27
4.1. Preinstalled Part Support 27
4.1.1. Differences in Preinstalled and SDK Part Handling 27
4.1.2. Viewing Preinstalled Part Support 28
4.2. SDK Part Support 29
4.2.1. Obtaining and Installing a Plugin SDK 29
4.2.2. SDK Part Support via SDK Builder 31
4.2.3. Obtaining and Installing an SDK via SDK Builder 31
4.2.4. Installing SDKs by Importing a Remote SDK Git Repository 33
4.2.5. Installing SDKs by Importing a Local Clone of an SDK Git Repository 36
4.2.6. Installed SDKs Operations 36
4.2.7. Installed SDKs Features 40
4.2.8. Advanced Use: SDK Importing and Configuration 40
4.2.9. Advanced Use: SDK Misc Options 42
4.2.10. Important notes for SDK Users 43
4.3. Enhanced Project Sharing Features 45
4.3.1. Project Drag and Drop 45
4.3.2. Project Local SDK Part Support 46
4.3.3. Project Local Support files 48
4.3.4. Export Project to Local SDK Git Repository 50
5. Creating New Projects using installed SDK Part Support 53
5.1. New Project Wizard 53
5.1.1. SDK New Project Wizard: Basic Project Creation and Settings 55
5.1.2. SDK New Project Wizard: Advanced Project Settings 58
5.2. Project Build 60
5.2.1. Build Configurations 61
6. Importing Example Projects (from installed SDKs) 62
6.1. SDK Example Import Wizard 63
6.1.1. SDK Example Import Wizard: Basic Selection 65
6.1.2. SDK Example Import Wizard: Advanced options 67
6.1.3. SDK Example Import Wizard: Import from XML fragment 68
6.1.4. Importing Examples to non default locations 70
7. SDK Project Component Management 71
7.1. SDK Project Component Management example 71
7.2. SDK Project Refresh 74
8. Open-CMSIS Component Management 75
8.1. Install a Pack 75
8.2. Add an Open-CMSIS-Pack component to a project 75
8.3. Manage components inside the project 76
9. Creating New Projects using Preinstalled Part Support 77

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

9.1. New Project Wizard 77
9.2. Creating a Project 78
9.2.1. Selecting the Wizard Type 79
9.2.2. Configuring the Project 80
9.2.3. Wizard Options 80
9.2.4. Project Created 83
10. Importing Example Projects (from the file system) 84
10.1. Code Bundles for LPC800 Family Devices 84
10.2. LPCOpen Software Drivers and Examples 85
10.3. Importing an Example Project 85
10.3.1. Importing Examples for the LPCXpresso4337 Development Board 87
10.4. Exporting Projects 88
10.5. Building Projects 89
10.5.1. Build Configurations 89
11. Importing Existing Executables 90
12. Debug Solutions Overview 94
12.1. Starting a Debug Session 94
12.2. An Introduction to Launch Configuration Files 95
12.3. LinkServer Debug Connections 98
12.4. LinkServer Debug Operation 98
12.4.1. LinkServer Debug Scripts 100
12.5. LinkServer Troubleshooting 101
12.5.1. Debug Log 101
12.5.2. Flash Programming 103
12.5.3. LinkServer executables 104
12.6. PEmicro Debug Connections 104
12.7. PEmicro Debug Operation 105
12.7.1. PEmicro Differences from LinkServer Debug 105
12.7.2. PEmicro Software Updates 106
12.8. SEGGER Debug Connections 106
12.8.1. SEGGER software installation 106
12.9. SEGGER Debug Operation 108
12.9.1. SEGGER Differences from LinkServer Debug 108
12.10. SEGGER Troubleshooting 108
13. Debugging a Project 112
13.1. Debugging Overview 112
13.1.1. Debug Launch 112
13.1.2. Debug Probe Selection Dialog (Probes Discovered) 113
13.1.3. Controlling Execution 116
13.2. Launch Configurations 118
13.2.1. Editing a Launch Configuration (LinkServer) 120
13.3. Common Debug Operations and Launch Configurations 123
13.3.1. Debug Quickstart Shortcuts 123
13.3.2. Connecting to a running Target (attach) 124
13.3.3. Controlling the initial Breakpoint (on main) 126
13.3.4. Debugging Pre-loaded binaries (Add Symbols) 129
13.3.5. Disconnect Behaviour 129
13.3.6. Project Flash Programming 130
13.4. Breakpoints 131
13.4.1. Breakpoint Types 131
13.4.2. Breakpoints Resources 132
13.4.3. Skip All Breakpoints 132
13.5. Watchpoints 133
13.5.1. Using Watchpoints to monitor stack depth 134
13.6. Registers 135
13.6.1. Basic Register set (Core Registers) 135
13.7. Faults 137
13.8. Peripherals 139

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

13.8.1. Custom SVD File 142
13.9. Offline Peripherals 143
13.9.1. Loading Custom SVD File in Offline Peripherals view 144
13.10. Global and Live Global Variables 144
13.11. Live Global Variable Plotting 147
13.11.1. Live Global Variable Graphing details 148
13.12. Heap and Stack View 150
13.13. Additional Debug Features 151
13.13.1. Local Variables 151
13.13.2. Disassembly view 152
13.13.3. Memory view 153
14. Configuring a Project 154
14.1. Changes available via Quickstart Quick Settings 154
14.2. Project Settings 155
14.3. Changing the MCU (and associated SDK) 155
14.3.1. Confirm device information 157
14.3.2. Removal of SDK components associated with the old MCU 160
14.3.3. Addition of SDK components associated with the new MCU 161
14.4. Changing the MCU (SDK) package type 162
15. MCUXpresso Config Tools 164
15.1. Using the Config Tools 164
15.1.1. Tool Perspectives 165
15.1.2. Pins Tool 165
15.1.3. Clocks Tool 165
15.1.4. Peripherals Tool 165
15.1.5. Device Configuration Tool 165
15.1.6. TEE Tool 166
15.1.7. Generate Code 166
15.1.8. SDK Components 166
16. The GUI Flash Tool 167
16.1. The Advanced GUI Flash Tool 168
16.1.1. Advanced GUI Flash Tool command Preview 170
16.1.2. Advanced GUI Flash Tool logged Output 172
16.1.3. Advanced GUI Flash Tool Programming an arbitrary Binary 172
17. LinkServer Flash Support 173
17.1. Default vs Per-Region Flash Drivers 173
17.2. Advanced Flash Drivers 174
17.2.1. LPC18xx / LPC43xx Internal Flash Drivers 174
17.2.2. LPC SPIFI QSPI Flash Drivers 175
17.2.3. i.MX RT QSPI and Hyper Flash Drivers 176
17.2.4. Flash Drivers using SFDP (LPC and iMX RT) 177
17.3. Kinetis Flash Drivers 180
17.4. Configuring projects to span multiple Flash Devices 181
17.5. The LinkServer GUI Flash Programmer 181
17.6. The LinkServer Command Line Flash Programmer 181
17.6.1. Command Line Programming 181
18. C/C++ Library Support 188
18.1. Overview of Redlib, Newlib and NewlibNano 188
18.1.1. Redlib extensions to C90 188
18.1.2. Newlib vs NewlibNano 188
18.2. Library Variants 189
18.3. Switching the selected C library 190
18.3.1. Manually Switching 190
18.4. What is Semihosting? 191
18.4.1. Background to Semihosting 191
18.4.2. Semihosting Implementation 191

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

18.4.3. Semihosting Performance 191
18.4.4. Important notes about using Semihosting 191
18.4.5. Semihosted printf and Debugging 192
18.4.6. Semihosting Specification 193
18.5. Use of printf 193
18.5.1. Redlib printf Variants 193
18.5.2. NewlibNano printf Variants 193
18.5.3. Newlib printf variants 194
18.5.4. Printf when using LPCOpen 194
18.5.5. Printf when using SDK 194
18.5.6. Retargeting printf/scanf 194
18.5.7. How to use ITM Printf 195
18.6. itoa() and uitoa() 196
18.6.1. Redlib 196
18.6.2. Newlib/NewlibNano 197
18.7. Libraries and linker scripts 197
19. Memory Configuration and Linker Scripts 199
19.1. Introduction 199
19.2. Managed Linker Script Overview 199
19.3. How are Managed Linker Scripts Generated? 200
19.4. Default Image Layout 201
19.5. Examining the layout of the generated image 201
19.5.1. Linker --print-memory-usage 202
19.5.2. arm-none-eabi-size 203
19.5.3. Linker Map Files 203
19.6. Image Information (Info) 203
19.6.1. Memory Usage 205
19.6.2. Memory Contents 205
19.6.3. Call Graph 206
19.6.4. Use of Filters 208
19.7. Enhanced Syntax Highlighting 209
19.8. Other Options affecting the Generated Image 215
19.8.1. LPC MCUs – Code Read Protection 215
19.8.2. Kinetis MCUs – Flash Config Blocks 216
19.8.3. Placement of USB Data 217
19.8.4. Plain Load Image 217
19.8.5. Link Application to RAM 218
19.9. Modifying the Generated Linker Script / Memory Layout 219
19.10. Using the Memory Configuration Editor 219
19.10.1. Editing a Memory Configuration 220
19.10.2. Device specific vs Default Flash Drivers 223
19.10.3. Restoring a Memory Configuration 223
19.10.4. Copying Memory Configurations 223
19.11. Global Data Placement 223
19.12. Modifying heap/stack placement 224
19.12.1. MCUXpresso style Heap and Stack 224
19.12.2. LPCXpresso style Heap and Stack 225
19.12.3. Reserving RAM for IAP Flash Programming 226
19.12.4. Stack Checking 226
19.12.5. Heap Checking 227
19.12.6. Checking the Heap from your Application 227
19.13. Placement of specific code/data Items 228
19.13.1. Placing code and data into different Memory Regions 228
19.13.2. Placing data into different RAM blocks using Macros 230
19.13.3. Noinit Memory Sections 230
19.13.4. Placing code/rodata into different FLASH Blocks 231
19.13.5. Placing specific functions into RAM Blocks 232

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

19.13.6. Reducing Code Size when support for LPC CRP or Kinetis Flash Config Block is Enabled ... 233
19.14. FreeMarker Linker Script Templates ... 233
19.14.1. Basics ... 234
19.14.2. Reference ... 234
19.15. FreeMarker Linker Script Template Examples ... 239
19.15.1. Relocating code from FLASH to RAM ... 239
19.15.2. Configuring projects to span multiple Flash Devices ... 242
19.16. Disabling Managed Linker Scripts ... 243
20. Multicore Projects ... 244
20.1. Introduction ... 244
20.2. Creating a Primary / Secondary project Pair (using an SDK) ... 245
20.2.1. Creating the M0 Secondary project ... 245
20.2.2. Creating the M4 Primary project ... 247
20.3. Creating a Primary / Secondary project Pair (using Preinstalled Part Support) ... 251
20.3.1. Creating the M0 Secondary project ... 251
20.3.2. Creating the M4 Primary project ... 253
20.4. Debugging MultiCore Projects ... 254
20.4.1. Controlling Debug Views ... 255
20.4.2. Secondary Project Debug ... 256
20.4.3. Auto-debug secondary project(s) for multicore projects ... 257
20.5. MultiCore Projects additional Information ... 258
20.5.1. Defines ... 258
20.5.2. Secondary Boot Code ... 258
20.5.3. Reset Handler code ... 259
21. Appendix – Additional Hints and Tips ... 260
21.1. Part Support Handling from SDKs ... 260
21.1.1. SDK Version control ... 260
21.1.2. SDK Manifest versioning ... 260
21.1.3. Device versions ... 261
21.2. How do I switch between Debug and Release builds? ... 262
21.2.1. Changing the build configuration of a single project ... 262
21.2.2. Changing the build configuration of multiple projects ... 262
21.3. Editing Hints and Tips ... 262
21.3.1. Multiple views onto the same file ... 262
21.3.2. Viewing two edited files at once ... 263
21.3.3. Source folding ... 263
21.3.4. Editor templates and Code completion ... 263
21.3.5. Brace matching ... 263
21.3.6. Syntax coloring ... 264
21.3.7. Comment/uncomment block ... 264
21.3.8. Format code ... 264
21.3.9. Correct Indentation ... 264
21.3.10. Insert spaces for tabs in editor ... 264
21.3.11. Replacing tabs with spaces ... 265
21.4. Hardware Floating Point Support ... 265
21.4.1. Floating Point Variants ... 265
21.4.2. Floating point use – Preinstalled MCUs ... 265
21.4.3. Floating point use – SDK installed MCUs ... 266
21.4.4. Modifying floating point configuration for an existing project ... 266
21.4.5. Do all Cortex-M4 MCUs provide floating point in hardware? ... 266
21.4.6. Why do I get a hard fault when my code executes a floating point operation? ... 267
21.5. LinkServer Scripts ... 267
21.5.1. Supplied Scripts ... 267
21.5.2. User Scripts ... 267
21.5.3. Debugging code from RAM ... 267
21.5.4. LinkServer Scripting Features ... 269

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.6. RAM projects with LinkServer 271
21.6.1. Advantages of developing with RAM projects 272
21.7. The Console View 273
21.7.1. Console types 273
21.7.2. Copying the contents of a console 274
21.7.3. Relocating and duplicating the Console view 275
21.8. Using Terminal View for UART communication with target 276
21.9. Using and troubleshooting LPC-Link2 279
21.9.1. LPC-Link2 hardware 279
21.9.2. Softloaded vs Pre-programmed probe firmware 279
21.9.3. LPC-Link2 firmware variants 279
21.9.4. Manually booting LPC-Link2 280
21.9.5. LPC-Link2 windows drivers 282
21.9.6. LPC-Link2 failing to enumerate 282
21.9.7. Troubleshooting LPC-Link2 284
21.10. Using and troubleshooting MCU-Link 285
21.10.1. MCU-Link hardware 285
21.10.2. MCU-Link CMSIS-DAP firmware 285
21.10.3. MCU-Link host drivers 287
21.10.4. MCU-Link JLink-compatible firmware 287
21.10.5. Troubleshooting MCU-Link 287
21.11. Creating bin, hex or S-Record files 288
21.11.1. Simple conversion within the IDE 288
21.11.2. From the command line 289
21.11.3. Automatically converting the file during a build 290
21.11.4. Binary files and checksums 290
21.12. Post-build (and Pre-build) steps 290
21.12.1. Temporarily removing post-build steps 291
21.13. Save Info for Support 291
22. Revision History 293
23. Legal information 294
23.1. Definitions 294
23.2. Disclaimers 294
23.3. Trademarks 295

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

1. Introduction to MCUXpresso IDE

MCUXpresso IDE version 11.8.0 is an easy-to-use Eclipse-based development environment for NXP MCUs based on Arm Cortex-M cores. It provides an end-to-end solution enabling engineers to develop embedded applications from initial evaluation to final production. The MCUXpresso IDE offers advanced editing, compiling, and debugging features with the addition of MCU-specific debugging views, code trace and profiling, multicore debugging, and integrated configuration tools.

The MCUXpresso platform ecosystem includes:

- MCUXpresso IDE [14] - a software development environment for creating applications for NXP's ARM Cortex-M based MCUs including "LPC", "Kinetis" and iMX RT" ranges.
- MCUXpresso Config Tools [164] , comprising of Pins, Clocks and Peripherals Tools that are designed to work with SDK projects and are fully integrated and installed by default.
- MCUXpresso SDKs [27], each offering a package of device support and example software extending the capability and part knowledge of MCUXpresso IDE.
- The range of LPCXpresso development boards, each of which includes a built-in "LPC-Link", "LPC-Link2", or CMSIS-DAP compatible debug probe. These boards are developed in collaboration with Embedded Artists.
- The range of Tower and Freedom development boards, most of which include an OpenSDA debug circuit supporting a range of firmware options.
- The range of iMX RT Series EVK development board which include an OpenSDA debug circuit supporting a range of firmware options, or high performance FreeLink (LPC-Link2 compatible) debug probe.
- The range of EVK development boards which include an MCU-Link debug circuit
- The standalone "LPC-Link2" debug probe.
- The standalone "MCU-Link" and "MCU-Link Pro" debug probes.

This guide is intended as an introduction to using MCUXpresso IDE. It assumes that you have some knowledge of MCUs and software development for embedded systems.

Note: MCUXpresso IDE incorporates technology and design from LPCXpresso IDE. This means that users familiar with LPCXpresso IDE will find MCUXpresso IDE looks relatively familiar.

1.1 MCUXpresso IDE Overview of Features

MCUXpresso 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.

MCUXpresso IDE is based on the Eclipse IDE and includes the industry standard ARM GNU toolchain. It brings developers an easy-to-use and unlimited code size development environment for NXP MCUs based on Cortex-M cores (LPC, Kinetis and iMX RT). The IDE combines the best of the widely popular LPCXpresso and Kinetis Design Studio IDE's, providing a common platform for all NXP Cortex-M microcontrollers.

MCUXpresso IDE is a free toolchain providing developers with no restrictions on code or debug sizes. It provides an intuitive and powerful interface with profiling, power measurement on supported boards, GNU tool integration and library, multicore capable debugger, trace functionality and more. MCUXpresso IDE debug connections support Freedom, Tower, EVK, LPCXpresso and custom development boards with industry leading open-source and commercial debug probes including MCU-Link, MCU-Link Pro, LPC-Link2, PEmicro and SEGGER.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

The fully featured debugger supports both SWD and JTAG debugging, and features direct download to on-chip and external flash memory.

For the latest details on new features and functionality, please visit:

https://www.nxp.com/mcuxpresso/ide

1.1.1 Summary of Features

Complete C/C++ integrated development environment

- Eclipse-based IDE with many ease-of-use enhancements
- The IDE installs with various Eclipse plugins including:
  - Git, support for PEmicro debug probes, ARM CMSIS-Pack
- The IDE can be further enhanced with many other Eclipse plugins
- Command line tools are included for integration into build, test, and manufacturing systems

Industry standard GNU toolchain including:

- C and C++ compilers, assembler, and linker
- Converters for SREC, HEX, and binary

Advanced project wizards

- Simple creation of pre-configured applications for specific MCUs [77]
  - Extendable with MCUXpresso SDKs [53]
- Device-specific support for NXP's ARM-based MCUs (including LPC, Kinetis and iMX RT)
- Automatic generation [199] of linker scripts for correct placement of code and data into Flash and RAM
  - Extended support for flexible placement of heap and stack [224]
- Automatic generation of MCU-specific startup and device initialization code
- **Note**: No assembler required with Cortex-M MCUs

Advanced multicore support

- Provision for creating linked projects [244] for each core in multicore MCUs
- Debugging of multicore projects [254] within a single IDE instance, with the ability to link various debug views to specific cores

Fully featured native debugger supporting SWD and JTAG connection via LinkServer

- Built-in optimized Flash programming [173] for internal and external QSPI and Hyper Flash
- High-level and instruction-level debug [116]
- Breakpoints [131] and Watchpoints [133]
- Views of CPU registers [135] and on-chip peripherals [139]
- Support for multiple devices on the JTAG scan-chain

Full install and integration of 3rd party debug solutions from:

- PEmicro [104]
- SEGGER J-Link [106]

Library support

- Redlib: a small-footprint embedded C library
  - RedLib-nf: a smaller footprint library offering reduced fprintf support

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

  - RedLib-mb: a library variant offering enhanced semihosting performance
- 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
- Extendible support per device via MCUXpresso SDKs

Trace functionality

- Instruction trace via Embedded Trace Buffer (ETB) on certain Cortex-M3/M4/M7/M33 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-M0+/M3/M4/M7/M33 based MCUs when debugging via MCU-Link, MCU-Link Pro and LPC-Link2, providing functionality including:
  - Profile tracing
  - Interrupt tracing
  - Datawatch tracing
  - Printf over ITM
    - **Note**: Now extended to work with PEmicro and SEGGER J-Link, in addition to native LinkServer

LinkServer Energy Measurement

- On LPCXpresso boards, sample power usage at adjustable rates of up to 100 ksps; average power and energy usage display option
- MCU-Link Pro or built-in implementations provide additional features, like simultaneous target supply and current measurement, dynamic range switching for increased accuracy, analog signal input, trigger-based measurements.
  - Power Profile view providing correlated energy and trace measurements.
- Explore detailed plots of collected data in the IDE
- Export and import data for offline analysis

RTOS Debug Awareness

- GDB thread awareness for various RTOS providers: FreeRTOS, Azure ThreadX, Zephyr and MQX
- Views for different RTOS elements: Thread list, Message queues, Semaphores, Mutexes, Event flags, Timers, Memory block pools, Memory byte pools, etc

MCUXpresso Configuration Tools

- MCUXpresso Config Tools [164], designed to work with SDK projects are fully integrated and installed by default

1.1.2 Supported Debug Probes

MCUXpresso IDE installs with built in support for 3 debug solutions. This support includes the installation of all necessary drivers and supporting software.

Note: Certain mbed boards require a serial port driver to be recognized and this one exception must be installed separately for each board. The driver is linked from *Help -> Additional Resources -> MBED Serial Port Driver Website*

In normal use MCUXpresso IDE presents a similar interface and array of features for each of the solutions listed below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Native LinkServer (including CMSIS-DAP) as also used in LPCXpresso IDE

- this supports a variety of debug probes including OpenSDA programmed with CMSIS-DAP firmware, LPC-Link2, MCU-Link etc.
- https://community.nxp.com/message/630896

PEmicro

- this supports a variety of debug probes including OpenSDA programmed with PEmicro compatible firmware and MultiLink and Cyclone probes
- https://www.pemicro.com/

SEGGER J-Link

- this supports a variety of debug probes including OpenSDA programmed with J-Link compatible firmware and J-Link debug probes
- https://www.segger.com/

Please see Debug Solutions Overview Chapter [94] for more details.

Note: Kinetis Freedom and Tower boards typically provide an on-board OpenSDA debug circuit. This can be programmed with a range of debug firmware including:

- mBed CMSIS-DAP – supported by LinkServer connections
- DAP-Link – supported by LinkServer connections (DAP-Link is preferred to mBed CMSIS-DAP when available)
- J-Link – supported by SEGGER J-Link connections
- PEmicro – supported by PEmicro connections

The default firmware can be changed if required, for details of the procedure and range of supported firmware options please information visit: https://www.nxp.com/opensda



Tip

Under Windows 10, OpenSDA Bootloaders might experience problems and the OpenSDA LED will blink an error code. The following article discusses the problem and how it can be fixed: https://mcuoneclipse.com/2018/04/10/recovering-opensda-boards-with-windows-10

1.1.3 Development Boards

NXP have a large range of development boards that work seamlessly with MCUXpresso IDE including:

LPCXpresso Boards for LPC

These boards provide practical and easy-to-use development hardware to use as a starting point for your LPC Cortex-M MCU based projects.

© 2023 NXP Semiconductors. All rights reserved.

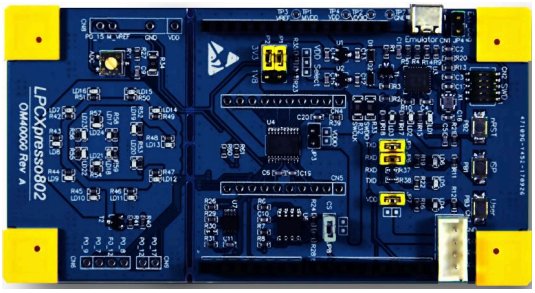


Figure 1.1. LPC800 series (LPCXpresso802)



Figure 1.2. LPCXpresso Development Board (LPCXpresso54608)

For more information, visit: https://www.nxp.com/lpcxpresso-boards

Freedom and Tower Boards for Kinetis

Similarly, for Kinetis MCUs there are many development boards available including the popular Freedom and Tower ranges of boards.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

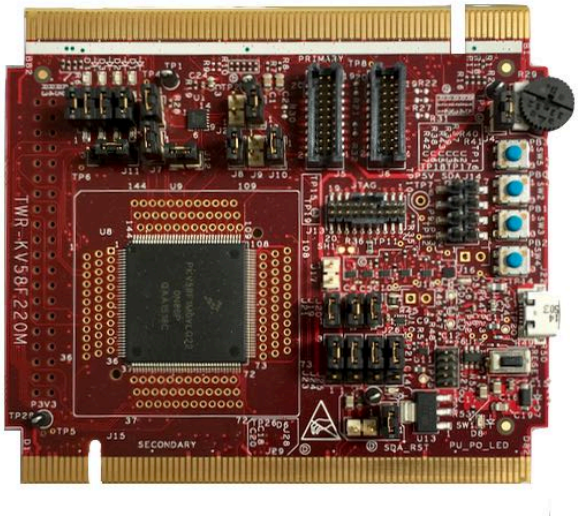


Figure 1.3. Tower (TWR-KV58F220M)

For more information, visit: https://www.nxp.com/pages/:TOWER_HOME

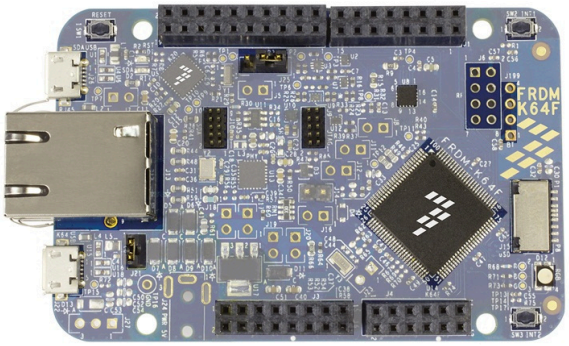


Figure 1.4. Freedom (FRDM-K64F)

For more information, visit: https://www.nxp.com/pages/:FREDEVPLA

iMX RT Crossover Processor Boards

iMX RT based boards bring the convergence of low power applications processors with high-performance microcontrollers.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Figure 1.5. i.MX RTxxxx Series (MIMXRT1050-EVK)

For more information, visit: https://www.nxp.com/pages/:IMX-RT-SERIES

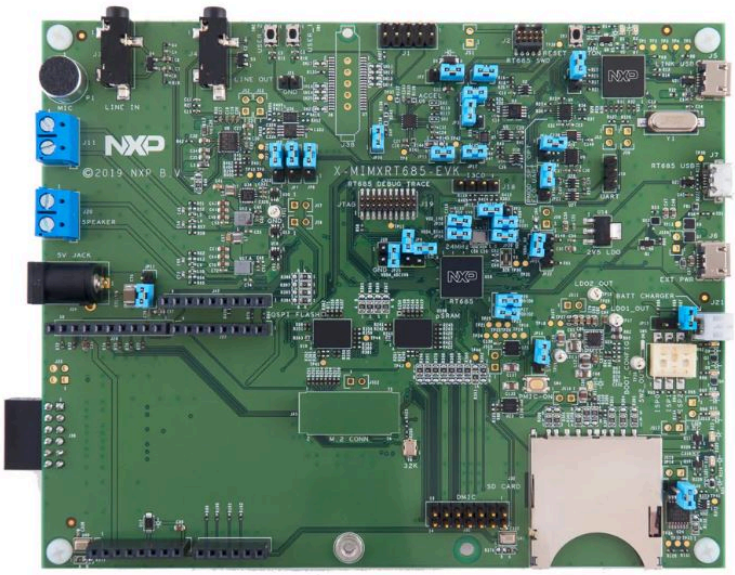


Figure 1.6. i.MX RTxxx Series (MIMXRT600-EVK)

For more information, visit: https://www.nxp.com/pages/:IMX-RT-SERIES

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

2. New Features in MCUXpresso IDE version 11.8.0

The new MCUXpresso IDE product comes with a set of improvements and bug fixes including:

Product

- New IDE build for Mac with native Apple silicon support. See download section for the new product.
- Upgraded: Eclipse version to 2022.12 (Eclipse Platform 4.26.0 / CDT 11.0.0).
- Upgraded: MCUXpresso IDE integrated with OpenJDK Runtime Environment Temurin-17.0.5+8.
- Upgraded: GNU ARM Embedded Toolchain to version 12.2.Rel1.
- Upgraded: Version v14 of MCUXpresso Config Tools.
- Upgraded: ARM CMSIS-Pack Eclipse Plug-ins 2.8.0.
- Upgraded: Newer SEGGER J-Link software (v7.88m).
- Upgraded: Newer PEmicro plugin (v5.6.7).
- Synchronization with SDK v2.14.x.

SDK

- Added support for selecting library type in SDK CLI. Now redlib, newlib, newlib_nano can be selected as options when generating a project. Check MCUXpresso_IDE_Command_Line_User_Guide.pdf for details.
- Extended integration with ARM CMSIS-Pack Eclipse Plug-ins. Now adding a new Open-CMSIS-Pack component is managed by ARM CMSIS-Pack Eclipse Plug-in. This brings support for:
  - components dependency
  - multiple component selection
  - automatically check dependencies in the new multiple component selection view
  - copy configuration and template files to project

Debug

- LinkServer LPC-Link2 firmware now softloaded as v5.460 which offers support for powering certain RT1xxx EVK boards (which incorporate on-board debug probes based on LPC-Link2 hardware) through the USB debug connection.

Please also see the supplied ReadMe document for further information and details of bug fixes and improvements. This document is located within the MCUXpresso IDE installation folder.

2.1 Feature Highlights from previous releases of MCUXpresso IDE

Product

- GNU make 4.4 is now integrated on IDE on all OS-es.
- The IDE does not internally use WMI command line utility anymore (deprecated as of Windows 10, version 21H1). PowerShell will be used instead.
- New Welcome View [14], designed to provide a dramatically improved out-of-box experience for new users
- Improved IDE Update [25] capability simplifying the update procedure for all supported hosts
- Scripts to create a command line environment now supplied in DOS and Bash versions
  - Use of these scripts is described within the Installation Guide
- SDK installation options improved, see SDK Importing and Configuration [40]
- Windows version now uses *Busybox* (from the GNU MCU Eclipse Windows Build Tools project) to provide Unix-like layer for GCC tools

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- All previous Pro Edition features have been incorporated into the standard Free edition and the Pro edition has been discontinued

IDE

- Import ELF binary/executable. This is available from File -> Import -> C/C++ -> MCUXpresso Executable Importer.
- Added Power Profile feature aiming to correlate energy/power measurement with SWO trace. The view displays the SWO based trace information (similar with SWO Profile view), plus each function showing various energy consumption information. The feature is available from Eclipse Menu -> Analysis -> Power Profile.
- Added analog data traffic statistics information for energy/power based views.
- Added FreeRTOS Task Notifications view to display task notification list for each task, including status and value properties. This included synchronization with FreeRTOS v10.4.3. For more information please refer to *MCUXpresso_IDE_FreeRTOS_Debug_Guide.pdf* documentation.
- Azure RTOS ThreadX debug awareness:
  - Ability to export trace that can be further used within TraceX Microsoft tool;
  - GDB thread awareness;
  - Views, similar to the FreeRTOS, for: Thread list, Message queues, Semaphores, Mutexes, Event flags, Timers, Memory block pools, Memory byte pools.
- Added offline peripheral view (“Offline Peripherals”). With this view, the peripheral registers can be inspected outside of a debug session. Reset value can be inspected as well together with the rest of the register elements.
- Peripherals+ view design was changed to support register group expansion directly into the Peripherals+ view, with no extra Memory View usage. Consequently, all elements shown before in Memory View are now available directly in the Peripherals+ view: values, bitfields, details.
- Added Energy Measurement view for energy consumption measurement.
  - Various measurement channels: voltage, current, power, etc, at various sampling speeds, up to 100ksps.
  - Import/Export measurement.
  - Zooming, panning and annotation capabilities on graph.
  - Unlimited sampling time, depending on the available space on disk.
  - Data gathering enabled / disabled by trigger signal MCU-Link Pro and on-board probes with energy measurement circuitry can use a GPIO signal as a trigger/enable (input) for the energy measurement data gathering such that data capture will commence and stop based on trigger signal transitions. The Energy Measurement view includes a trigger configuration section to make use of this probe capability. The supported modes of operation include level-based and pulse-based transitions of the trigger signal, with additional configuration of start and stop conditions. **Note**. This feature requires firmware version MCU-LINK CMSIS-DAP v2.249 (or greater).
- Community forum accessible now from the main toolbar too (together with the older link from **Help -> MCUXpresso IDE support forum**).
- Expressions added in Global Variables are now persistent between debug sessions.
- Added new control to manage the maximum number of child expressions that are evaluated in advance by the Live Variables service. This improves the Global Variables window responsiveness for instance when large structures are displayed. New control available on Eclipse Preferences -> MCUXpresso IDE -> Debug Options -> “Number of subexpressions proactively evaluated by Live Variables service”. Default is 2 set as depth.
- Added Save Info for Support [291] option to help reporting an issue by gathering MCU IDE environment information.
- New Plugin SDK [29] mechanism that provides a simpler flow for selection and installation of MCUXpresso IDE SDKs
- New Dark Theme [20] provides a low-light interface that displays mostly dark surfaces that may be more relaxing on the eye
- Improved Image Information View [203]

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Improved Installed SDK Operations [36]
- Improved Code Size [202]
  - the code size of debug builds of SDK projects has been reduced by decreasing the overhead of the assert() function, which is commonly called by SDK functions.
- Added support for handling more complex specification of dependencies between SDK components.
- Heap and Stack View [150] for all debug solutions
  - shows usage against allocated Managed Linkerscript [199] RAM allocation for bare metal projects
  - Live Heap updates and stack when paused
- Image Information View [203] extends and replaces the Symbol Browser
  - incorporating detailed memory usage plus hyperlinked Memory Content and Static Call Graph display
- Revamped Develop Perspective [18]
- Editor Syntax Highlighting [209] for linker scripts, linker templates and debug map files
  - providing linked navigation of file contents
- Redesigned Quickstart Panel [18]
  - Quick Start panel -> Quick Settings [154] now displays the current settings for Library
  - links for Dedicated Debug Operations [123] for all supported Debug Solutions
- Faults View [137] automatically displayed (for LinkServer) should a CPU fault occur
- Improved Registers View [135] with enhanced display and grouping options
- Launch Configurations [95] are now only automatically generated for the selected build configuration
- Project Memory Configuration [220] can now be edited *in place* for settings and wizards
- Project Explorer view enhanced to display current project build configuration for the selected project (also displayed in Quickstart view)
- Support for new MCUs based on the ARM Cortex M33

Projects

- A specific SVD file can be assigned to a project that can be used afterwards within OfflinePeripherals [144] or Peripherals+ [142] views.
- Imported or new projects [23] will now expand to show the source file containing the main function and also opens this file within the editor
- Improved display of Components in New Project Wizard [53]
- Quick Start panel -> Quick Settings [154] now displays the current settings
- Project association with an SDK (MCU) can now be flexibly managed, maintaining existing memory configuration if desired see Project Configuration [154]
- Many enhancements for improved Project Sharing [45] including:
  - Drag and Drop of projects for import and export
  - Options for project local inclusion of: SDK part support, flash drivers, and LinkServer connect and reset scripts
- Project Virtual Nodes [155] introduced to enable easy visibility and editing of project configurations
- Project GUI Flash Tool [130] for all debug solutions delivered via project launch configurations

Debug

- Added Zephyr RTOS Awareness:
  - Added GDB thread awareness for LinkServer debug connection.
  - Added Threads view.
- The target configuration for SWO trace is now optional. The default setting is to have the IDE perform the necessary configuration for SWO trace. However, the user can choose to disable this functionality and rely on target configuration performed by the application. The behavior

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

is controlled by the “SWO configured by IDE” (depending on the debug probe) checkbox in “SWO Trace Config” view -> “Change” button -> “Clock speed configuration” dialog. For more information please refer to *MCUXpresso_IDE_SWO_Trace.pdf* documentation.

- A new console named SWO and Trace Console [274] displays all configuration and register settings performed while configuring SWO.
- [J-Link] Added possibility to connect to a remote gdb server. In launch configuration -> J-Link Debugger tab -> GDB Server Settings, use Server execution option to set a remote server.
- UART console is the default debug console when importing a project.
- The IDE displays inside Probes Discovered dialog the nickname assigned to a PEmicro or J-Link debug probe. Each solution offers a specific procedure for assigning a nickname, thus the appropriate documentation must be followed.
- Auto-debug secondary project(s) for multicore projects option becomes default option for multicore debug purpose for LinkServer debug connection. That means, in the case of multicore projects on which primary project refers one or several secondary projects, debug sessions will be automatically started for secondary projects after initiating debug with the primary project.
  - Option is set by default on: **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options -> Miscellaneous -> Enable auto-debug secondary project(s) for multicore projects**.
  - If you don’t want to have this feature enabled (so if you want to start debug sessions for each core independently), uncheck this option.
- Similar, auto-debug secondary project(s) for multicore projects option becomes default option for multicore debug purpose for PEmicro too. Option is enabled by default on: **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> PEMicro Options -> Enable auto-debug secondary project(s) for multicore projects** and also for J-Link debug session, set by default on: **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> J-Link Options -> Enable Auto-debug secondary project(s) for multicore projects**.
- Firmware version check on MCU-Link probes [115].
- Most LinkServer Flash Programming [103] now implements a *Verify Same* operation for any flash sector that are unchanged from previous debug operations
- LinkServer MultiCore debug operations can now be started via a single click
- Reworked Live Global Variables [144] graphing offering improvements to variable selection and display
- Reworked SWO Interrupt trace
- LinkServer LPC-Link2 firmware now softloaded as v5.361 which offers improved debug control through target reset
- Redesigned LinkServer Launch configuration [95] dialogue offering improved functionality and ease of use
  - this is reflected in a new LinkServer Launch configuration icon
- New launch configuration tab for all debug solutions to allow the loading of Debug Symbols [129] from additional images
- Improved performance for Single Stepping LinkServer debug connections
- Implemented support for SWO Trace on Cortex-M33 based MCUs
- Live Global Variables [144] are now available for SEGGER JLINK and PEmicro debug probes in addition to LinkServer LPC-Link2
- LinkServer internal flash drivers prioritised over supplied SDK drivers
- Debug Shortcut buttons [123] now Multicore aware ensuring secondary project attach settings are observed
- Improved Faults View [137] now displays Fault Address when available
- SWO trace features are now available for SEGGER JLINK and PEmicro debug probes in addition to LinkServer LPC-Link2 and MCU-Link
- LinkServer debug probes now support selection via their serial number (for command line use)
- Increased integration of our supported debug solutions including:
  - GUI Flash Tool [167] is re-architected to provide support for LinkServer, PEmicro and SEGGER debug solutions

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

    - offering binary flash programming and erase capability for all supported debug solutions
    - with a feature set integrated into the Quickstart panel, project Launch Configurations and from the IDE as before
  - Instruction trace is seamlessly supported by LinkServer, PEmicro and SEGGER debug solutions
- LinkServer Semihosted Operations [192] including printf are further optimised to deliver approximately double the performance of the previous release
- Re-architected Semihosting Mechanism [192] via new library variant *Redlib MB* and LinkServer which can deliver both a further increase in performance and no disruption to code executing with time critical interrupts
- LinkServer Graphing of Global Variable Values [147]
  - *Live* global variable values can now be traced both in graphical and tabular forms
- Peripheral Display Filtering to simplify complex peripheral views

LinkServer Flash Programming

- External flash drivers for RT116x, RT117x, RT500 and RT600 available as examples in <install_dir>/ide/Examples/Flashdrivers/NXP/iMXRT.
- SFDP Flash Drivers [177] extended to support iMX RT MCUs
- Programming of data flash regions on certain Kinetis parts is now supported
- Improved flash programming performance and reliability
- LinkServer Enhanced External SPIFI/QSPI programming [177] via self configuring flash drivers
  - using JEDEC SFDP (Serial Flash Discovery Protocol) available for LPC18/43, LPC546xx, LPC540xx, iMX RTxxx, iMX RTxxxx

SDK

- Provide CLI utility to merge sub-manifest files: added the manifest.merge command:
  - Running the headless mode with -help manifest.merge generates a template property file, which contains the following:
    - manifest.xml (location of the manifest containing references to sub-manifests)
    - repo.location (repository where the manifest specified in the manifest.xml property is located).
    - merged.manifest.xml (location of the result manifest file).
  - All properties from the template file must be specified for the command to run.
  - The manifest specified in the manifest.xml file must be inside the specified repository.
- Complex dependencies: support for < not > operator in the dependency conditions.
- Added possibility to explore Open-CMSIS packs and import (middleware) components into an Eclipse project.Note that in the current version of the feature, component dependencies shall be manually added by users. A future version will automatically resolve dependencies and add them into the project:
  - CMSIS-Pack Management for Eclipse created by ARM plugin included in the product for packs management: Perspective -> Open Perspective -> Other -> CMSIS-Pack Manager. From Packs view (toolbar) you can: Reload, Check for updates on Web, Import Packs from disk, etc.
  - Once desired packs are available, you can add them into the Eclipse project by right-click on project entry in Project Explorer -> SDK Management -> Add components from Open-CMSIS-Pack and select the desired one from the “Add Open-CMSIS component to project” wizard. The component will be then available on Project Explorer view (the sources being linked to the original pack location), and also on Project Settings with details about its hierarchical path.
  - Components can be deleted from project by selecting the component from Project Explorer -> <select project> -> Project Settings -> Open-CMSIS components, right-click on it, and choose “Delete Open-CMSIS component”.
- Support to allow sub-manifest under same SDK.
  - Adapted SDK Creator for creating split manifests.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

    - Updated “Contribute project to SDK Git repository” feature to work with the new sub-manifests.
- A GitHub SDK repository can be imported and managed within IDE, integrating existing IDE’s SDK management functionality and git capability.
    - Support to import [36] an already cloned repository;
    - Support to install/clone [33] a remote repository (using west init and west update);
    - Ability to contribute [50] a project back to SDK GitHub repository;
- Improved SDK installation and refresh time
- Redesigned New and Import SDK example wizard
    - incorporating Error Decorators
- SDK part support is now generated within the current workspace eliminating issues that could arise if multiple IDEs were launched
    - part support is intelligently regenerated when required avoiding unnecessary delays
- SDK drag and drop location [40] can now be set via a workspace preference
- Installed SDK view improved to display version information and enhanced tooltips
- SDK Manifest Analyser to provide visibility of SDK XML description
- Easy access to Embedded Documentation [36]
- Extension of SDK Component Management to allow Project Refresh [74]
    - improved SDK Component Management
- General Improvements in SDK Handling including:
    - SDK version string now present and reported in SDK view
    - user selection of versioned internal XML descriptions (enabled via preference)
    - better automatic support for SDKs with overlapping capabilities

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

3. IDE Overview

The following chapter provides a high level overview of the features offered by MCUXpresso IDE (often referred to as the IDE).

3.1 Workspaces

When you first launch MCUXpresso IDE, you will be asked to select a Workspace, as shown in Figure 3.1.

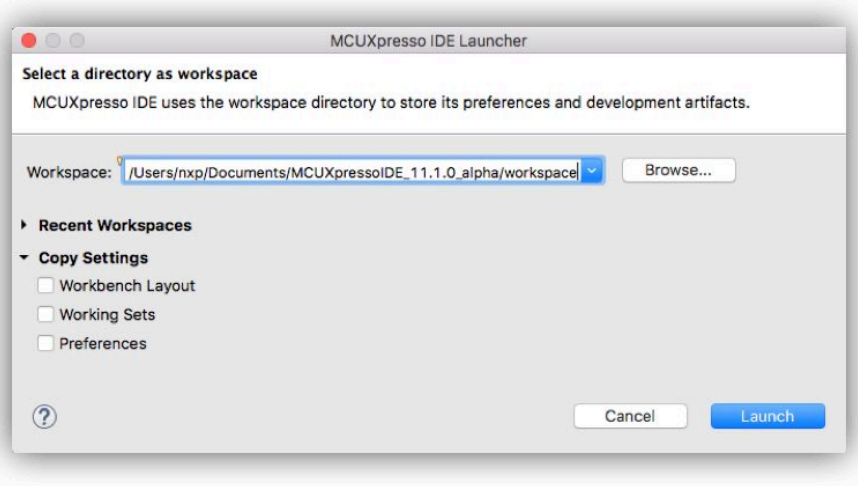


Figure 3.1. Workspace selection

A Workspace is simply a filing system directory used to store projects and data, and the recommended default location should typically be accepted for new installations. If you tick the **Use this as the default and do not ask again** option, then MCUXpresso IDE will always start up with the chosen Workspace opened; otherwise, you will always be prompted to choose a Workspace.

MCUXpresso IDE can only access a single Workspace at a time but many Workspaces may be used. You may change the Workspace that MCUXpresso IDE uses, via the **File -> Switch Workspace** option.



Tip

It is possible to run multiple instances of the IDE in parallel with each instance accessing a different Workspace.

Note: when changing workspaces, you may choose to copy settings (preferences) from an existing workspace to the new workspace using the various *Copy Settings* tick box options.

3.2 Welcome View

MCUXpresso IDE version 11.1.0 launches with a new Welcome View. This View is intended to help reduce the learning curve for new users by offering links and help for common tasks and IDE operations.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

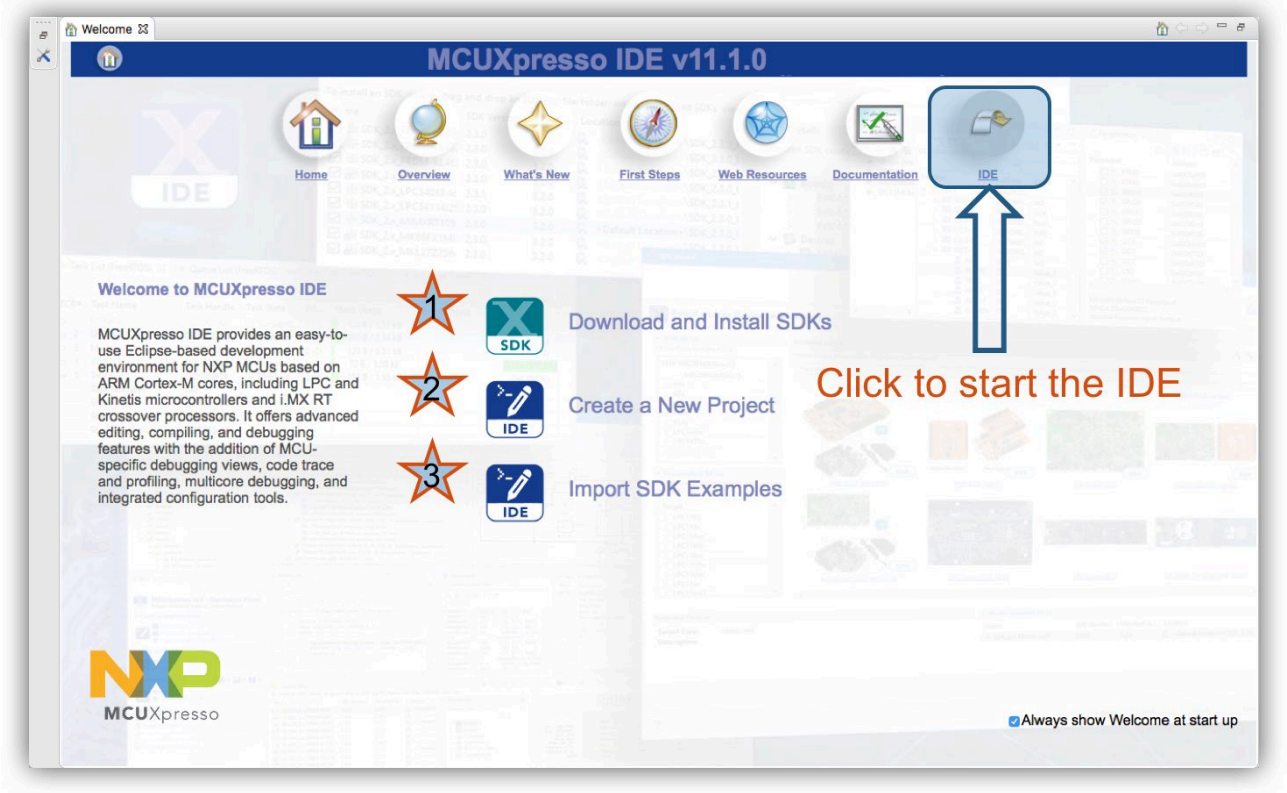


Figure 3.2. Welcome View

1. Click to select, download and install a Plugin SDK [29]
2. Click to be guided through Creating a New Project [53]
3. Click to be guided through Importing an Example [62]

Since support for most NXP MCUs is added to the IDE via the installation of an SDK, the first option is to guide the user to a new Plugin SDK [29] installation view. From this view, an SDK for a required MCU or (development board) can be selected, downloaded and installed with just a few clicks. Also from this screen are guided workflows for creating New Projects and Installing SDK Examples.

Across the top of this View are links to Features and Resources including a *jump to* IDE link (highlighted above) which takes the user directly to the IDE's main development view (Perspective).

Note: This Welcome View is provided by Eclipse functionality and so incorporates standard icons to maximise, minimise and restore etc. like all Eclipse views. Since this view is intended to be used *full screen*, minimising or restoring may lead to a poor screen layout. The recommended way to switch back to the main IDE Develop view is via the IDE link or by closing this Welcome Screen. The Welcome View can be restored at any time by clicking the Home Icon within the main Eclipse Icon view.

The Welcome view can be disabled from appearing at startup by unchecking the box at the lower right of the view.

3.3 Documentation and Help

In addition to the help features offered from the Welcome View are a comprehensive suite of Guides.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

MCUXpresso 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 MCUXpresso IDE's **Help -> Help Contents** menu. It also provides access to the MCUXpresso IDE User Guide (this document), as well as the documentation for the compiler, linker, and other underlying tools.

MCUXpresso IDE documentation comprises a suite of documents including:

- MCUXpresso IDE Installation Guide
- MCUXpresso IDE User Guide (this document)
- MCUXpresso IDE SWO Trace Guide
- MCUXpresso IDE Instruction Trace Guide
- MCUXpresso IDE LinkServer Energy Measurement Guide
- MCUXpresso IDE FreeRTOS Debug Guide
- MCUXpresso IDE Azure RTOS ThreadX Debug Guide
- MCUXpresso IDE Zephyr RTOS Debug Guide
- MCUXpresso IDE MQX RTOS Debug Guide
- MCUXpresso (IDE) Config Tools User's Guide

These Guides are also supplied in *PDF* format within the MCUXpresso IDE's installation folder.

To obtain assistance on using MCUXpresso IDE, visit: https://www.nxp.com/mcuxpresso/ide

Related web links can be found at *Help -> Additional resources* as shown below:

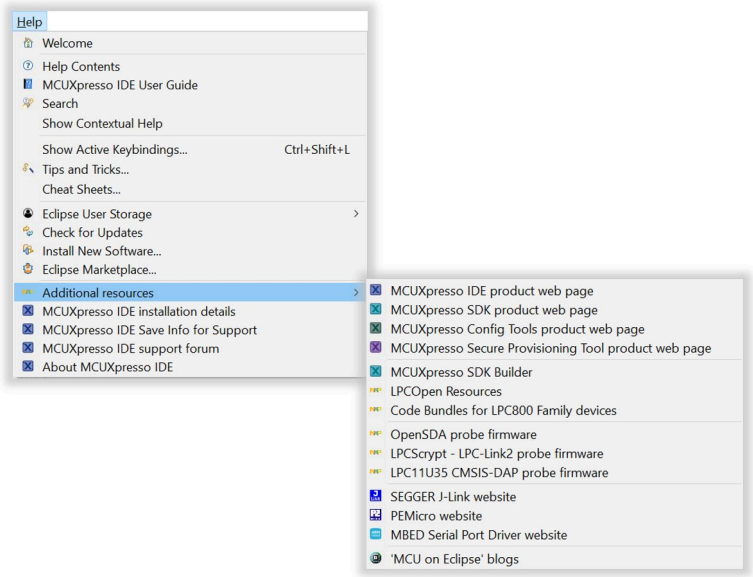


Figure 3.3. Additional Resources

3.4 Perspectives and Views

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

Typically, MCUXpresso IDE operates using the single **Develop Perspective**, under which both code development and debug sessions operate as shown in Figure 3.6. This single perspective

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

simplifies the Eclipse environment, but at the cost of slightly reducing the amount of information displayed on screen.

Alternatively, MCUXpresso 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.

Note: when within the debug perspective, the concept of a selected project remains. The *Blue Debug* button tool tip will display this selected project. Also, if a debug operation is started within the Debug perspective and a switch is made to the Develop perspective, the IDE will automatically open a debug stack view to display the active debug connection.

You can manually switch between Perspectives using the Perspective icons in the top right of the MCUXpresso IDE window, as shown in Figure 3.4.



Figure 3.4. Perspective selection

New perspectives can be selected by clicking the view+ icon. Once a view has been selected, its icon will appear within the horizontal section as highlighted above.

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 -> Perspective -> Reset Perspective**.

Commonly used Views for Analysis (Trace) and RTOS debugging have been made more readily available via top level drop down menus as shown below:

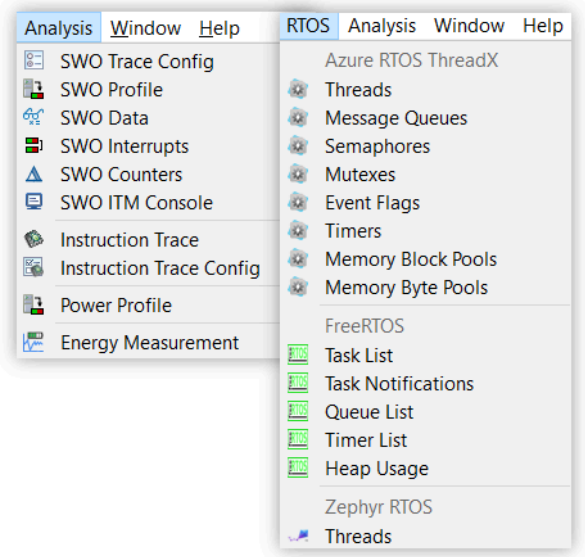


Figure 3.5. Additional Views

One selected, these additional views will appear alongside the Console view but can be relocated as desired.

Note: The rest of this guide assumes the default Develop Perspective is used.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

3.5 Major Components of the Develop Perspective

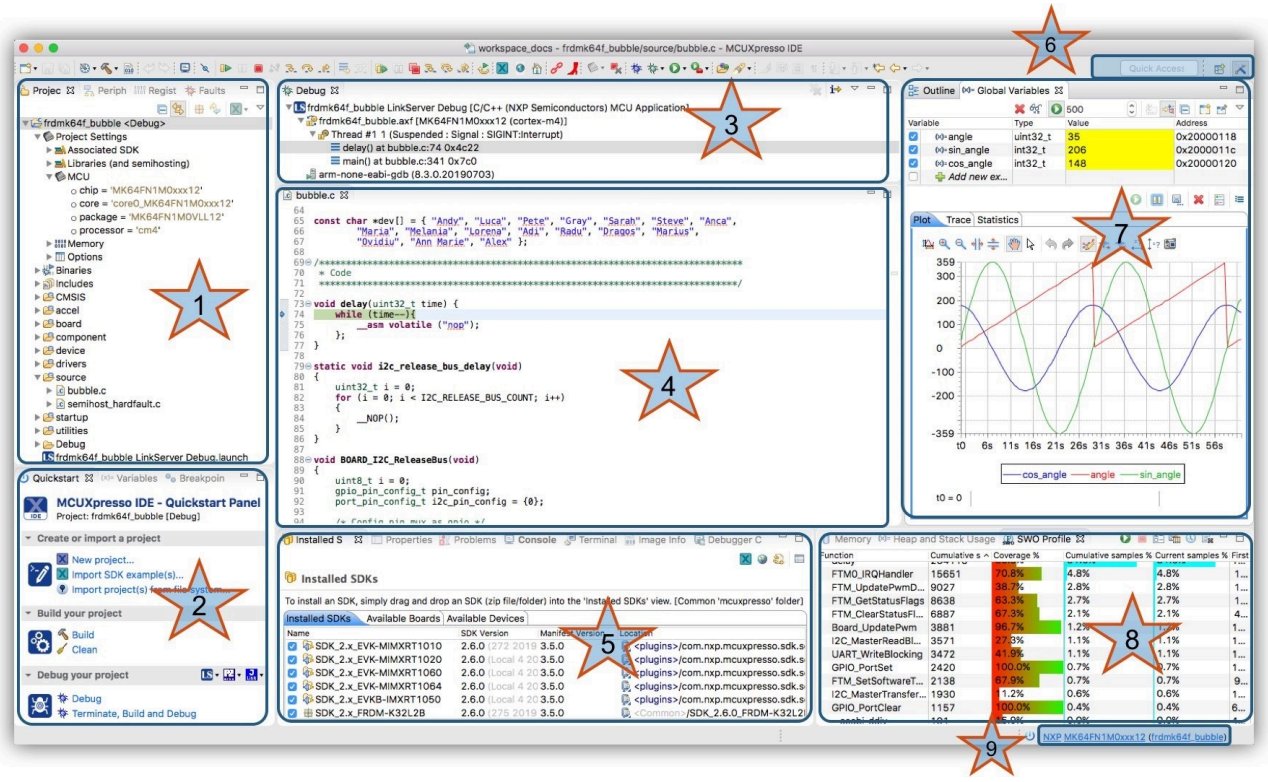


Figure 3.6. Develop Perspective (whilst debugging)

1. **Project Explorer / Peripherals / Registers / Faults**
   - The **Project Explorer view** (shown) gives you a view of all the projects within your current Workspace. [14]
     - Many editing and configuration features are available from this view including new Project Sharing [45] options and Virtual Nodes [155]
   - When debugging, the **Peripherals** view allows you to display a list of the MCU Peripherals [139] and project memory regions. **Note**: depending on your MCUs configuration, some peripherals may not be powered/clocked and hence their content will not display.
   - When debugging, the improved **Registers** view allows you to view the Registers [135] and their content within the CPU of your MCU.
     - Pseudo registers are also displayed here such as 'cycle delta' which shows the calculated number of cycles since the last pause
   - Also displayed here is the **Faults** view, which will appear automatically if a CPU Fault [137] (such as hard fault) occurs. This view decodes CPU registers to provide detailed information indicating the reason for the fault occurring.
2. **Quickstart / Variables / Breakpoints**
   - On the lower left of the window, the **Quickstart Panel View** (shown) has fast links to commonly used features. From here you can launch various wizards including New Project, Import projects from SDK and also from the File System plus options such as Build, Debug, Export etc.. The large icon in each section will perform the first option in the group i.e. New project, Build, Debug. Also, the Debug group contains debug solution specific Debug Shortcut buttons [123].
     - **Note**: This Panel is essential to the operation of MCUXpresso IDE and so cannot be removed from the perspective.
   - Sitting in parallel to the Quickstart Panel, the **Variables** View allows you to see and edit the values of local variables.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Sitting in parallel to the Quickstart Panel, the **Breakpoints View** allows you to see and modify currently set Breakpoints [131] and Watchpoints [133] .

3. **Debug**
   - The Debug View appears when you are Debugging [112] your projects. This view shows you the debug stack, in the “stopped/paused” state you can click within the stack and inspect items in scope such as local variables.
4. **Editor**
   - Centrally located is the **Editor**, which allows creation and editing of source code and other text files. 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 from source to assembly instructions. Clicking in the left margin will set and delete Breakpoints [131]
     - Enhanced Editors [209] provides structure, keyword and linkage for debug Map files, Linker Script and Linker Template files.
5. **Console / Installed SDKs / Problems / Trace Views / Power Measurement**
   - On the mid lower of the window are Console, Installed SDK and Problems Views etc. The Console View displays status information on compilation and debug operations, as well as displaying semihosted program output.
   - The Installed SDK [27] view (shown) enables the management of installed SDKs. New SDKs can be added as Plugins via Drag and Drop or Copy and Paste. Other SDK management features are also provided from this view including unzip, explore and delete. Details of any selected SDK can be viewed in Outline view.
     - SDK Documentation can be browsed and extracted
   - The **Problems View** shows all compiler errors and warnings and will allow easy navigation to the error location in the Editor View.
   - The **Image Information View**
     - This Image Information [203] view provides detailed information on an image (or object) static memory footprint (usage and content).
6. **Quick Access/Perspective Selection**
   - Enables quick access to features such as views, perspectives etc. for example enter ‘Error’ to view and open the IDE’s Error Log, or ‘Trace’ to view and open the various LinkServer Trace views.
   - Perspective Selection allows you to switch between the various defined perspectives.
7. **Outline / Global Variables**
   - The **Outline** View allows you to quickly locate symbols, declarations, functions within the editor view. This view can also display details of any SDK selected in the Installed SDK view.
   - Sitting in parallel is the **Global Variables View** (shown) which allows you to see and edit the values of Global variables.
     - Variables can be monitored while the target is running using the Live Variables [144] and Variable Graphing [147] features.
8. **Memory / Heap and Stack / Trace**
   - The **Memory View** provides a range of options for viewing target memory
   - The **Heap and Stack View** enables easy monitoring of Heap and Stack [150] values for bare metal projects.
     - Warnings are given when preset limits are approached or exceeded
   - **Trace Views**
     - Trace Views including SWO Trace (Profiling shown), Instruction Trace, and Power are not shown on this screenshot, however these views may be selected when required from the Analysis Menu. For more information on Trace functionality, please see the *MCUXpresso IDE SWO Trace Guide* and/or the *MCUXpresso IDE Instruction Trace Guide* and/or the *MCUXpresso IDE LinkServer Power Measurement Guide*.
     - The **SWO Trace Views** allow you to gather and display runtime information using the SWO/SWV technology that is part of Cortex-M0+/M3/M4/M7/M33 based parts.
     - The **Instruction Trace view** on certain MCUs, you can capture and view instruction trace data downloaded from the MCU’s Embedded Trace Buffer (ETB) or Micro Trace Buffer (MTB).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- The **Power Measurement View**, this view is capable of displaying real-time target power usage. For more information please see the *MCUXpresso IDE Power Measurement Guide*.

9. **Status Bar Shortcuts**
   - Various useful shortcuts, for example to open a project's workspace or to open a terminal at the project's location with the IDE's environment. Hover here and tooltips will explain the various options.

3.5.1 Dark Theme

MCUXpresso IDE contains support for a *Dark Theme*. Dark Theme is a Workspace preference that can be selected from *Window -> Preferences -> Appearance -> Theme* followed by a selection from the drop down menu.

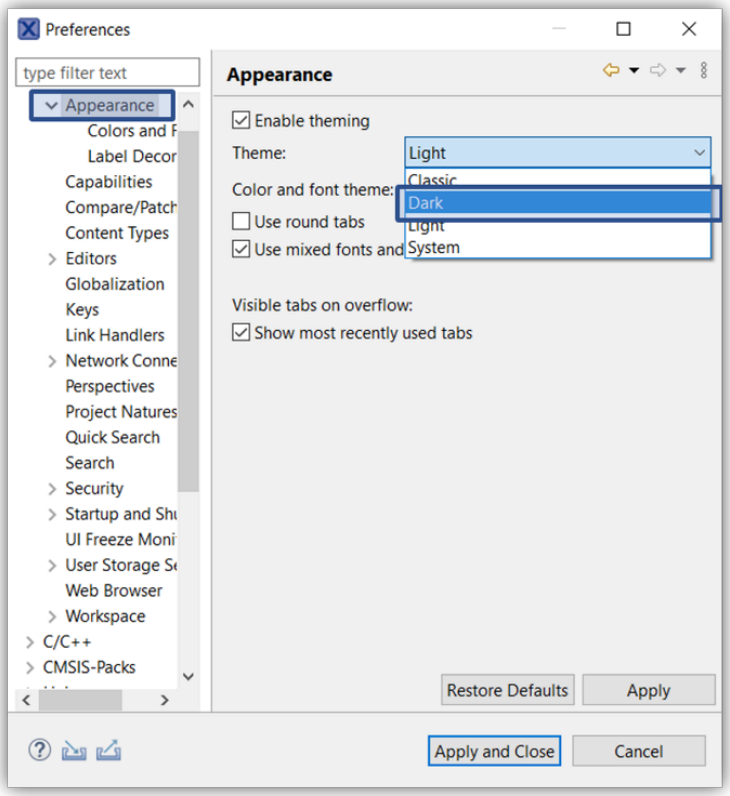


Figure 3.7. Appearance Preference

When selected, a Dark theme with be used to render the perspective and will appear similar to the image below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

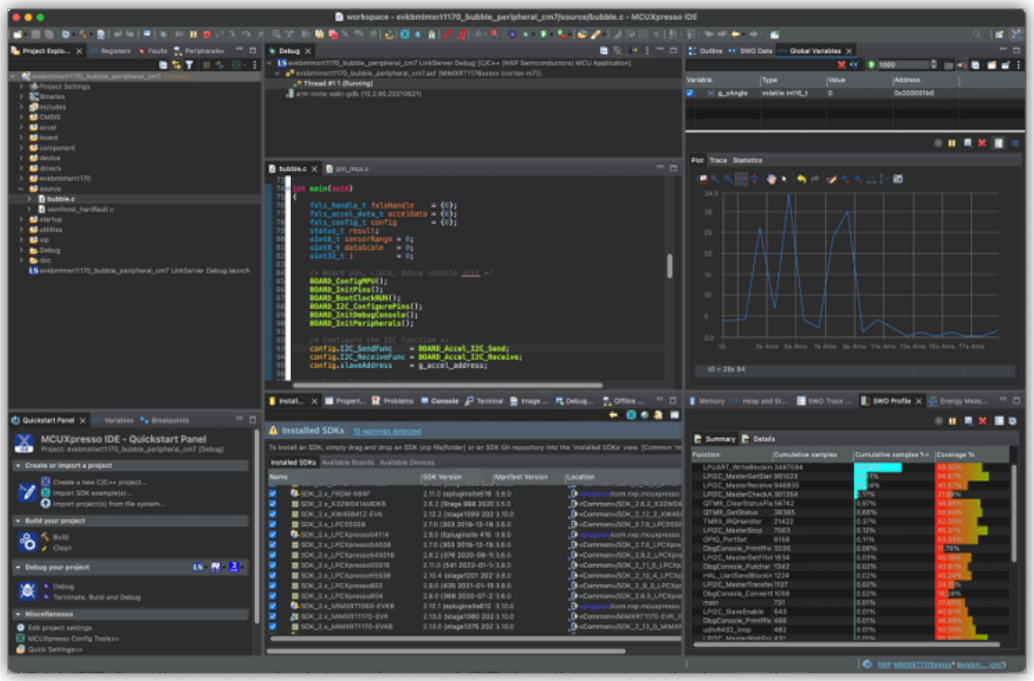


Figure 3.8. Develop Perspective Dark

Note: An IDE restart *File -> Restart* is required for the perspective to display correctly.

3.6 The Quickstart Panel

A key feature of MCUXpresso IDE is the **Quickstart** Panel – which is frequently referenced with this document. The Quickstart panel is designed to bring together many of the common IDE features and operations including links to Project Creation, Project Building, Project Debug and Miscellaneous common Project operations.

It is **strongly recommended** that this panel is used to perform the supported MCUXpresso IDE operations described below since many underlying Eclipse features are enhanced when accessed in this way to improve and simplify the user experience.

Features of the Panel are highlighted and described below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

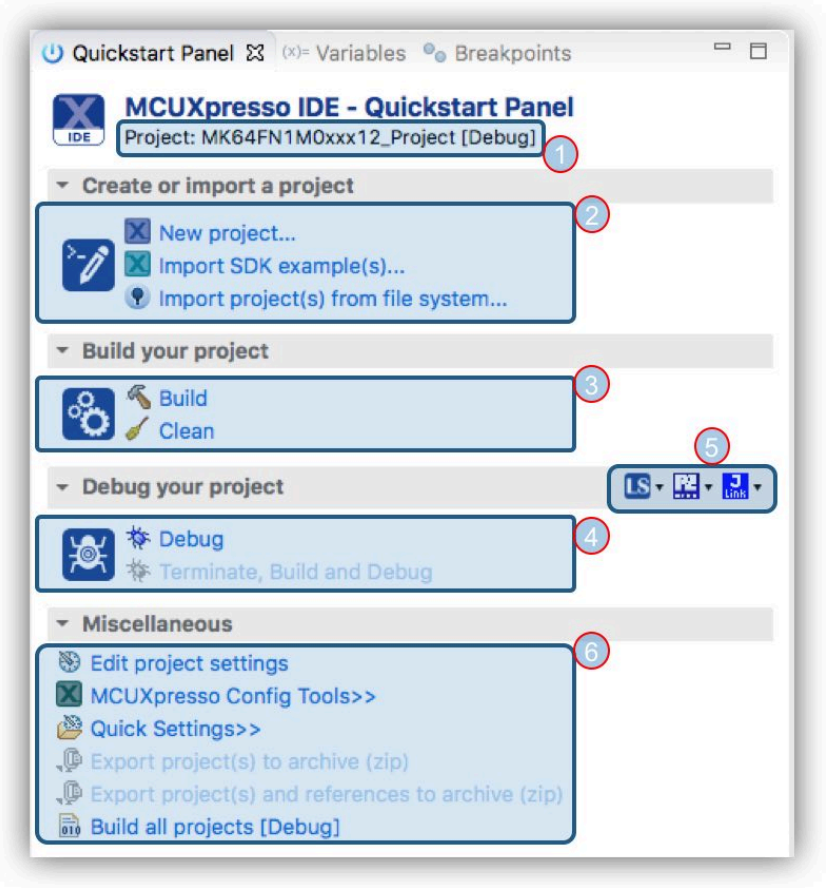


Figure 3.9. The Quickstart Panel



Tip

The Large Icon performs the action of the first button in the group

Where:

1. Shows the Project currently selected within the *Project Explorer* view. Build, Debug and Miscellaneous operations will be performed to this Project
2. Links to Create New [53], Import SDK [62] and Import from the FileSystem [84]
3. Build [60] (or Clean) the currently selected Project
   - See progress and results within the Console [273] view
4. Debug the currently selected Project
   - Clicking Debug [112] will by default Build the project (if necessary), perform a Debug Probe Discovery, create a default Launch Configuration [95] (if necessary) and if successful, begin the debug session.
   - Terminate, Build and Debug, Terminates the existing Debug session for the selected project, and them performs another debug operation. It is intended to be used for iterative source code fix and debug retry operations
5. Debug Shortcuts [123] offer a range of debug operations for specific vendor Debug Solutions
6. The Miscellaneous section offers a range of option and shortcuts
   - Edit project settings is a shortcut equivalent to a right click on a project and then selecting *Properties*
   - MCUXpresso Config [164] Tools offers shortcuts to launch one of the Config tools for the selected project
   - Quick Settings [154] offers a range of options for the currently selected project

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Export the selected Project (and References) to the file system. See also additional information on Sharing Projects [45]
  - this feature requires that the project is selected at the top level within Project Explorer
- Build the *Active* Build Configuration of all projects within the current Workspace.



Tip

If the Quickstart panel has become hidden, then in the menu bar at the top of the IDE, select *Window -> Show View -> MCUXpresso IDE* and double-click on Quickstart

The Quickstart panel is directly linked to active selection from Project Explorer, which controls the enablement state of various actions within the view. However, when the Quickstart panel is visible it also controls automatic selection of the debugged project when various debug launch events are encountered. Behavior can be adapted by accessing the appropriate preference page: *Window -> Preferences -> MCUXpresso IDE -> Quickstart Panel*. The project associated with the active debug session can be auto-selected in Project Explorer when adding, terminating or removing a launch from Debug view. If a child resource of the project is already selected when a launch event occurs, selection will not be changed.

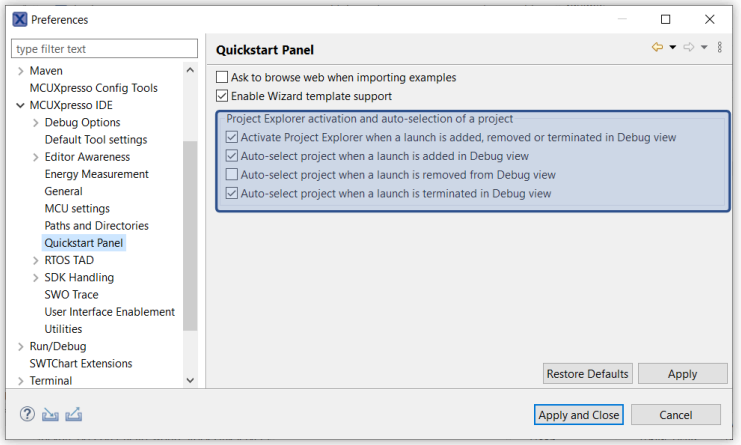


Figure 3.10. Quickstart panel Preferences

3.7 Project Explorer and New Projects

The version of Eclipse underlying MCUXpresso IDE incorporates some new **Project Explorer** functionality that is seen only when there are **no projects** within the chosen Workspace [14] - as shown below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

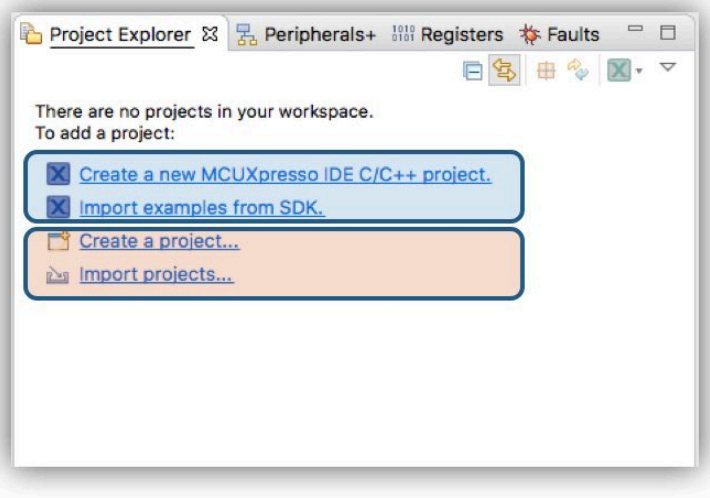


Figure 3.11. Project Explorer Empty

The first two options here are directly equivalent to the first two operations offered via the **Quickstart** panel. It is recommended that the **Quickstart** is used in preference to the remaining options since this ensures that MCUXpresso IDE wizards and functionality is used.

Note: Due to this Eclipse feature, Drag and Drop functionality to the Project Explorer view is unavailable until the first project is created or imported.

New or Imported Projects will appear in the Project Explorer view. A newly created project will automatically expand to show the source file containing the main function. This source file will also be opened into the editor for convenience as shown below.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

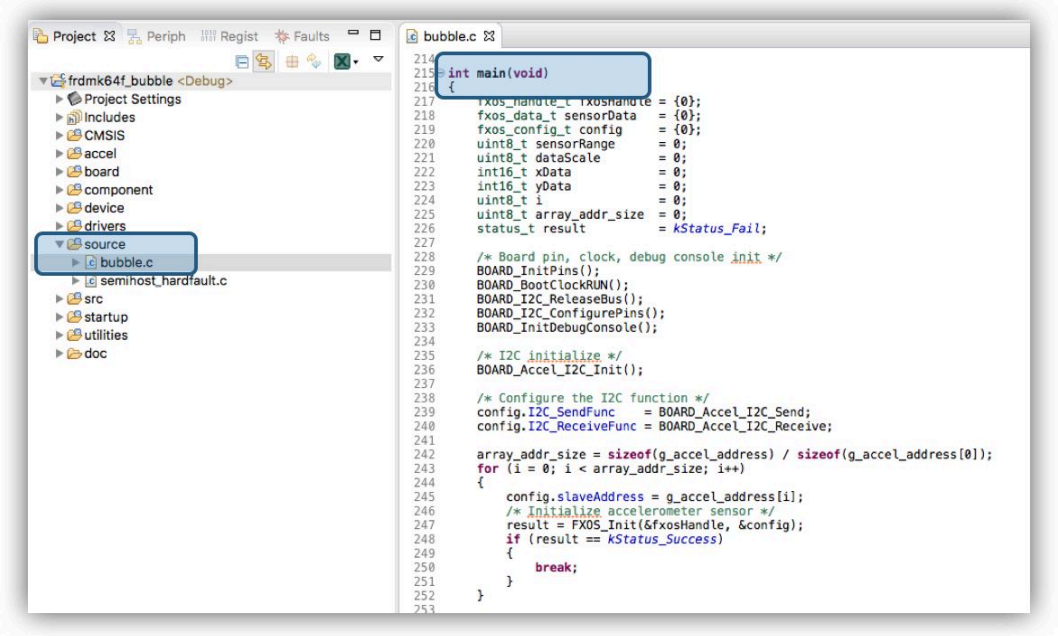


Figure 3.12. New or Imported Project

3.8 Updating MCUXpresso IDE

MCUXpresso IDE incorporates the facility to update an installation to add new features, updates and/or to roll out bug fixes etc. To facilitate this mechanism, MCUXpresso IDE version internals locate key components with Eclipse style plugins.

Tip

Locating low level components can be difficult due to both the complex directory structure but also because component locations may change after an update is performed. Therefore, to simplify the experience a number of softlinks are available within the *install_dir/ide* as discussed in section below “Locating IDE Components”

By default, when an update is released by NXP, a notification of the availability will appear at the bottom of the screen.

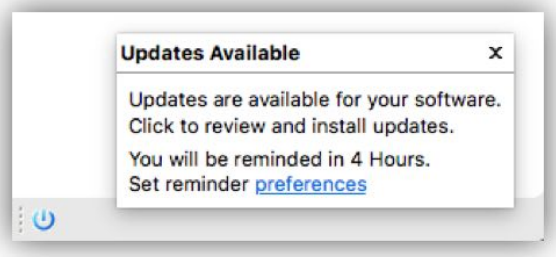


Figure 3.13. Update Notification

Alternatively you can check for updates via *Help -> Check for Updates*. If updates are available you will be presented with a dialogue similar to that below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

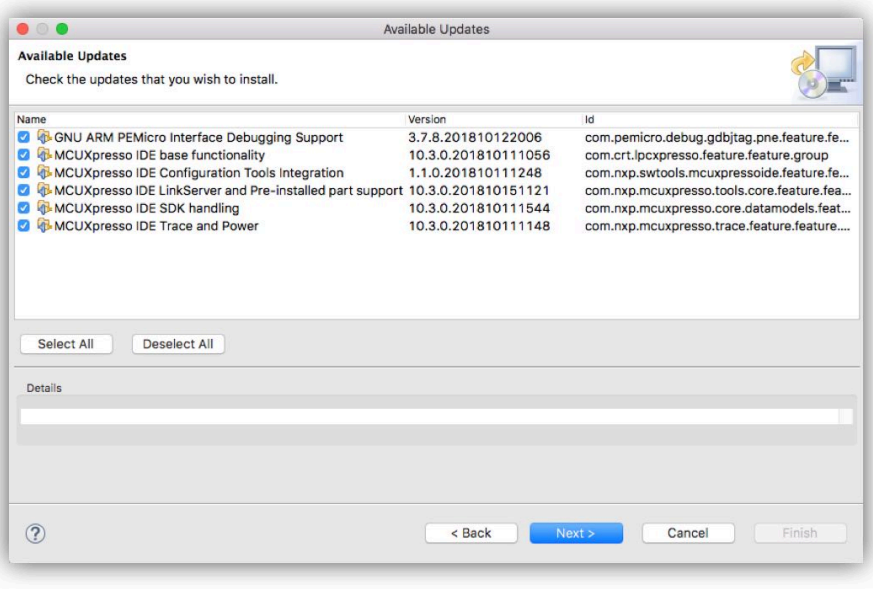


Figure 3.14. Updating MCUXpresso IDE Components

Simply, ensure the required updates are checked and click *Next*. At this point the components will be downloaded and installed into MCUXpresso IDE. After installation, a restart will be required before new features are available.

Note: In addition to updates for MCUXpresso IDE, updates to the MCUXpresso Config tools and PEMicro debug solution are also delivered using this mechanism.

Major product releases will only be delivered as full product installations since these are typically based on newer versions of Eclipse

3.8.1 Locating IDE Components

MCUXpresso IDE consists of many components, some of which may be used independently from the IDE. Also included are documents, examples, scripts, drivers etc. that may need to be referenced from within the IDE.

Due to the structural changes introduced in MCUXpresso IDE version 10.3.0, the paths for certain items may be both different from previous releases and may change after a product update (and also be quite long). For example, the IDE binaries folder will now be at a location of the form:

```
<install_dir>/ide//plugins/com.nxp.mcuxpresso.tools.bin.macosx_11.1.0.201911211415/binaries
```

Therefore, to simplify the location of certain folders, shortcuts (or symbolic links) are installed into the *ide* directory within the product's installation directory. These links can be used directly to locate components or items, or within script paths.
Shortcuts are available for the following directories:

- binaries -> install_dir/ide/binaries
- Examples -> install_dir/ide/Examples
- Wizards -> install_dir/ide/Wizards
- tools -> install_dir/ide/tools

In practice, these links will allow paths to be used unchanged from earlier version of MCUXpresso IDE, yet will always reference the latest plugin components.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

4. Part Support Overview (Preinstalled and via SDKs)

To support a particular MCU (or family of MCUs) and any associated development boards, a number of elements are required. These break down into:

- Startup code
  - This code will handle specific features required by the MCU
- Memory Map knowledge
  - The addresses and sizes and types of all memory regions
- Peripheral knowledge
  - Detailed information allowing the MCUs peripherals registers to be viewed and edited
- Flash Drivers
  - Routines to program the MCU's on and off chip Flash devices as efficiently as possible
- Debug capabilities
  - Knowledge of the MCU debug interfaces and features (e.g. SWO, ETB)
- *Example Code* (this is not strictly required or a part support element)
  - Code to demonstrate the features of the particular MCU and supporting drivers

Collectively, this data is known as *Part Support*, MCUXpresso IDE uses these data elements for populating its wizards and for built in intelligence features, such as the automatic generation of linker scripts etc.

MCUXpresso IDE installs with a base set of part support primarily for older LPC Devices (Preinstalled). Knowledge of later devices such as the LPC5xxxx, Kinetis, iMXRTxxx etc. must be provided to the IDE via the installation of an SDK [29] .

4.1 Preinstalled Part Support

The IDE installs with an enhanced version of the part support as provided with the older NXP IDE *LPCXpresso IDE v8.2.2*. This provides support for the majority of LPC Cortex-M based parts 'out of the box'. This is known as preinstalled part support. In general, SDKs are not available for these older parts however, the LPC5410x and LPC5411x part families can be used with either Preinstalled Part Support or SDK Part support.

Example code for these preinstalled parts is provided by sophisticated LPCOpen packages (and Code Bundles). Each of these contains code libraries to support the MCU features, LPCXpresso boards (and some other popular ones), plus a large number of code examples and drivers. Version of these are installed by default at:

```
<install dir>/ide/Examples/LPCOpen
<install dir>/ide/Examples/CodeBundles
```

Further information can be found at:

https://www.nxp.com/lpcopen

https://www.nxp.com/LPC800-Code-Bundles

4.1.1 Differences in Preinstalled and SDK Part Handling

Since SDKs combine part (MCU) and board support into a single package, MCUXpresso IDE is able to provide linkage between SDK installed MCUs and their related boards when creating or importing projects.

For preinstalled parts, the board support libraries are provided within LPCOpen packages and Code Bundles. It is the responsibility of the user to match an MCU with its related LPCOpen board and chip library when creating or importing projects.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Creating and importing project using Preinstalled and SDK part support is described in the following chapters.

Note: When exporting or sharing projects with created with Preinstalled part support, no special actions are required, since other installations of MCUXpresso IDE will also contain the required part support. For sharing projects created from SDKs, please see Sharing Projects. [45]

4.1.2 Viewing Preinstalled Part Support

When MCUXpresso IDE is installed, it will contain preinstalled part support for most LPC based MCUs.

To explore the range of preinstalled MCUs simply click ‘New project’ in the **Quickstart** panel. This will open a page similar to the image below:

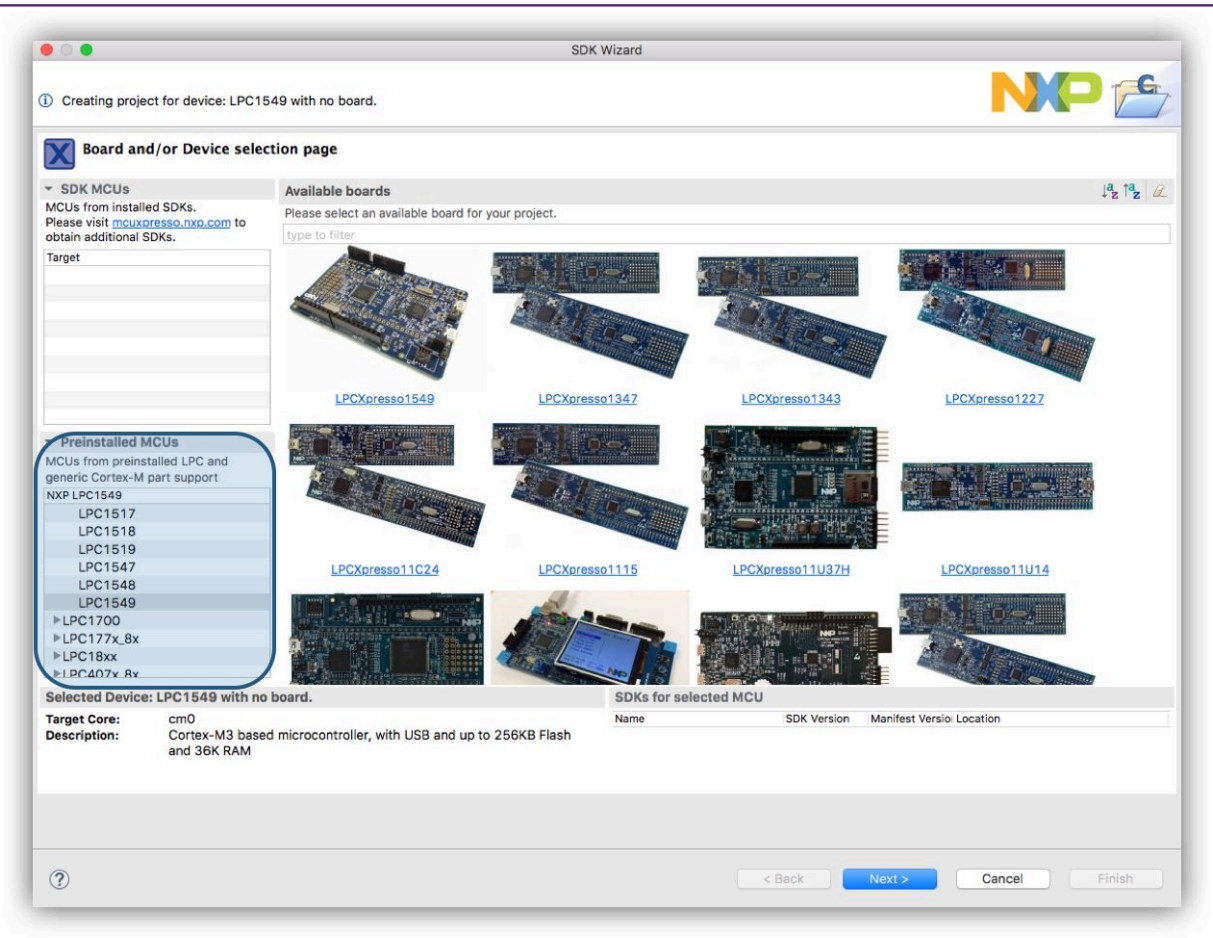


Figure 4.1. New Project Wizard

The list of preinstalled parts is presented on the bottom left of this window.

You will also see a range of related development boards indicating whether a matching LPCOpen Library or Code Bundle is available.

For creating project with preinstalled part support please see: Creating Projects with Preinstalled part support [77]

If you intend to work on an MCU that is not available from the range of preinstalled parts, for example a Kinetis MCU, then you must first extend the part support of MCUXpresso IDE by installing the appropriate MCU SDK.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

4.2 SDK Part Support

The IDE's Part Support can be extended using freely available MCUXpresso SDK v2.x packages. SDK 2.x packages are used to add support for all Kinetis, iMX RT and newer LPC MCUs etc.

Starting with MCUXpresso IDE version 11.1.0 there is a streamlined approach to the supply and installation of SDKs – these SDKs are known as **Plugin SDKs**. Plugin SDKs are pre-built SDKs hosted on NXP's servers that can be browsed, downloaded and installed directly from within the IDE when required. See Obtaining and Installing a Plugin SDK [29]

Each SDK will install as an Eclipse plugin and so benefit from the standard Eclipse management and update mechanisms. MCUXpresso IDE Plugin SDKs are available for a wide range of NXP's MCUs. Like all Eclipse plugins, once Plugin SDKs are installed, they become part of the product itself. Management of a Plugin SDK can be performed using the standard Eclipse mechanisms [39]

The previous *Classic* method of SDK installation and handling is still available. See SDK Part Support via SDK Builder [31]

Once an SDK has been installed, the included part support becomes available through the New Project Wizard and also the SDK example import Wizard, and for use by imported projects. [45]

4.2.1 Obtaining and Installing a Plugin SDK

SDKs are installed and managed via the *Installed SDKs* view, which is located by default as the first tab within the Consoles view. See Major Components [18] item 3 for more information. Plugin SDK installation can also be started via the New Welcome system and via the *Download and Install SDKs* icon on the main IDE icon bar.

Once launched a dialogue similar to that below will be displayed:

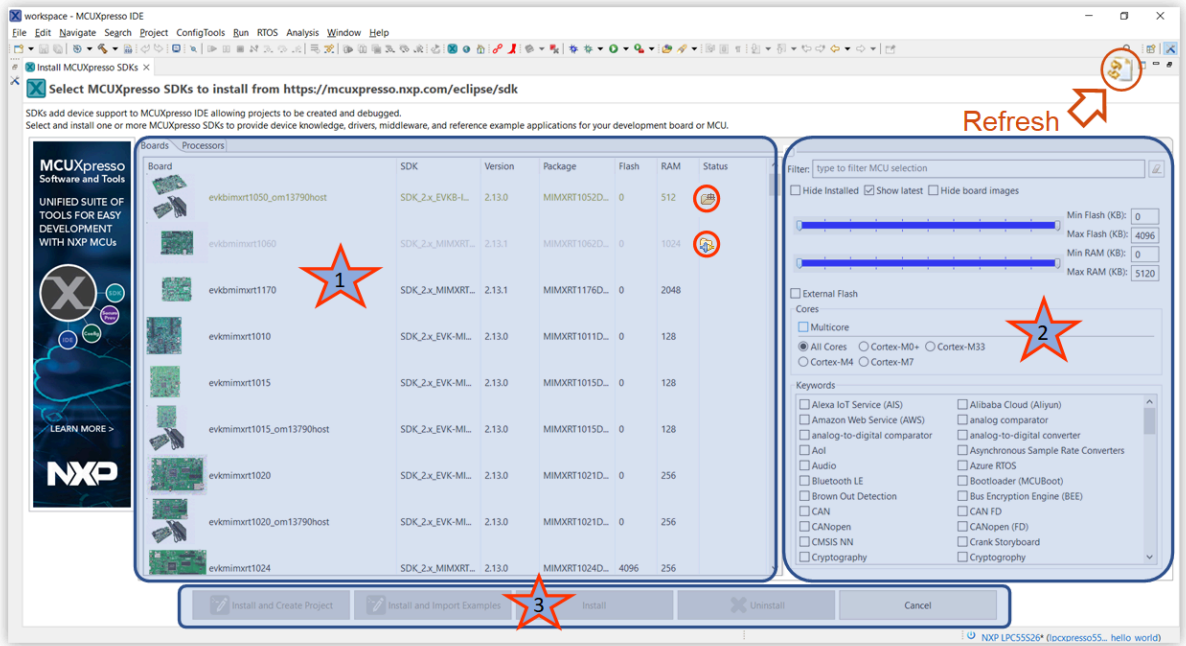


Figure 4.2. Plugin SDK Installation

1. From this section the SDK for the desired Board (or Processor) can be selected for installation. Column sorting is supported to help location and options for filtering the list are discussed below

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- By default, SDKs that are already installed are hidden from this view
- If the *Hide Installed* is unchecked, installed SDKs will also be shown along with Status indication for the SDKs already installed (shown as red circle)

2. A range of filtering options may be selected to reduce the list of displayed SDKs. These filters allow the user to explore the capabilities of the MCUs and Boards.
3. Once an SDK has been selected it can be installed (with options)
   - *Install and Create Project* will Download, Install and launch the New Project Wizard with the chosen board selected
   - *Install and Import Example* will Download, Install and launch the Import SDK Example Wizard with the chosen board selected
   - *Install* will Download and Install
   - *Uninstall* will remove the Plugin SDK from the IDE

Note: On rare occasions, it may be necessary to manually force a refresh of the cached contents of the remote repository – and this can be performed via the button highlighted above.

Once an SDK (or SDKs) is selected and an install operation begun, you will be presented with an option to accept the SDK license condition as below:

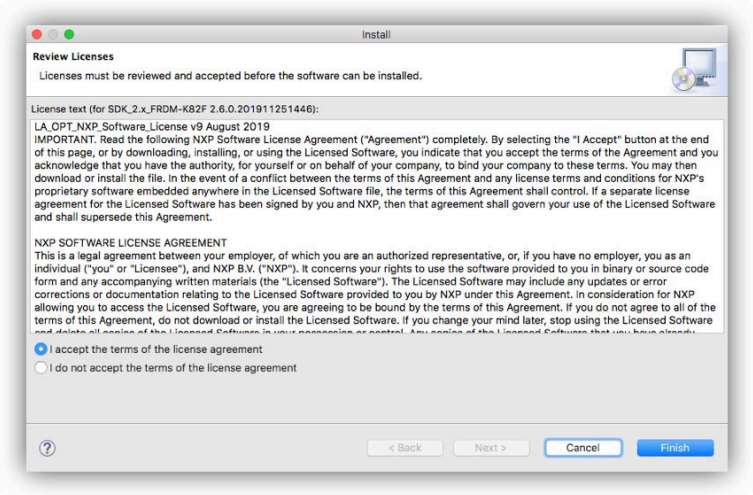


Figure 4.3. Plugin SDK Installation License

Download and install progress can be monitored via the Installation dialogue:

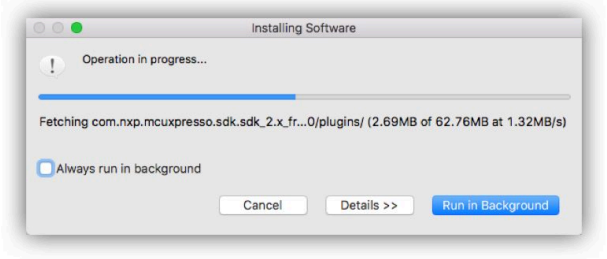


Figure 4.4. Plugin SDK Installation Progress

If *Run in Background* is clicked, control is returned to the IDE, of course the SDK will not become available until download and installation completes – at this time a Wizard may be launched if a *Create or Import* option was chosen. While the Plugin SDK Installer may be restarted, any existing SDK installations must complete before another Install can be started.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: When the IDE is started for the first time, data for this display is automatically loaded in the background. If the Plugin SDK Installer is started promptly after the IDE starts, there may be a short pause while the data populates.

4.2.2 SDK Part Support via SDK Builder

NXP also provides SDKs for toolchains (including MCUXpresso IDE) via their SDK Builder site. Through this site (login required), NXP MCU users may request builds for NXP MCUs that can be configured to include a range of software features. Once built, the SDK can be downloaded and installed into MCUXpresso IDE – this is the *Classic* method for installing SDKs as used in all previous versions of MCUXpresso IDE. SDKs installed in this way are now referred to as **FileSystem SDKs** since they become a shared resource for any IDE installation rather than part of a particular IDE installation.

These SDKs can be installed via a simple 'drag and drop' mechanism or from the dedicated dropdown menu in the Installed SDKs view, which will then automatically enhance the IDE with new part and board knowledge (and usually a large range of examples).

SDKs for MCUXpresso IDE can be generated and downloaded as required using the SDK Builder on the MCUXpresso Tools website at:

https://mcuxpresso.nxp.com/

Important Note: Only SDKs built specifically for **MCUXpresso IDE** are compatible with MCUXpresso IDE. **SDKs created for any other toolchain will not work!** Therefore, when generating an SDK be sure that MCUXpresso IDE is specified as the Toolchain.

4.2.3 Obtaining and Installing an SDK via SDK Builder

Users of earlier versions of the IDE may be more familiar with this model of SDK build and installation.

SDKs are installed and managed via the *Installed SDKs* view, which is located by default as the first tab within the Console view. See Major Components [18] item 3 for more information.

SDKs are free to download (login is required); MCUXpresso IDE offers a link to the SDK portal (shown below) from the Installed SDK Console view, which will open in an external browser. From this portal, required SDKs can be downloaded onto the host machine. Alternatively, you can open the portal by going to *Help -> Additional Resources -> MCUXpresso SDK Builder*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

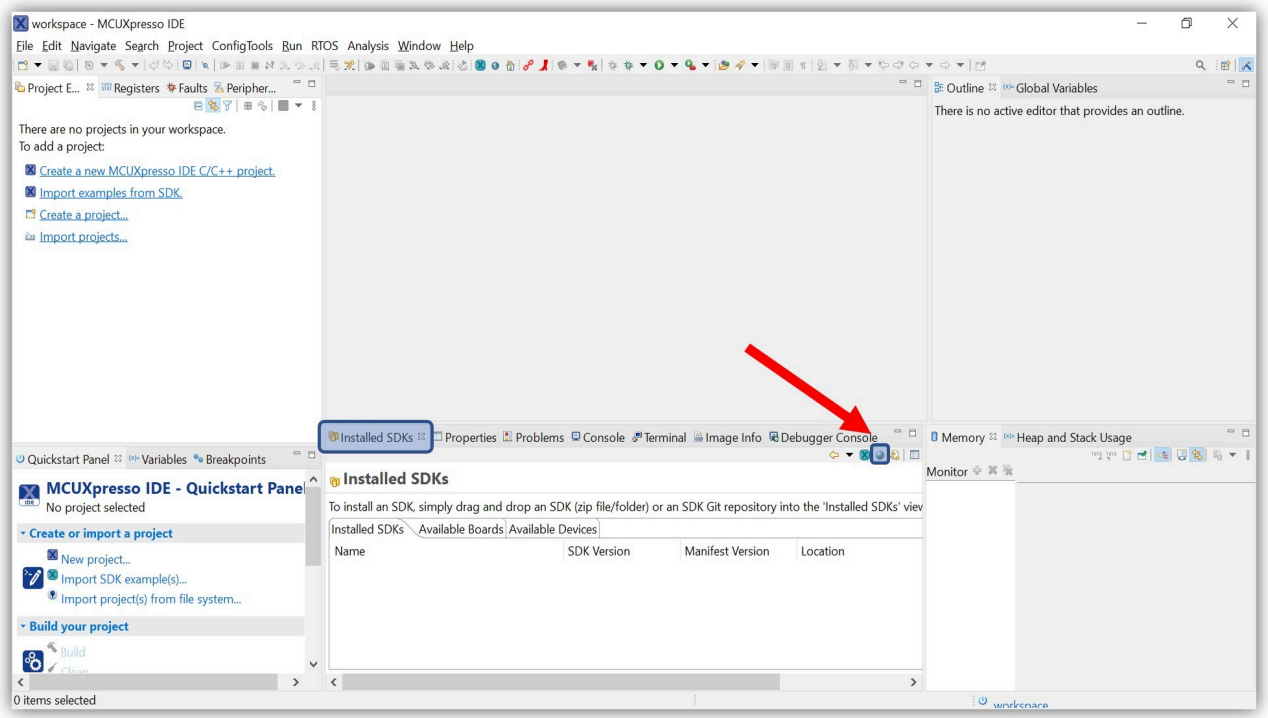


Figure 4.5. SDK Import

Once downloaded, an SDK package(s) can be installed by simply *dragging* from the downloaded location into the *Installed SDKs* view or by using the dedicated dropdown menu in the view. In case the *dragging* method is used, once *dropped*, you will be prompted with a dialog asking you to confirm the import – click OK. The SDK package(s) will then be automatically installed into MCUXpresso IDE part support repository.

Once complete the “Installed SDKs” view will update to show you the package(s) that you have just installed.

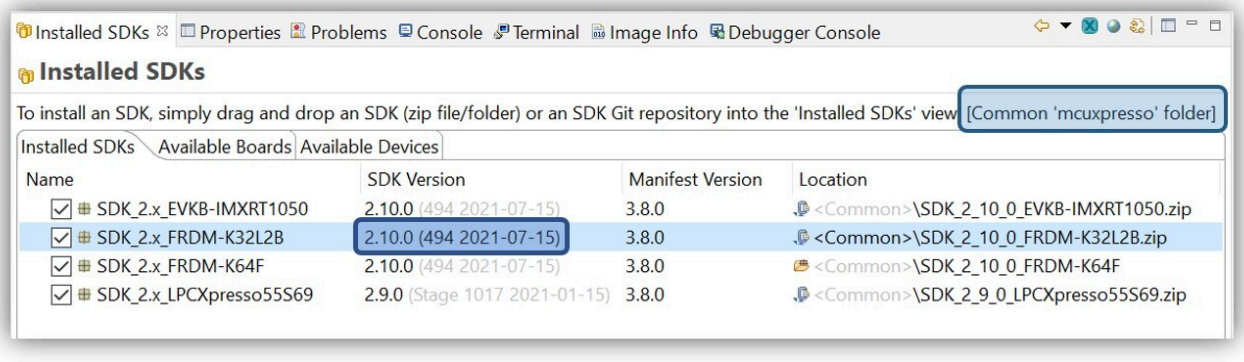


Figure 4.6. SDK Import View

By default, SDKs will install into a *Common* folder and will therefore be available to any MCUXpresso IDE instance. Alternatively SDKs can be installed into the current Workspace so making their installation local to that Workspace. The selected install location is shown in the SDK Window text as highlighted above. Also highlighted is the new version information string (displayed in grey), this feature allows different SDK builds to be distinguishable. Please also see SDK Advanced Importing [40] for further information on SDK installation options.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

SDK Notes:

- Released in parallel with MCUXpresso IDE version 11.8.0 are updated SDKs (MCUXpressoSDK v2.14.x). These are indicated by their version 2.14.x and a manifest version 3.13.0 in the Installed SDK view. While older SDKs are still compatible with the newest MCUXpresso IDE version, it is recommended that users check and update to the latest available SDK package.
  - Installed SDK view tooltips display comprehensive version information
- MCUXpresso IDE can import an SDK as a zipped package or unzipped folder (or zipped Plugin). Typically importing as a zipped package is expected.
  - The main consequence of leaving SDKs zipped is that you will not be able to create (or import projects) into a workspace with linked references back to the SDK source files.
- When an SDK is imported via drag and drop, required files are copied and the original file/folder is unaffected. The copied files are installed into a default location allowing imported SDKs to be shared among different IDE instances/installations and workspaces. Data from imported SDKs is populate wizards with available MCU and board information. In addition they are parsed to generate part support and make example projects and drivers available etc.
  - By default, SDKs (like workspaces) are located in user local storage, this means they will only be available to the user who performed the installation. Please also see SDK Advanced Importing [40] for details of how a shared location could be used if needed.
- Once installed the part support provided by the SDKs is regenerated. This regeneration is required because an MCUs part support may be specified (with different versions) within more than one SDK. On rare occasions, it may be necessary to force a regeneration of SDK part support, this can be done by clicking the *Recreate and Reload* button within the top right block inside the *Installed SDK* view, or by right clicking within the view and selecting *Recreate*.

4.2.4 Installing SDKs by Importing a Remote SDK Git Repository

NXP also provides SDKs via its MCU SDK Git repository. These SDKs can be installed automatically using the wizard from *Installed SDKs* view.

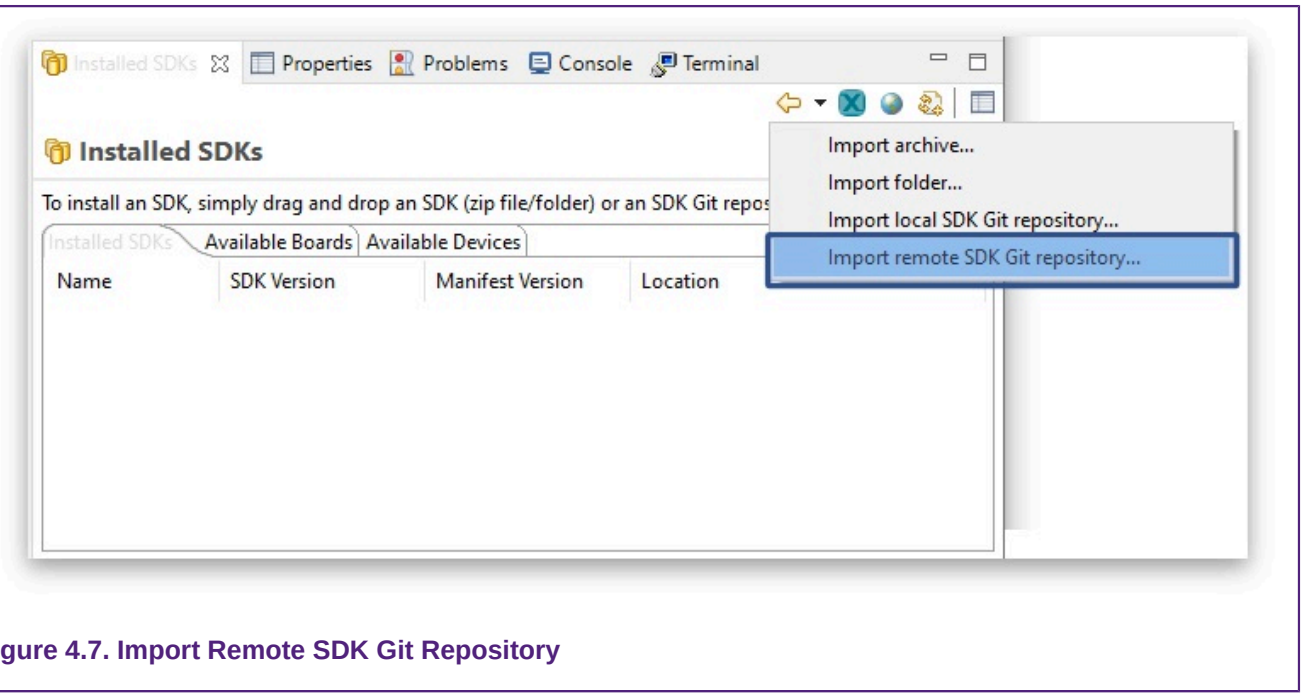


Figure 4.7. Import Remote SDK Git Repository

After selecting *Import Remote SDK Git Repository...* from the menu, the following window will be opened:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

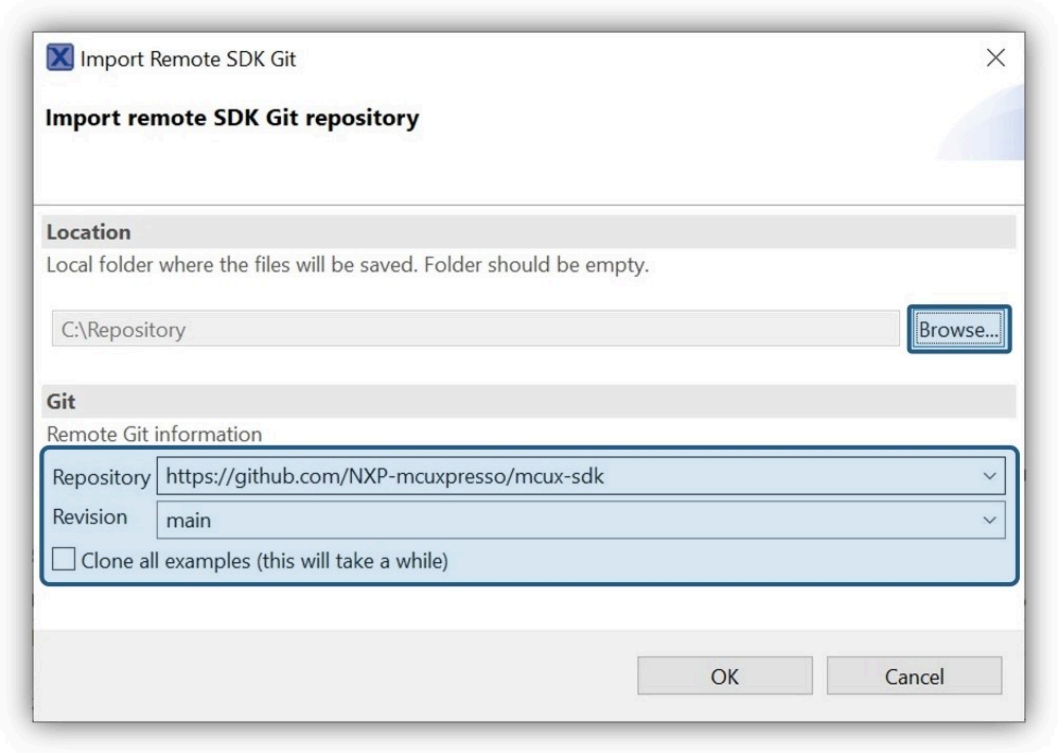


Figure 4.8. Import Remote SDK Git Repository Wizard

You'll have to select an empty folder where the SDK Git repository will be cloned. Revision can be "main" if you want the latest state, or any commit SHA.

Note: In order to speed up the remote import process, the default operation does not clone the example sources for every available board. Instead, they are downloaded on-demand whenever a specific example is selected for import. However, if you prefer a complete download from the start, you can select the *Clone all examples* checkbox. This option allows you to have all examples readily available, but keep in mind that this will lead to an increased download duration.

After clicking *OK* a window showing operation's progress will be opened.

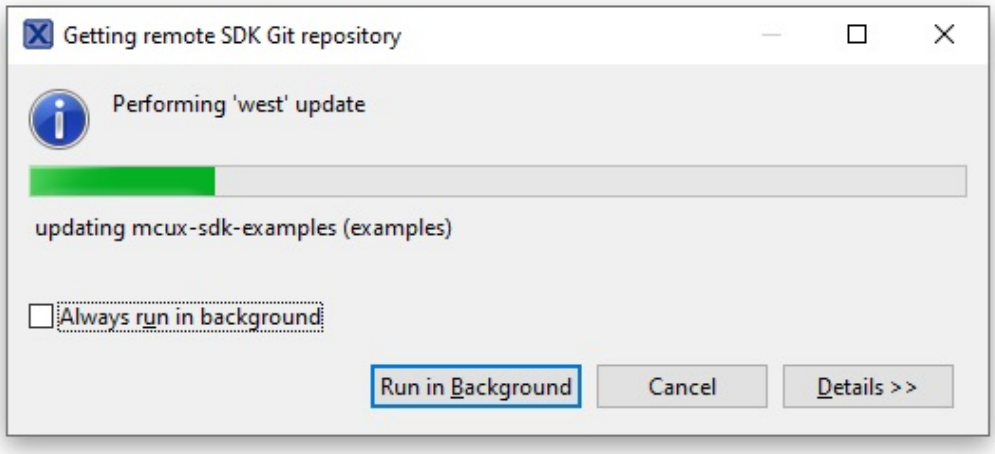


Figure 4.9. Import Progress

Please note that, depending on your Internet connection's speed, the operation may take a few minutes to complete. After cloning has completed, you will be prompted with a window providing

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

you with the option to import the repository. If you do not wish to change the manifest folder, pressing *OK* will import the repository with the default settings.

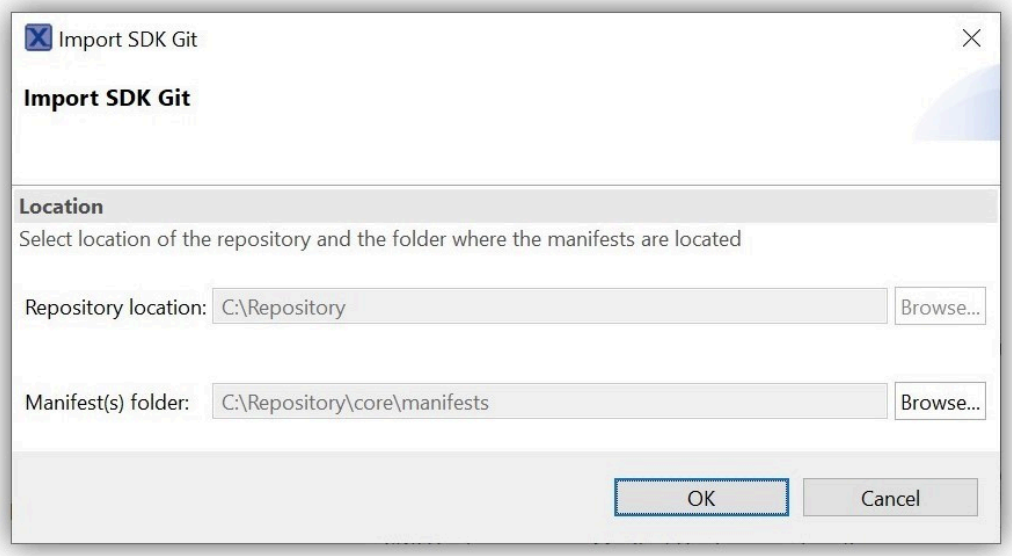


Figure 4.10. Importing Cloned SDK Git Repository

Afterwards, the SDKs will be imported and shown in the *Installed SDKs* view.

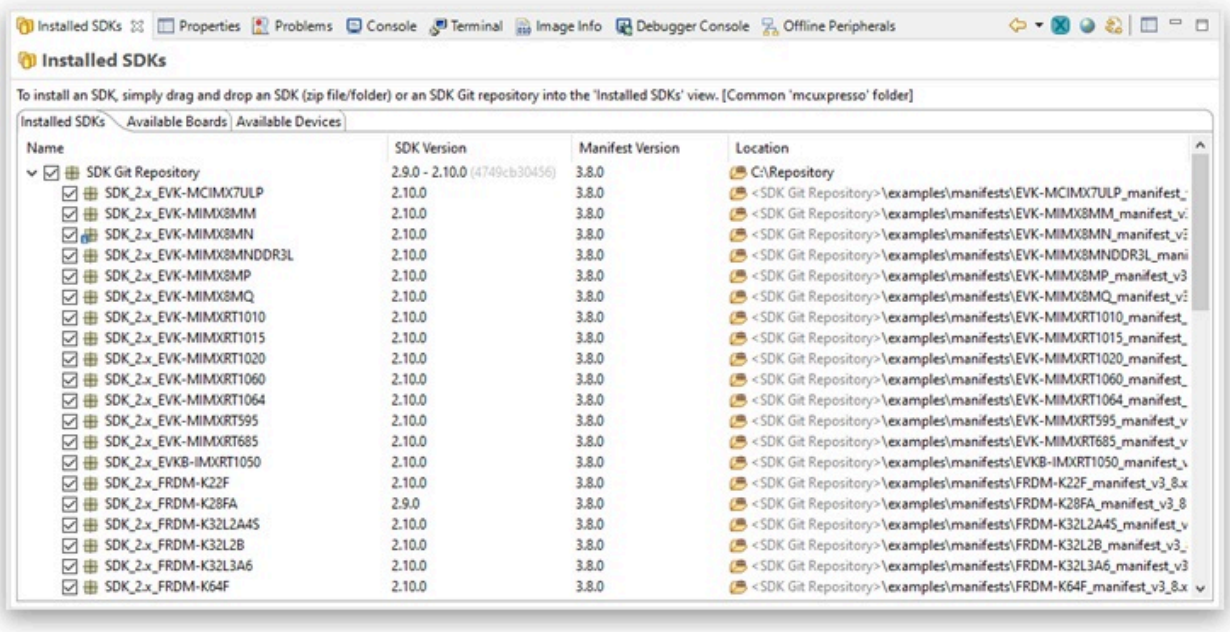


Figure 4.11. Installed SDK Git Repository

Important Note: You need to have both Git and West installed in order to use this wizard. If West is not installed or not found in the PATH environment variable, IDE will display the following warning:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

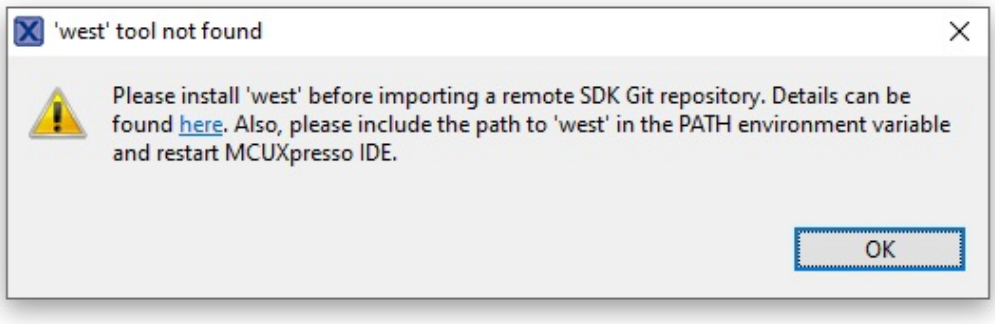


Figure 4.12. Import Remote Warning

On MacOS one way to add West to Path environment variable is to use “launchctl” command. Usage:

```
sudo launchctl config user path $PATH:{West absolute path}
```

4.2.5 Installing SDKs by Importing a Local Clone of an SDK Git Repository

If you used command line to obtain a local copy of the remote SDK Git repository, you can import it using *Import Local SDK Git Repository...* menu form *Installed SDKs* view.

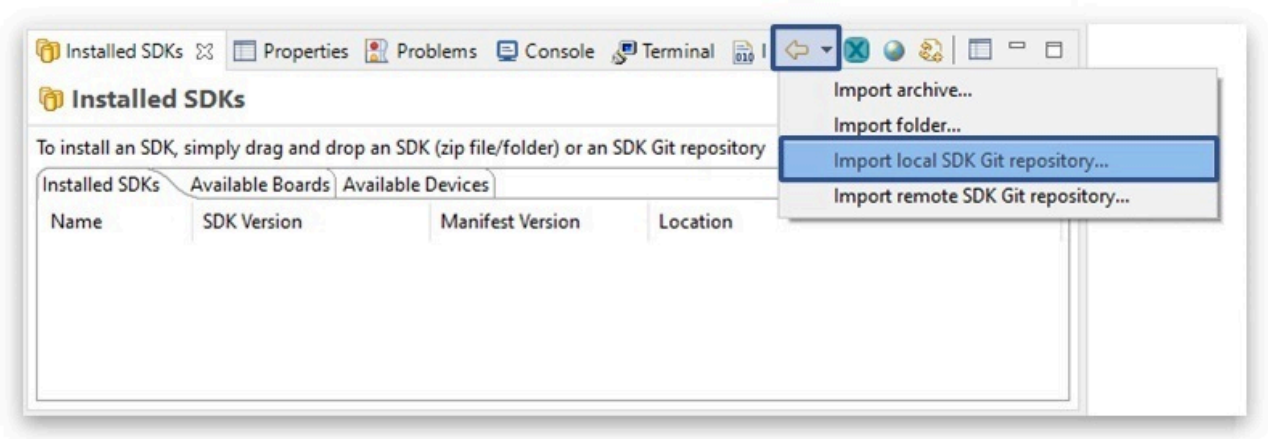


Figure 4.13. Import Local SDK Git Repository

4.2.6 Installed SDKs Operations

The installed SDKs view now incorporated 3 tabs. In addition to the *Installed SDKs* tab, new *Available Boards* and *Available Devices* tabs are provided. These tabs expose the supported boards and devices provided by the installed SDKs and allow the direct invocation of New Project and Example Import Wizards:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

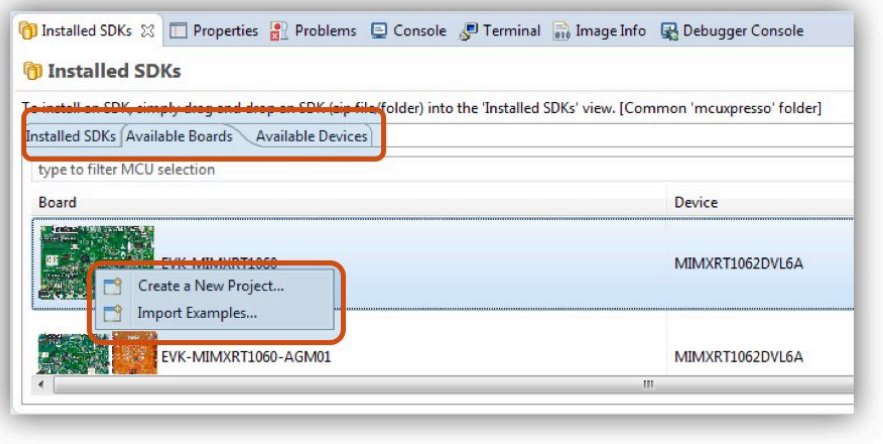
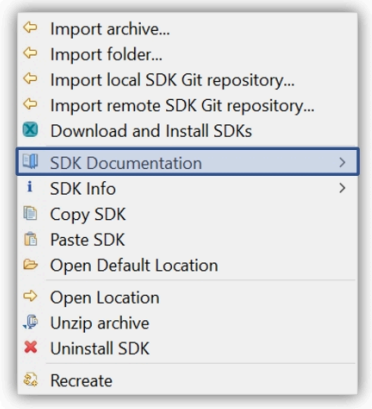


Figure 4.14. SDK Tabbed Views

Various other operations are available from the Installed SDKs view some from a right click menu options:

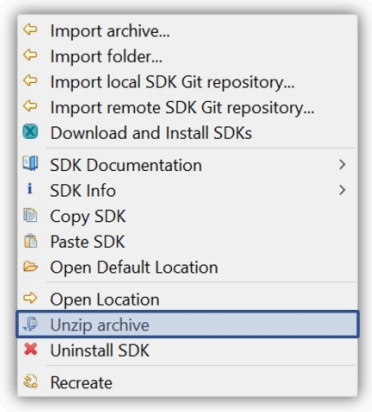


Important Note: Plugin SDKs cannot be unzipped from this view, however they can be converted to FileSystem SDKs [38]. Do not attempt to manually modify a Plugin SDK in any way, doing so could lead to a loss of SDK part support from the IDE. Plugin SDKs can be deleted either using the *Uninstall SDK* button from the Installed SDKs view, or using the *Uninstall* buton from the Install MCUXpresso SDKs view.

From here you can perform many actions such as view associated embedded SDK documentation that would otherwise require the unzipping and exploration of the SDK structure.

The Installed SDKs view will show whether the SDKs are stored as zipped archives or regular folders. MCUXpresso IDE offers the option to unzip a filesystem SDK archives in place via a right click option onto the selected SDK (as below).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Note: Unzipping an SDK may take some time and is generally not needed unless you wish to make use of referenced files or perform many example imports (where some speed improvement will be seen).

Once an SDK has been unzipped, its icon will be updated to reflect that it is now stored internally as a folder.

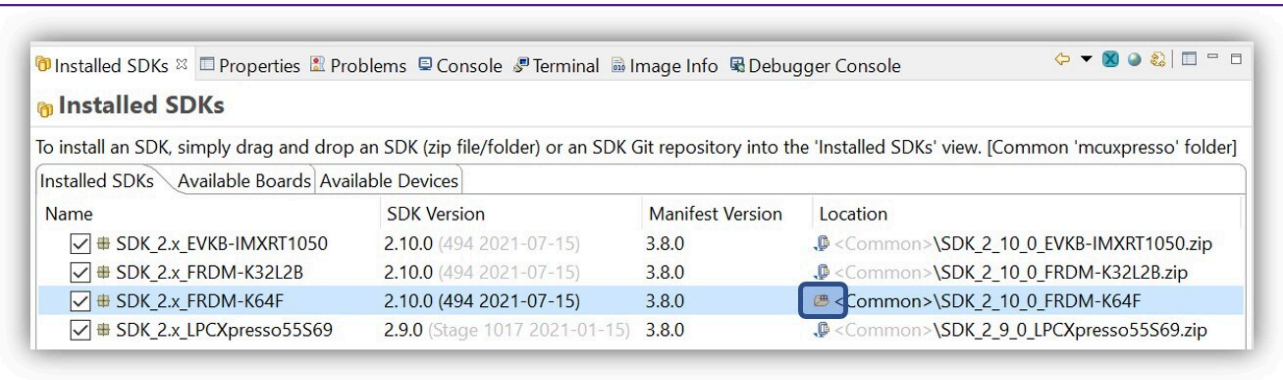


Figure 4.15. SDK Unzipped

Many other options are available such as examining SDK XML description files, Copying and Pasting SDKs and managing the library of installed SDKs.



Tip

To edit (and save) SDK XML files, the SDK must first be unzipped and the following preference changed: *Preferences -> MCUXpresso IDE -> SDK Handling -> Misc*, uncheck the read only mode option. Once saved, changes will become permanent for that SDK installation.



Tip

In addition to the other SDK options, an SDK can be pasted into the Installed SDK view from the file system or another IDE instance.

Finally, SDK part support automatically regenerates when a new SDK is installed. If a project is imported and the expected part support is not available then select *Recreate* from the right click menu option to force a recreation of the SDK part support.

Converting a Plugin SDK into a FileSystem SDK

On occasion it may be useful to migrate a Plugin SDK to become a FileSystem SDK – for example if you require the SDK to be unzipped or to be shared with other IDE installations. To do this

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

simply select the Plugin SDK within the Installed SDK view then from the right click menu select *Copy* followed by *Paste*. This will launch an Import operation and copy the SDK contents from the Plugin into the default SDK FileSystem location. This SDK will be used in preference to the Plugin version.

Note: A Plugin SDK is a part of an IDE installation and can only be deleted using the dedicated "Uninstall" buttons from Installed SDKs and Install MCUXpresso SDKs views.

Uninstalling (Deleting) an Installed SDK

Plugin SDKs become part of the IDE and so cannot simply be deleted from the filesystem. Always use *Uninstall* button from Install MCUXpresso SDKs view or *Uninstall SDK* button from the Installed SDKs view.

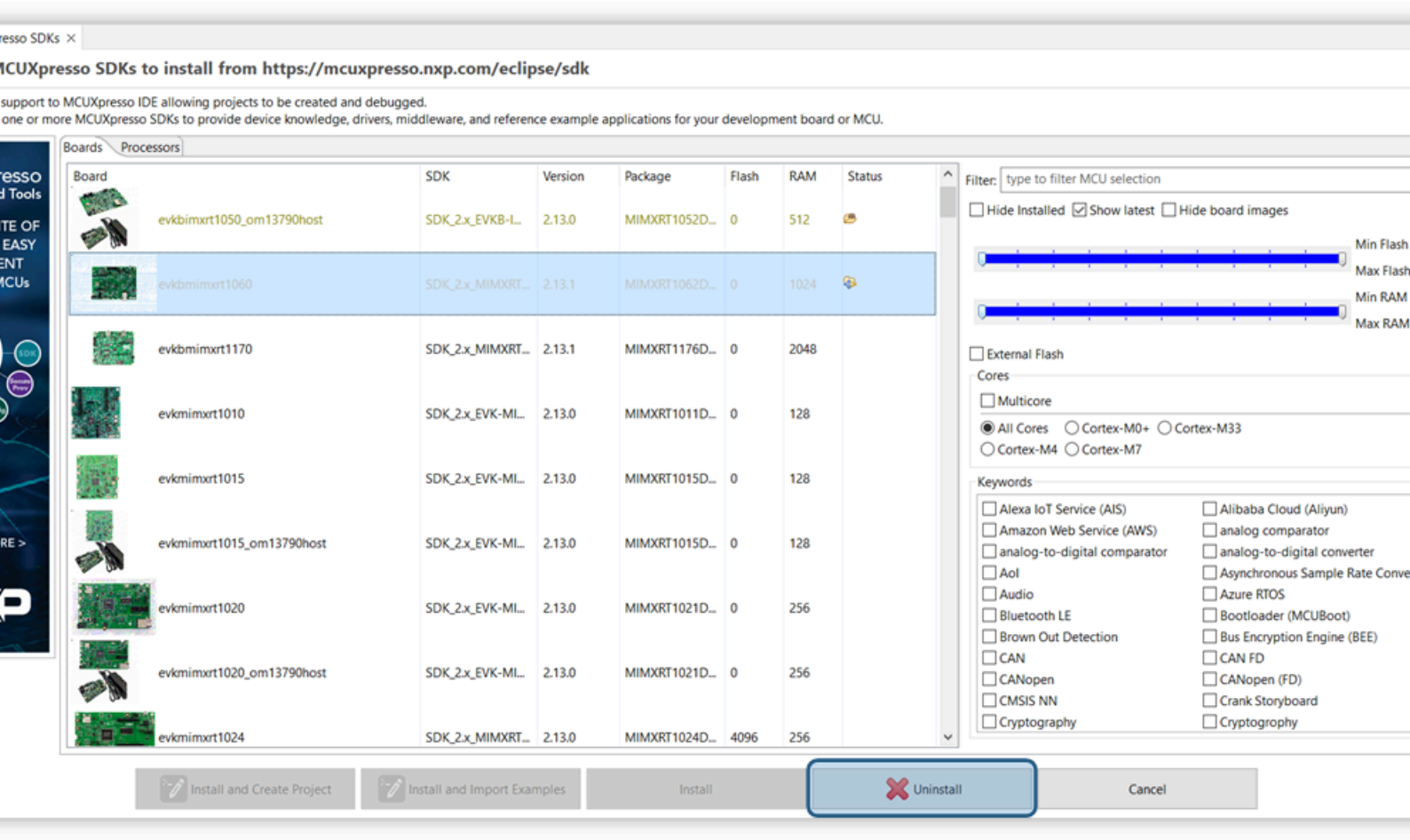


Figure 4.16. Plugin SDK Delete

Note: A FileSystem SDK will always be used in preferences to a Plugin SDK, so a Plugin SDK can effectively be replaced by the installation of a FileSystem SDK offering equivalent features.

If an SDK has been installed by the 'Drag and Drop' method, then a copy of the SDK will have been installed into the *Default Location*. SDKs installed into this location can be uninstalled and deleted via a right click option. Once an SDK has been uninstalled, part support will automatically be recreated for the remaining SDKs. Please see Uninstallation Considerations [45] for more information.

Alongside each installed SDK is a check box, if this is unchecked the SDK will be hidden from MCUXpresso IDE until re-checked. If multiple SDKs are installed that contain shared part

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

support, then this feature may be useful to force the selection of part support from a particular SDK. Please see Shared Part Support Handling [44] for more information.

SDKs installed into non default file system locations must be manually deleted or hidden if they are no longer required. **Note**: you may have to quit MCUXpresso IDE to delete these SDKs. Please see SDK Importing and Configuration [40] for more information.

SDKs installed from a Git repository can only be uninstalled by deleting the entire repository from the Installed SDKs view.

4.2.7 Installed SDKs Features

You can explore each of the SDKs within the Installed SDKs View to examine content such as Components, Memory Settings, included Examples etc.

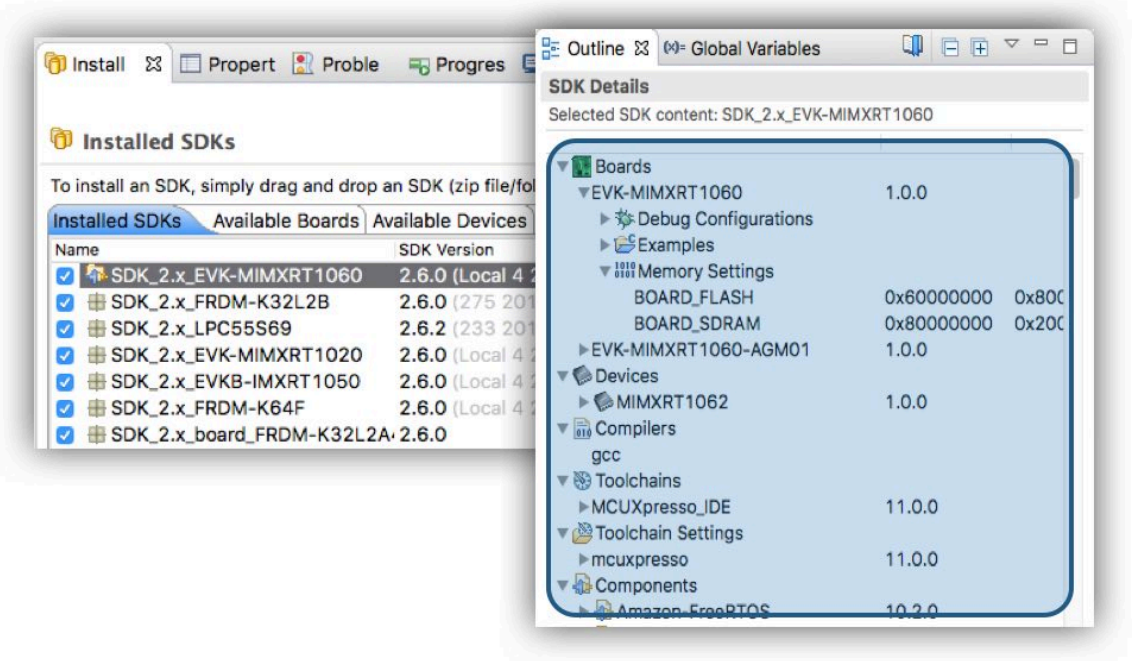


Figure 4.17. SDK Explore

4.2.8 Advanced Use: SDK Importing and Configuration

SDK importing via drag and drop incorporates two features. Firstly the location where the SDK will be copied, and secondly the automatic scanning of this location to create the required *Part Support*. The behaviour can be explored and changed via a preference *Preferences -> MCUXpresso IDE -> SDK Handling -> Installation* leading to the window below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

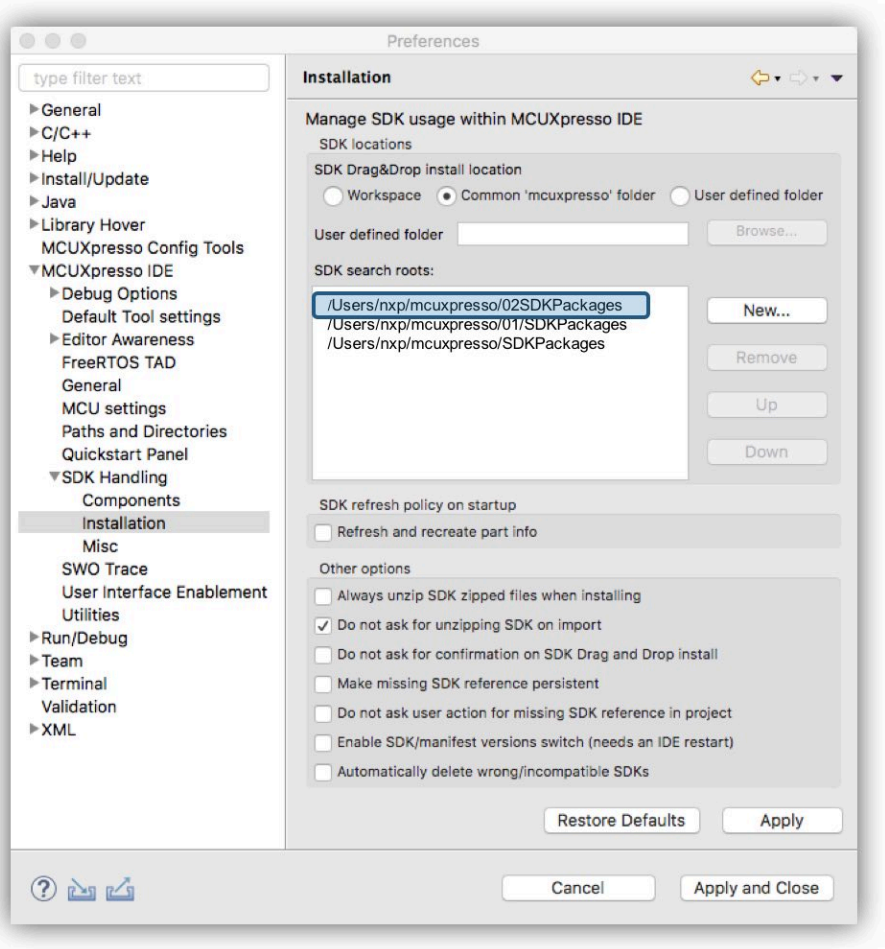


Figure 4.18. SDK Installation Preferences

You can see in the above graphic that two search locations are present. The *02* path is the default search path for MCUXpresso IDE version 11.0, earlier versions of MCUXpresso IDE used the *01* path. This older path will only appear if the location actually contains installed SDKs (typically installed via an earlier version of MCUXpresso IDE). The reason for these separate paths is to allow users to have both the latest and older versions of MCUXpresso IDE installed without presenting incompatible versions of SDK to older versions of the tools. Please see SDK compatibility with earlier versions of MCUXpresso IDE [44] for more information.

- Workspace
- Common (the default)
- User Defined

The default *Common* install location can be changed to either the currently selected Workspace or a User Defined location. Once this is done, a new SDK Search Root path will automatically be added to the search roots list.

Note: while other search roots can be removed if desired, the currently selected drag and drop location root cannot.

In addition, from this dialogue you can add new search paths to folders where you have stored or plan to store SDK folders/zips. Those SDKs will appear in the Installed SDKs View along with those from the default location when the Installed SDK view is refreshed.

The main differences between having SDKs in the default location(s) or leaving them in other folders are:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- “Delete SDK” function is disabled when using non-default locations
  - since these SDKs are not imported, they may be original files
- The knowledge of the SDKs and their part support is per-workspace

The order of the SDKs in the SDK location list may be important on occasion: if you have multiple SDKs for the same part in various locations, you can choose which to load by reordering. If multiple SDK are found, a warning is displayed into the Installed SDK view.

Note: Only the default SDK location(s) is persistent between workspaces. Any other locations must be created for each Workspace as required.

Also displayed in the dialogue (above) are a number of ‘checkbox’ options that are discussed below:

- Always Unzip SDK ... if checked, a zipped SDK will be unzipped on import.
- Do not ask for unzipping ... if checked (default) the IDE will not prompt the user to consider unzipping the SDK.
- Do not ask for confirmation ... if checked, the IDE will import and SDK via drag and drop without requesting user confirmation.
- Make missing SDK reference persistent ... this setting controls the persistence setting when the option below is checked.
- Do not ask for User action ... see shared part support [44] - if checked, this SDK association setting will be made without prompting the user.
- Enable SDK/manifest version... if multiple SDKs for the same part are installed, this option if checked also allows an older SDK to be selected from within the Installed SDK view via a drop down menu on the SDK Version
  - also, some SDKs include older versions of the manifest (XML description) ... if checked, this option allow an older manifest version to be selected from within the Installed SDK view via a drop down menu on the Manifest Version.
- Automatically uninstall ... if checked an SDK found in drag and drop install location that is incompatible with MCUXpresso IDE will be deleted.

4.2.9 Advanced Use: SDK Misc Options

Additional miscellaneous SDK preferences are also available. These checkbox options are shown below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

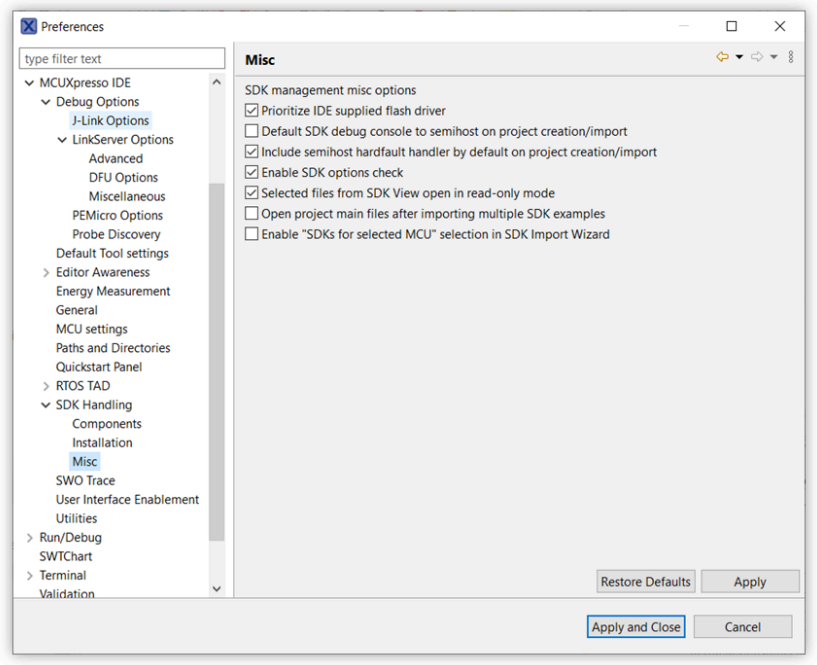


Figure 4.19. SDK Preferences Misc

Where:

- Prioritize IDE supplied flash drivers ... typically LinkServer flash drivers are supplied as part of the SDK part support for a particular MCU. However these LinkServer flash drivers are usually duplicated within the IDE installation where newer versions might be found. This option, checked by default causes the IDE supplied drivers to be used in preference to SDK supplied flash drivers. Searching the IDE's flash driver directory in preference to SDK dynamically part support files also simplifies flash driver development
- Default SDK debug console to semihost ... this option, checked by default, sets project defines to select semihosting as the output format
- Include semihost hardfault handler ... this option, checked by default, causes a minimal hardfault handler to be included within new and imported projects. The purpose of this handler is to send semihost operations to null when no debug tools are connected. Without such a handler, any semihosted operation will halt the MCU when no debug tools are connected. This is probably the most useful option for early project development, however this may clash with any *real* hardfault handler.
- Enable SDK options check ... this option, checked by default, allows the IDE to check an SDK examples options on import and attempt to resolve any incompatible options found.
- Selected files from SDK view ... this option, checked by default, forces any file opened from the Installed SDK view to be opened in Read Only mode. This is to protect SDK files from accidental corruption. Note: this option only applies to SDK that are imported unzipped.
- Open Project main files ... when an example project is imported it will be opened within the project explorer view and the source file containing the main function will be opened. This option, unchecked by default, will allow this to occur if multiple files are imported at the same time.

4.2.10 Important notes for SDK Users

Installing an SDK into MCUXpresso IDE adds to its default capabilities, but SDKs come in many different configurations, and versions. The section below discusses some of the issues that users may experience when working with SDKs.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Only SDKs created for MCUXpresso IDE can be used

If an error of the form *MCUXpresso IDE was unable to load one or more SDKs* is seen, the most likely reason is that the SDK was not built for MCUXpresso IDE. Within the SDK Builder, verify that the Toolchain is set to MCUXpresso IDE. If necessary, reset the toolchain to MCUXpresso IDE and rebuild the SDK.

SDK compatibility with earlier versions of MCUXpresso IDE

A new SDK version 2.14.0 has been released in parallel with MCUXpresso IDE version 11.8.0. However, this SDK format includes features that are not compatible with earlier versions of MCUXpresso IDE. As a result, these new SDKs may fail to install or offer reduced features when used in older versions of MCUXpresso IDE.

To support users who might have both this and also older versions of MCUXpresso IDE installed on their system, we have adopted a new default SDK installation location but also maintained support for the default used by older versions (now effectively Read Only from version 10.1.0 onwards).

The result of this is that MCUXpresso IDE version 10.1.0 and later will automatically inherit any SDKs installed into the (old) default location by previous versions of the IDE. While older versions of the IDE will not 'see' any SDKs installed with MCUXpresso IDE version 10.1.0 or later.

Note: If there is no need to maintain compatibility with older versions of the IDE, it is recommended that users migrate to using the latest SDKs where available.

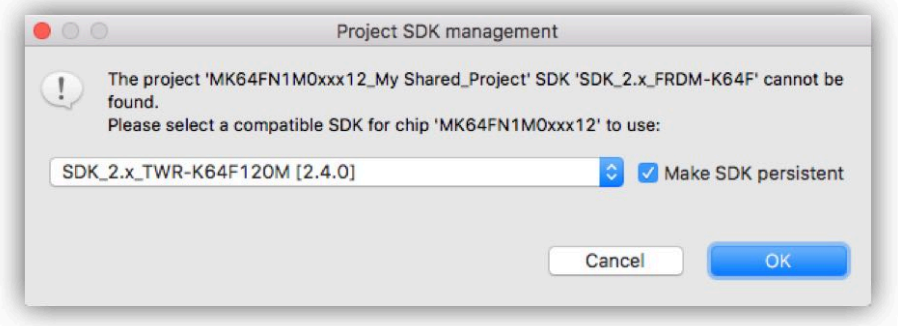
Shared Part Support Handling

Each SDK package will contain part support for one or more MCUs, therefore it is possible to have two (or more) SDK packages containing the same part support. For example, a user might request a Tower K64 SDK and later a Freedom K64 SDK that both target the same MK64FN1M0xxx12 MCU. If both SDKs are installed into the IDE, both sets of examples and board drivers will be available, but the IDE will select the most up to date version of part support specified within these SDKs. This means the various wizards and dialogues will only ever present a single instance of an MCU, but may offer a variety of compatible boards and examples. **Note**: If a board is selected (from one SDK) and part support is provided by another SDK, a message will be displayed within the project wizard to show this has occurred but no user action is required.

If two SDKs with matching part support are installed, and the SDK providing part support later deleted, then part support will automatically be used from the remaining SDK.

Finally, if a project created with one SDKs part support – for example Freedom K64, and then: - that SDK is changed to another SDK with compatible part support – for example TWR K64 - the project is shared with another user who has a different SDK that includes compatible part support (perhaps an SDK that has only device support).

A dialogue similar to the one below will be generated for each project where this occurs:



All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Where the option to *Make persistent* will permanently change the project to be associated with the selected SDK. If unticked, the IDE will accept the change as temporary and no data will be written back to the project.

Note: When this new association is made, the project will contain files from one SDK but be associated with another. If the project is refreshed or the component management feature is used, then incompatible code may be copied into the project.

Building a Fat SDK

An SDK can be generated for a selected part (processor type/MCU) or a board. If just a part is selected, then the generated SDK will contain both part support and also board support data for the closest matching development board.

Therefore, to obtain an SDK with both Freedom and Tower board support for say the Kinetis MK64... part, simply select the part and the board support will be added automatically.

If a part is chosen that has no directly matching board, say the Kinetis MK63… then the generated SDK will contain:

- part support for the requested part i.e. MK63...
- part support for the recommended closest matching part that has an associated development board i.e. MK64...
- board support packages for the above part i.e. Freedom and/or Tower MK64...

Uninstallation Considerations

MCUXpresso IDE allows SDKs to be installed and uninstalled as required (although for most users there is little benefit in uninstalling an SDK). However, since the SDK provides part support to the IDE, if an SDK is uninstalled, part support will also be removed. Any existing project built using part support from an uninstalled SDK will no longer build or debug. Such a situation can be remedied by re-installing the missing SDK. **Note**: if there is another SDK installed capable of providing the 'missing' part support, then this will automatically be used.

Sharing Projects

Note: Also see Enhanced Project Sharing Features [45] below:

If a project built using part support from an SDK and is then exported – for example to share the project with a colleague who also uses MCUXpresso IDE, then the colleague must also install an SDK providing part support for the projects MCU.

4.3 Enhanced Project Sharing Features

MCUXpresso IDE has a range of features designed to improve the ease of project sharing. These features combine to streamline the sharing and collaboration process.

4.3.1 Project Drag and Drop

In addition to the existing project import and export capabilities available from the **Quickstart** panel, a new set of features has been introduced to ease the transfer of projects.

Previously, the import of a project required browsing to a project location followed by an import. Now ...

- Projects can now be imported into a Workspace by simply dragging and dropping a folder (or zip) containing one or more projects into the Project Explorer view
- Projects can be copied from one IDE instance to another by simply dragging and dropping from one Project Explorer view to another

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Eclipse also offers the following functionality:

- Projects can also be exported by dragging from the Project Explorer view onto a host filer
  - **Warning**: Care must be used here since the default Eclipse behaviour when dragging is to **move** files from the workspace rather than performing a **copy**. This behaviour can be modified to **copy** on Mac via holding the Option Key, and on Windows via holding Ctrl. Note that **if** the underlying files of a project are moved, the project will remain visible within the project explorer view but will no longer be usable. A project explorer refresh (F5) should be performed.



Tip

If a project is moved accidentally (as described above), it can be re-imported by dragging back from the filer location into to the project explorer view (the original project must be removed first otherwise a clash of names will prevent import).

4.3.2 Project Local SDK Part Support

One weakness of the SDK model of extending the capabilities of the IDE comes when sharing projects with colleagues – since they must also have the same SDK installed to use this shared project.

To avoid this problem, SDK projects (and examples) can be modified to contain a local copy of the required SDK part support.

SDK project may be enhanced to contain local SDK part support

- SDK based projects can now import a cache of part knowledge from an installed SDK
  - Simply right click on a project and select *add SDK Part Support*

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

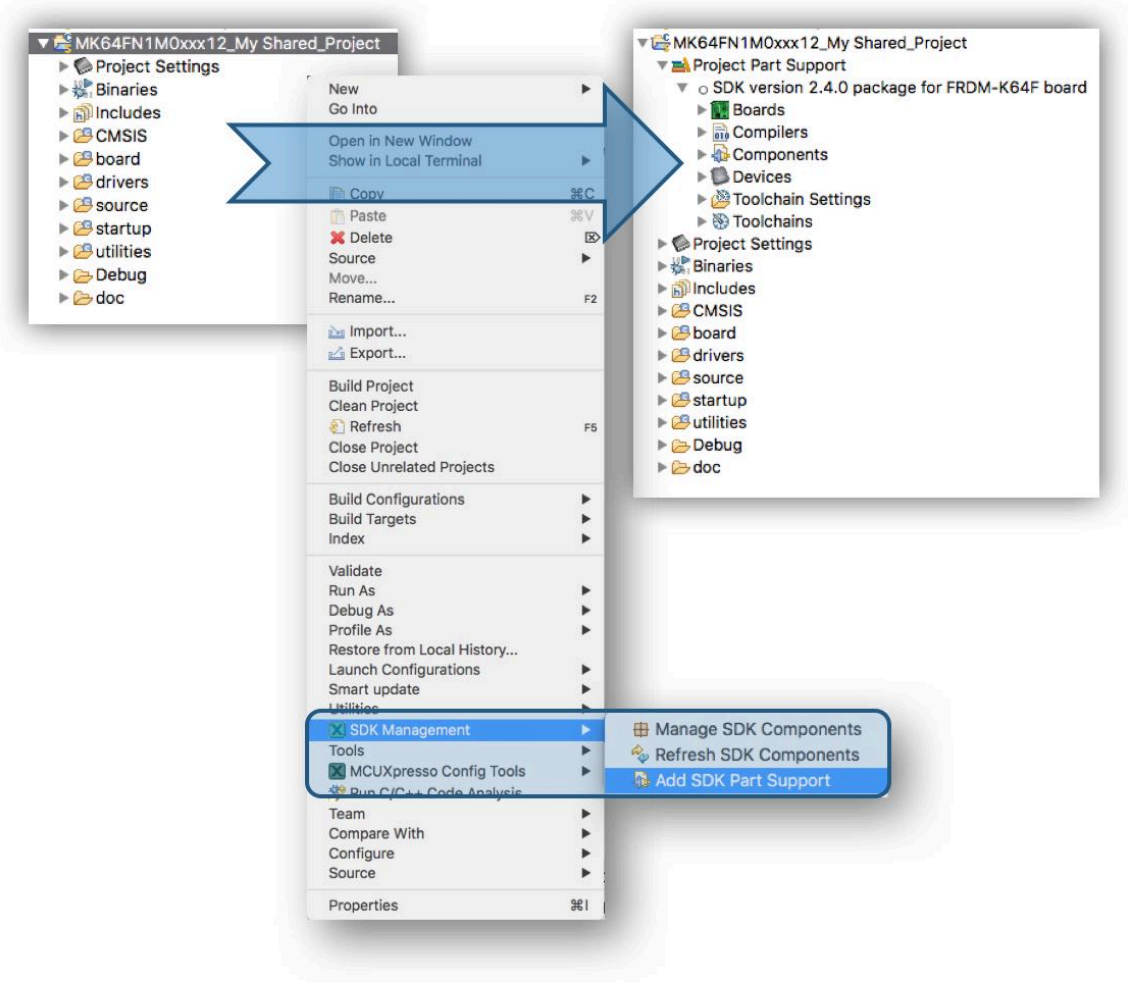


Figure 4.20. Add SDK Local Part Support

- Such projects can then be used (in other user's MCUXpresso IDEs version 10.2.0 or later) without first downloading and installing the appropriate SDK
  - In such cases, the project local part support will be visible as an installed SDK

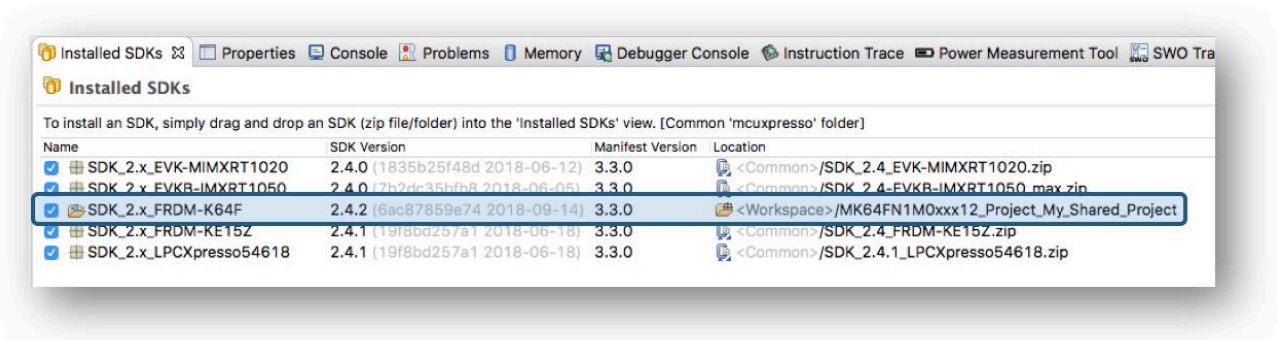


Figure 4.21. View SDK Local Part Support

Note: this feature is not designed to replace the need for ultimately installing an SDK, since there are implications in project size etc. rather it is intended as short term solution to decouple projects from the requirement for an SDK.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Finally, local part support can be removed in the same way as it was added. Simply right click on a project and select *SDK Management -> Remove SDK Part Support*. Once this has been done, an appropriate SDK must be installed for the project to be used.

4.3.3 Project Local Support files

Supporting files required for debug such as flash drivers, LinkServer Connect and Reset scripts will usually be found (automatically) either within an SDK or installed by default within the IDE.

However, on occasion, bespoke flashdrivers and/or scripts may be required. While these files could be stored and referenced from various locations within the file system, to enhance project sharing such files can now be included directly within a project and locally referenced.

To use script and flash driver files in this way, first they can simply be dragged into the local Project structure:



LinkServer launch configurations can now be used to directly browse to local scripts (connect or reset) as shown below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

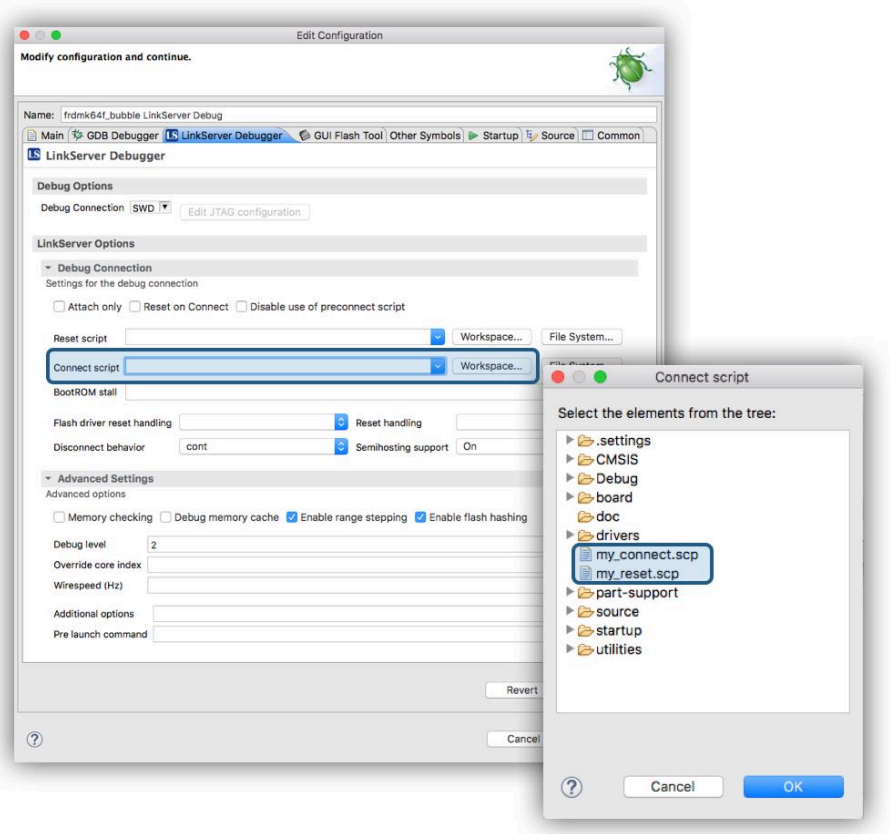


Figure 4.22. Local Script file

Similarly a project local flash driver can be referenced by editing a project's memory configuration and again browsing for the required flash driver within the project as below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

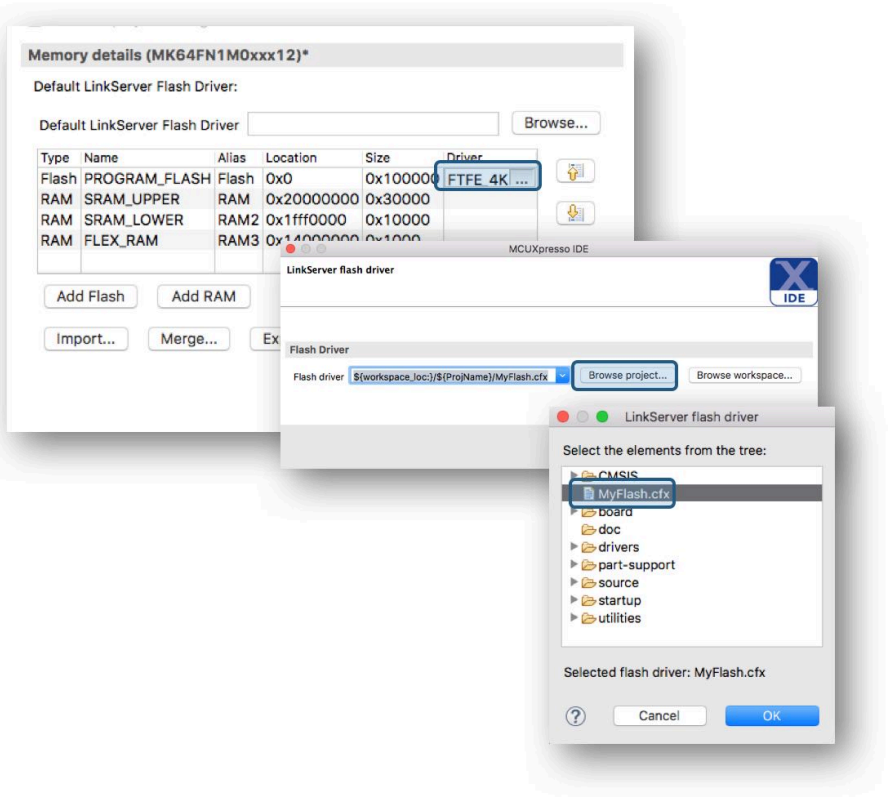


Figure 4.23. Local flash driver

See additionally Memory Configuration and Linker Scripts. [199]

The features described above will rarely be required, but on the occasions where shared projects have bespoke debug files, the above scheme should simplify the sharing and use of MCUXpresso IDE projects.

4.3.4 Export Project to Local SDK Git Repository

A project can now be exported into a local SDK Git Repository. The board and device for which the project was created have to be supported by the local SDK otherwise it can't be exported.

In order to export a certain project, right click on it inside the *Project Explorer* and select *SDK Management -> Export to SDK Git repository...*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

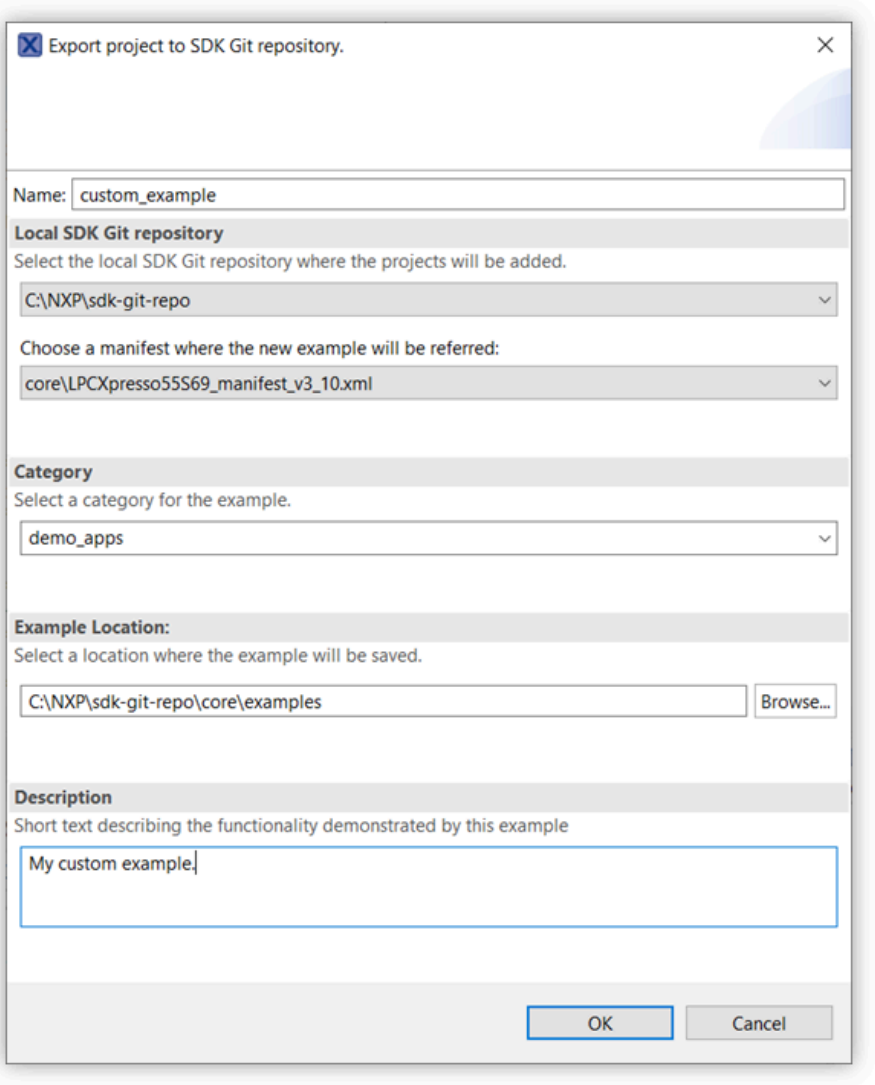


Figure 4.24. Export to Local SDK Git Repository

The name of the new example will be the name of the project, but it can be modified. For SDKs with split manifests, the user can also select the manifest where the project will be referred. A combo box presents all the categories available in the SDK for the current board. For the new example you can select one of the available categories or select a new one. The project location can also be chosen and it has to be in the same folder or in a subfolder of the manifest where the project will be exported. After clicking *OK* the example will be added in the SDK Git repository.

The new example will be available in the *SDK Import Wizard*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

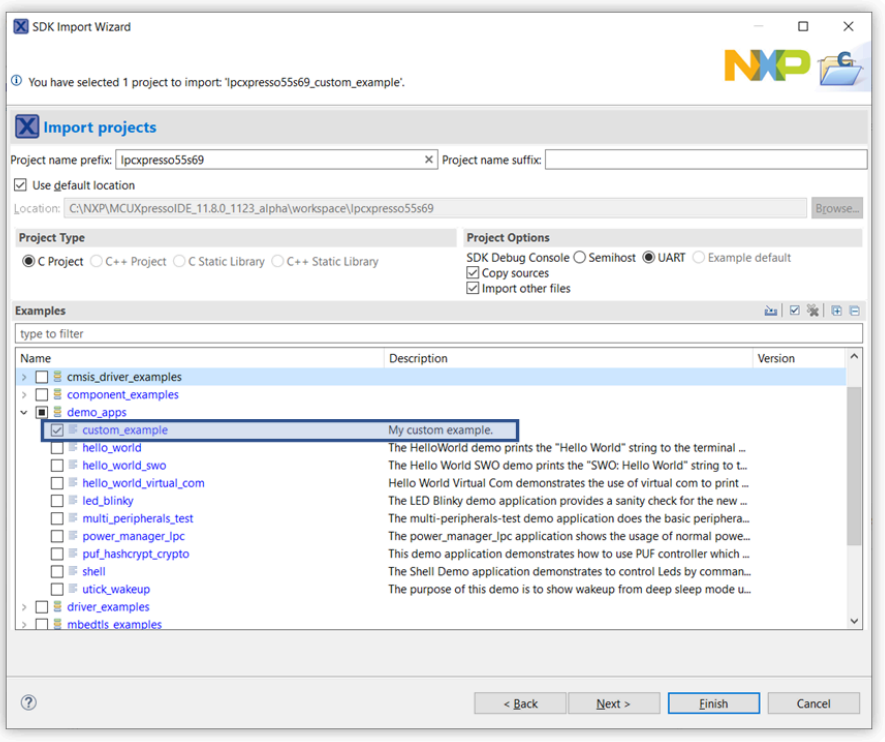


Figure 4.25. New Example Exported to Local SDK Git Repository

Limitations - Projects using linked references to the SDK source code are not supported. - Once exported to the local SDK Git repository, the new example can be imported only with “Copy Sources” set (source code files will be copied inside the project).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

5. Creating New Projects using installed SDK Part Support

For creating a project using *Preinstalled Part Support* please see: Creating Projects using Preinstalled Part Support [77]

Locate the Quickstart panel [21] at the bottom left of the MCUXpresso IDE perspective and see the first entry *New project*.

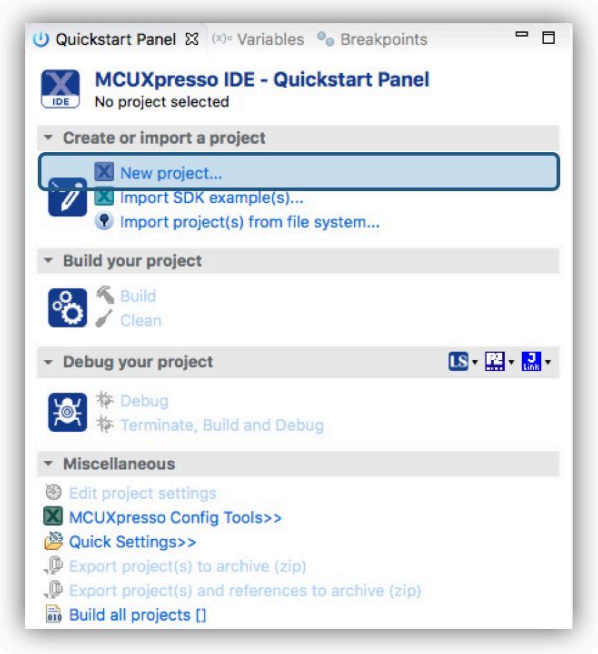


Figure 5.1. SDK Projects

The **New Project Wizard**, guides the user in creating new projects from the installed SDKs (and also from preinstalled part support – which will be discussed in a later chapter).

Click *New project* to launch the New Project Wizard as detailed below:

5.1 New Project Wizard

The New Project Wizard will begin by opening the “Board and/or device selection” page, this page is populated with a range of features described below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Figure 5.2. New Project Wizard first page

1. A display of all parts (MCUs) installed via SDKs. Click to select the MCU and filter the available matching boards. SDK part support can be hidden by clicking on the triangle (highlighted in the blue oval)
2. A display of all preinstalled parts (these are all LPC or Generic M parts). Click to select the MCU and filter the available matching boards (if any). Preinstalled part support can be hidden by clicking on the triangle (highlighted in blue)
3. A display of all boards from both SDKs or matching LPCOpen packages. Click to select the board and its associated MCU.
   - Boards from SDK packages will have **SDK** superimposed onto their image.
4. Some description relating to the user's selection
5. A display to show the matching SDK for a chosen MCU or Board. If more than one matching SDK is installed, the user can select the SDK to use from this list
6. Any Warning, Error or Information related to the current selection
7. An input field to filter the available boards e.g. enter '64' to see matching MK64… Freedom or Tower boards available
8. 3 options: to Sort boards from A-Z, Z-A or clear any filter made through the input field or a select click.



Tip

: Once a project has been created the selected board and/or MCU will be remembered and selected the next time the wizard is entered. To remove this selection, click the clear filter button (or any background white space)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

This page provides a number of ways of quickly selecting the target for the project that you want to create.

In this description, we are going to create a project for a Freedom MK64xxx board (The required SDK has already been imported).

First, to reduce the number of boards displayed, we can simply type ‘64’ into the filter (7). Now only boards with MCUs matching ‘64’ will be displayed.

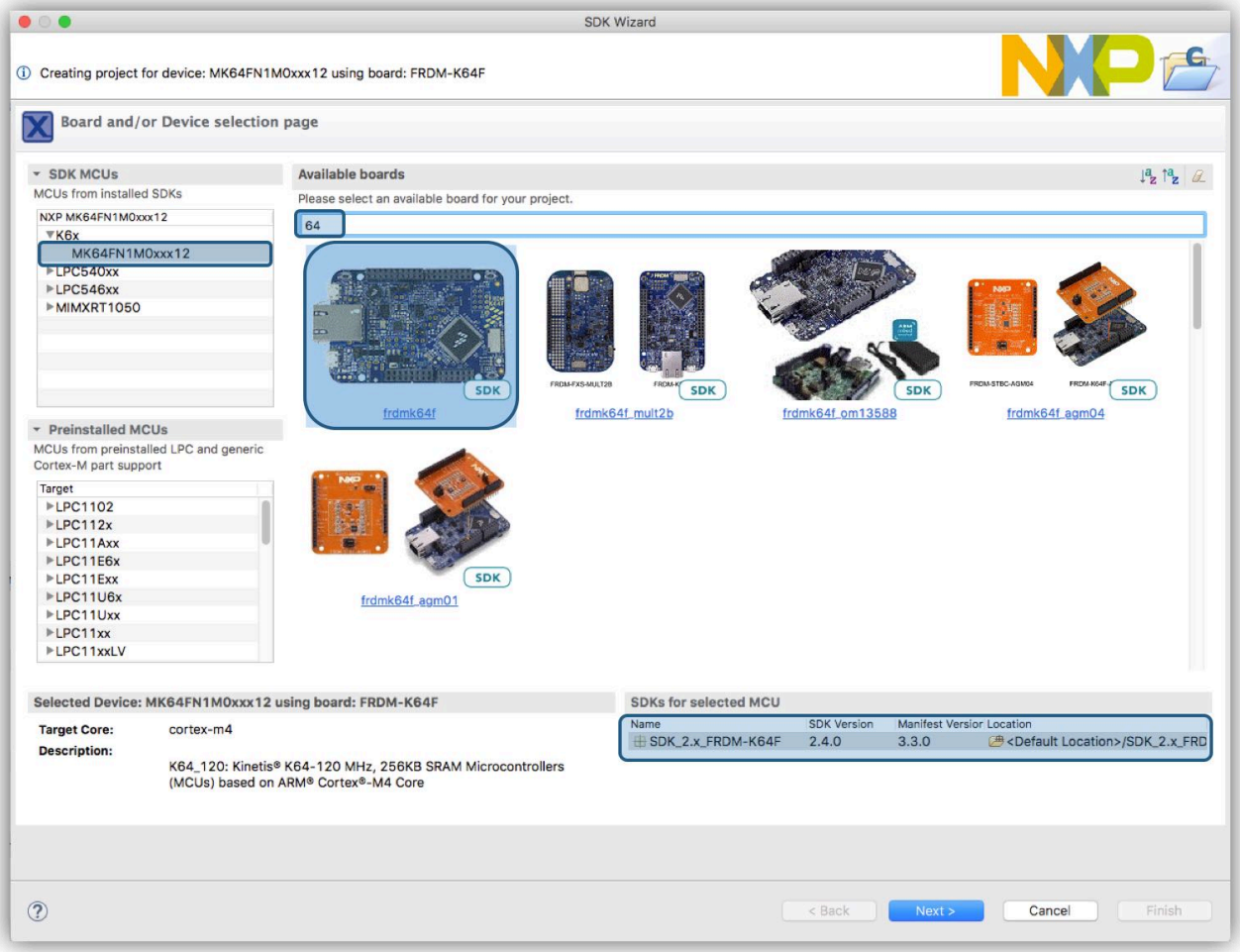


Figure 5.3. New Project Wizard selection

When the (SDK) board is selected, you can see highlighted in the above figure that the matching MCU (part) and SDK are also selected automatically.

With a chosen board selected, now click ‘Next’...

5.1.1 SDK New Project Wizard: Basic Project Creation and Settings

The SDK New Project Wizard consists of two pages offering basic and advanced configuration options. Each of these pages is pre-configured with default options (the default options offered on the advanced page may be set based on chosen selections from the basic page).

Therefore, to create a simple ‘Hello World’ C project for the Freedom MK64… board we selected, all that is required is simply click ‘Finish’.

Note: The project will be given a default name based on the MCU name. If this name matches a project within the workspace e.g. the wizard has previously been used to generate a project with

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

the default name, then the error field will show a name clash and the 'next' and 'finish' buttons will be 'greyed out'. To change the new project's name; the blank 'Project Name Suffix' field can be used to quickly create a unique name but retain the original prefix.

This will create a project in the chosen workspace taking all the default Wizard options for our board.

However, the wizard offers the flexibility to select/change many build, library and source code options. These options and the components of this first Wizard page are described below.

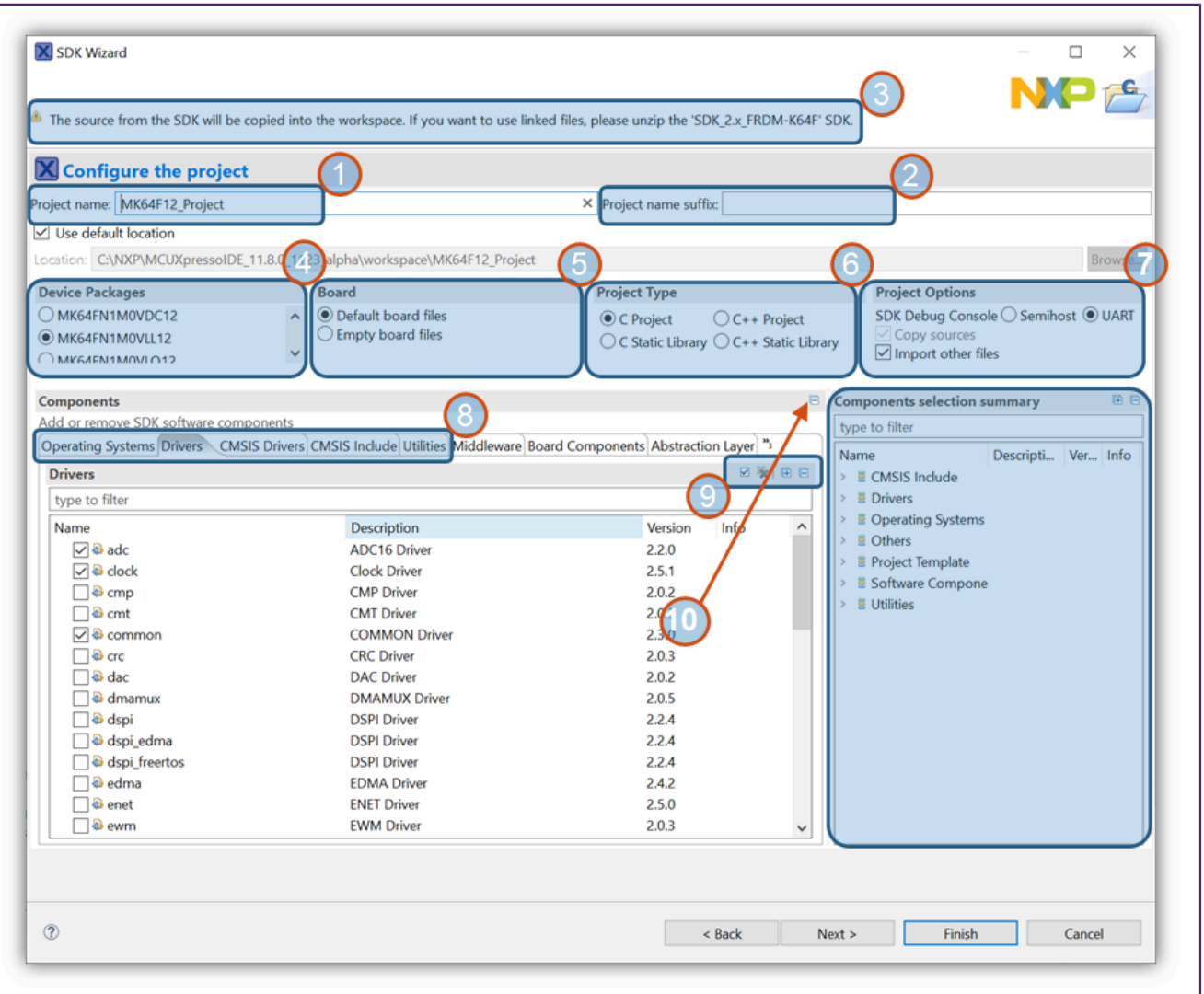


Figure 5.4. New Project Wizard basic SDK settings

1. Project Name: The default project name prefix is automatically selected based on the part selected on the previous screen
   - **Note**: Due to restrictions in the length of filenames accepted by the Windows version of the underlying GCC toolchain, it is recommended that the length of project names is kept to 56 characters or less. Otherwise you may see project build error messages regarding files not being found, particularly during the link step.
2. Project Suffix: An optional suffix to append to a project name can be entered here.
3. Errors and Warnings: Any error or warning will be displayed here. The 'Next' option will not be available until any error is handled. Errors may include such things as dependency problems or for example, a project name may have been selected that matches an existing project name in your workspace. The suffix field (2) allows a convenient way to create a unique project name.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

4. MCU Package: The device package can be selected from the range contained with the SDK. The package relates to the actual device packaging and typically has no meaning for project creation.
5. Board files: This field allows the automatic selection of a default set of board support files, else empty files will be created. If a part rather than a board had been selected on the previous screen, these options will not be displayed.
   - If you intend to use board specific features such as output over UART, you should ensure Default board files are selected.
6. Project Type: C or C++ projects or libraries can be selected. Selecting 'C' will automatically select RedLib libraries, selecting C++ will select NewlibNano libraries. See C/C++ Library Support [188]
7. Project Options:
   - Semihost: will cause the Semihosted variant of the chosen library to be selected. For C projects this will default to be Redlib Semihost-nf. Semihosting allows IO operations such as printf and scanf to be emulated by the debug environment.
   - UART: will cause the nohost variant of the chosen library to be selected. For C projects this will default to be Redlib Nohost. IO operations such as printf and scanf will occur via UART (or emulated UART provided by the debug probe over USB).
   - Copy Sources: For zipped SDKs, this option will be ticked and greyed out. For unzipped SDKs, projects can be created using linked references to the SDK sources.
8. Components:
   - OS: This provides the option to pull in and link against Operating System resources such as FreeRTOS.
   - Driver: enables the selection of supporting driver software components to support the MCU peripheral set.
   - CMSIS Drivers: code and headers for standard arm hardware
   - CMSIS Include: will cause a CMSIS folder containing a variety of support code such as Clock Setup, header files to be created. It is recommended to leave the associated options ticked.
   - Utilities: a range of optional supporting utilities.
     - For example select the debug_console to use SDK Debug Console handling of IO
     - Selecting this option will cause the wizard to substitute the (SDK) PRINTF() macro for C Library printf() within the generated code
       - The debug console option relies on the debug probe communicating to the host via VCOM over USB (LPC-Link2 and OpenSDA debug probes support this feature)
   - Middleware: enables the selection of various middleware components
   - Project Template: adds support files for the selected device/board
   - Depending on the SDK selected, additional options may also appear
9. Each set of components support a filter and check boxes for selection. These icons allow filters to be cleared, all check boxes to be set, all check boxes to be cleared.
10. Enable or Disable Component Selection Summary view: Since the selected software components cannot be viewed together, an optional summary view can be selected via this button. This setting will be remembered as a workspace preference.
11. Summary view of the software components selected for the project being created enabled as enabled via the above option.

Finally, if there is no error condition displayed, 'Finish' can be selected to finish the wizard, alternatively, select 'Next' to proceed to the Advanced options page (described next).

Important Note: Any components (OS, driver, utilities, middleware etc.) selected by default within this wizard will be available for use within the project. However the linker may remove the components supporting functions from the generated image if they are not referenced from within the user's project code. Additionally, selecting a component will automatically select any dependencies. Finally, please also note that this is an additive process, **removing components may leave unresolved dependencies resulting in a project that will not build**.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: Some middleware components are not currently compatible with the New project wizard functionality and so will be hidden. The recommended approach if such components are required is to import an example including the component and then modify this as required. Please see SDK Project Component Management [71] for details of how this might be done.

Note: By default, new project files are stored within the current MCUXpresso IDE workspace, **this is recommended since the workspace then contains both the sources and project descriptions**. However, the New Project Wizard allows a non default location to be specified if required. To ensure that each project's sources and local configuration are self contained when using non standard locations, the IDE will automatically create a sub directory inside the specified location using the *Project name prefix* setting. The newly created project files will then be stored within this location.

5.1.2 SDK New Project Wizard: Advanced Project Settings

The advanced configuration page will take certain default options based on settings from the first wizard project page, for example a C project will pre-select Redlib libraries, where as a C++ project will pre-select NewlibNano.

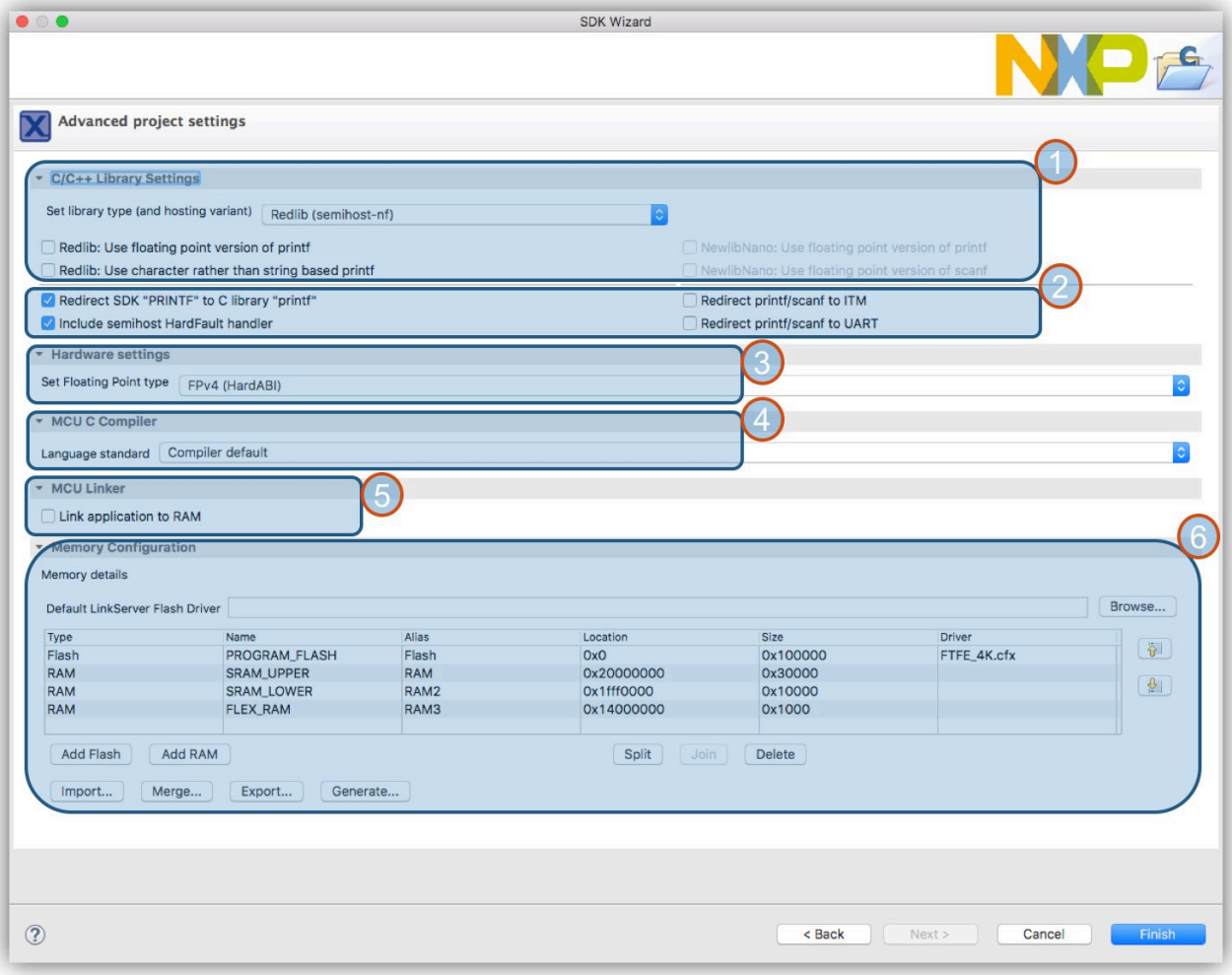


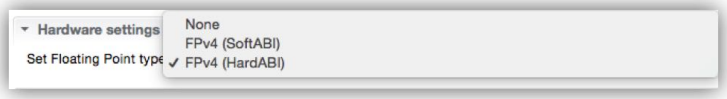
Figure 5.5. New Project Wizard advanced SDK settings

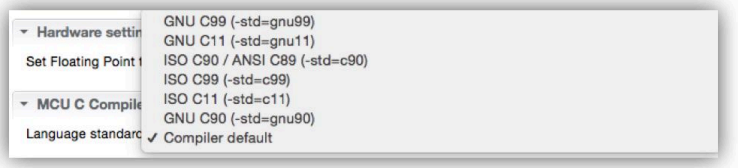
1. This panel allows the selection of library variants. See C/C++ Library Support [188]. **Note**: if a C++ project was selected on the previous page, then the Redlib options will be greyed out.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- 

  - Also, based on the selection, a number of options may be chosen to modify the capability (and size) of printf support
  - Redlib Floating Point printf: If this option is ticked, floating point support for printf will automatically be linked in. This will allow printf to support the printing out of floating point variables at the expense of larger library support code. Similarly for Newlib.
  - Redlib use Character printf: selecting this option will avoid heap usage and reduce code size but make printf operations slower.

2. This panel allows options to be set related to Input/Output. See C/C++ Library Support [188].
   - Redirect SDK “PRINTF”: many SDK examples use a PRINTF macro, selecting this option causes redirection to C library IO rather than options provided by the SDK debug console.
   - Include Semihost Hardfault Handler: selected by default, this option when checked will add a hardfault handler to the project sources. This handler is specifically written to deal with the situation that will occur if a semihosted function such as printf is executed when no debug tools are attached to support the operation. If this occurs, this handler will catch the operation and safely return to the executing application. Uncheck this option if you do not wish to use semihosted libraries or you intend to use your own hardfault handler. See semihosted printf [192] for more information.
   - Redirect printf/scanf to ITM: causes a C file 'retarget_itm.c to be pulled into your project. This then enables printf/scanf I/O to be sent over the SWO channel. The benefit of this is that I/O operations can be performed with little performance penalty. Furthermore, these routines do not require debugger support and for example could be used to generate logging that would effectively go to Null unless debug tools were attached. **Note**: This feature is not available on Cortex M0 and M0+ parts.
     - More information can be found in the MCUXpresso IDE SWO Trace Guide.
   - Redirect printf/scanf to UART: Sets the define SDK_DEBUGCONSOLE_UART causing the C libraries printf functions to re-direct to the SDKs debug console UART code.
3. Hardware Settings: from this drop down you can set options such as the type of floating point support available/required. This will default to an appropriate value for your MCU.
   - 

4. MCU C Compiler: from this drop down you can set various compiler options that can be set for the GNU C/C++ compiler.
   - 

5. Link Application to RAM checkbox reflects or sets the option to force the linker to ignore any defined flash regions and link the application to the first RAM region defined. This option is a copy of the flag at *Properties -> C/C++ Build -> Settings -> Managed Linker Script -> Link application to RAM* **Note**: This setting is only sensible for projects under development, since

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

debug control or a bootloader is required to load the code/data into RAM and simulate a processor reset.

6. Memory Configuration: This panel shows the Flash and RAM memory layout for the MCU project being created. The pre-selected LinkServer Flash driver is also shown. **Note**: this Flash driver will only be used for LinkServer (CMSIS-DAP) debug connections.
   - From this dialogue, the project's default memory setting may be edited in place if required and hence also the automatically generated linker scripts. See Memory Configuration and Linker Scripts [199]

5.2 Project Build

To build a project (created by the New Project Wizard), simply select the project in the 'Project Explorer' view, then go to the '**Quickstart**' Panel and click on the **build button** to build the project. This will build the active configuration of the selected project, where newly created projects default to the *Debug* configuration.

Note: MCUXpresso IDE projects are created with two build configurations, Debug and Release (more can be added if required). These differ in the default level of compiler optimization. Debug projects default to None (-O0), and Release projects default to (-Os). For more information on switching between build configurations, see How do I switch between Debug and Release builds? [262]

The build log will be displayed in the console view as below.

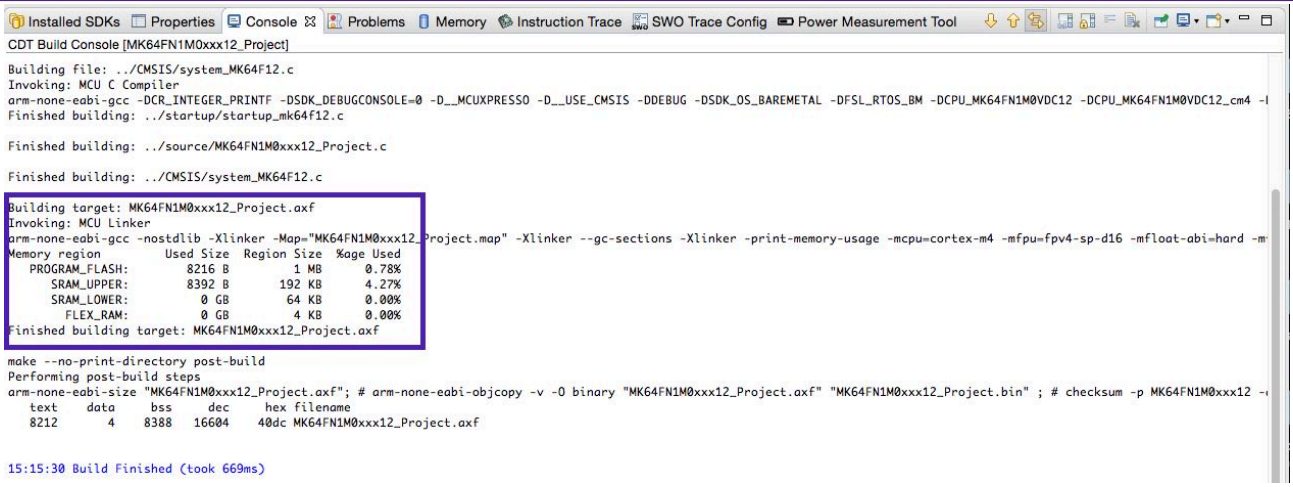


Figure 5.6. New Project Wizard Build

The project's memory usage as highlighted above is shown below:

```
Memory region    Used Size  Region Size %age Used

  PROGRAM_FLASH:    8216 B      1 MB      0.78%
  SRAM_UPPER:       8392 B    192 KB      4.27%
  SRAM_LOWER:         0 GB     64 KB      0.00%
  FLEX_RAM:           0 GB      4 KB      0.00%

Finished building target: MK64FN1M0xxx12_Project.axf
```

By default, the application will build and link against the first Flash memory found within the device's memory configuration. For most MCUs there will only be one Flash device available. In this case our project requires 8216 bytes of Flash memory storage, 0.78% of the available Flash storage.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

RAM will be used for global variable, the heap and the stack. MCUXpresso IDE provides a flexible scheme to reserve memory for Stack and Heap. The above example build has reserved 4KB each for the stack and the heap. Please See Memory Configuration and Linker Scripts [199] for detailed information.

Please also see Image Information [203] for details of how to explore the composition of an image in detail.

5.2.1 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.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

6. Importing Example Projects (from installed SDKs)

In addition to drivers and part support, SDKs also deliver many example projects for the target MCU.

To import examples from an installed SDK, go to the **Quickstart** panel and select **Import SDK example(s)**.

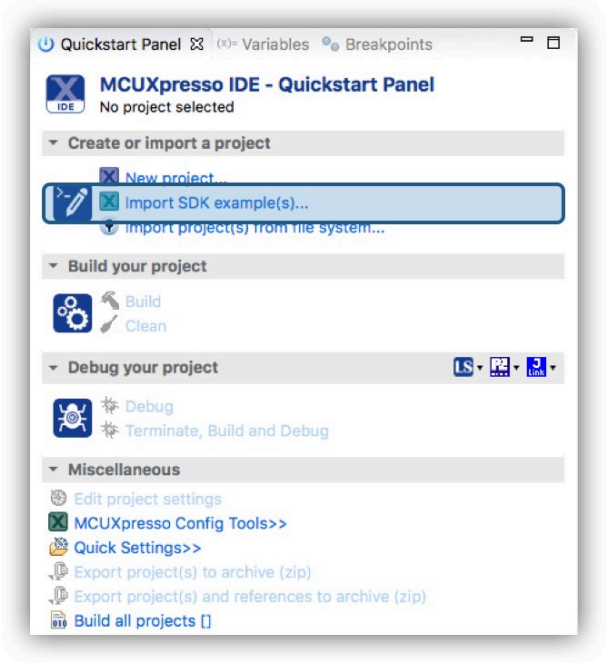


Figure 6.1. SDK Example

This option invokes the **Import SDK Example Wizard** that guides the user to import SDK example projects from installed SDKs.

Like the New Project wizard, this will initially launch a page allowing MCU/board selection. However now, only SDK supported parts and boards will be presented.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

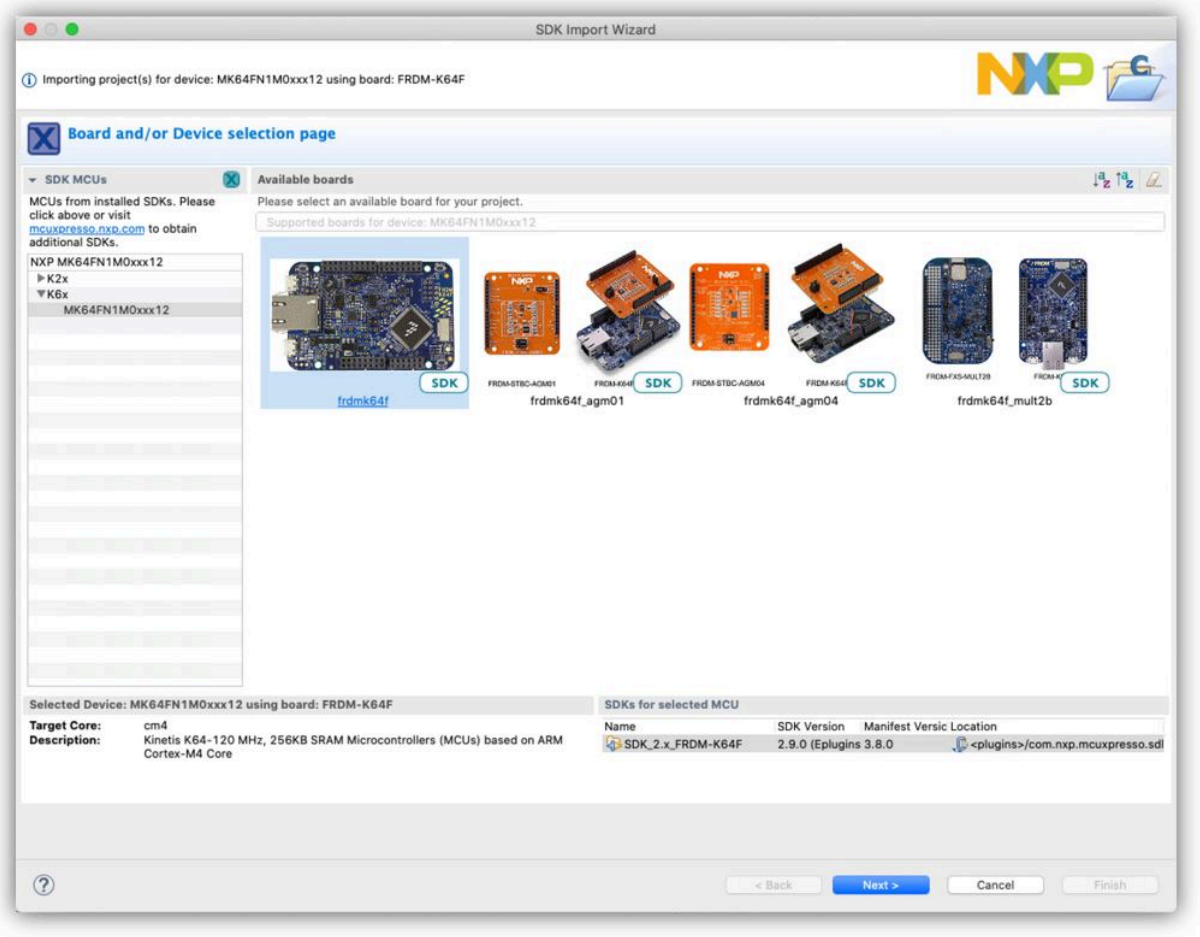


Figure 6.2. SDK Example Board

6.1 SDK Example Import Wizard

Selection and filtering work in the same way as for the New Project Wizard [53] but please be aware that examples are created for particular development boards, therefore a board must be selected to move to the ‘Next’ page of the wizard.

In the case of " *SDK Example Import Wizard*", the " *SDKs for selected MCU*" control is disabled and the selection of the proper SDK source for a specific chosen board/device is automatically performed. This is done in order to prevent unwanted selection of a SDK which can lead on getting files from a wrong source. This situation can occur when the same device is presented in multiple SDKs.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

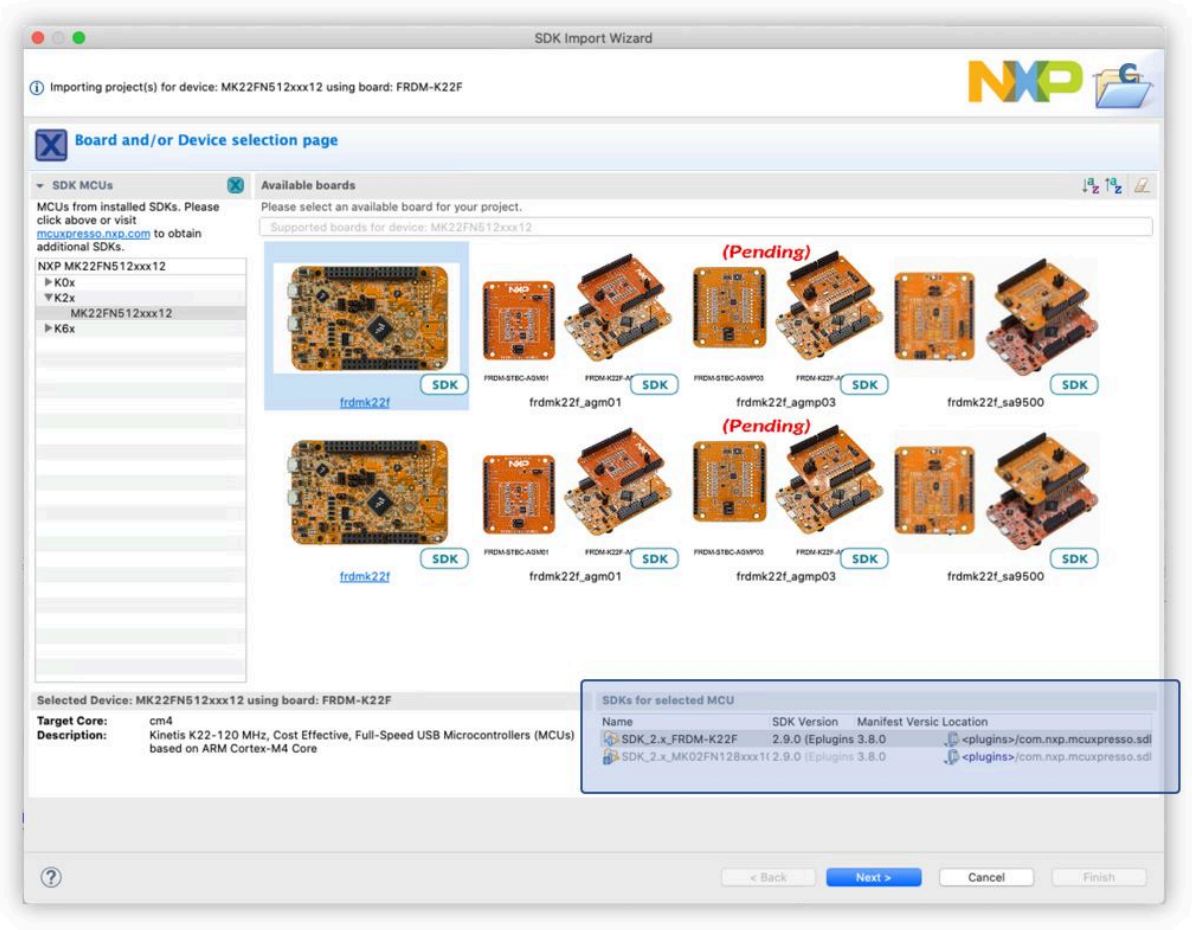


Figure 6.3. SDK Importer Multiple SDKs

Note: Even if not recommended, if it is absolutely necessary, the user can still force the option. This hard-coded selection can be removed by selecting the option from *Preference -> MCUXpresso IDE -> SDK Handling -> Misc -> Enable "SDKs for selected MCU" selection in SDK Import Wizard*. Then, a different SDK can be selected.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

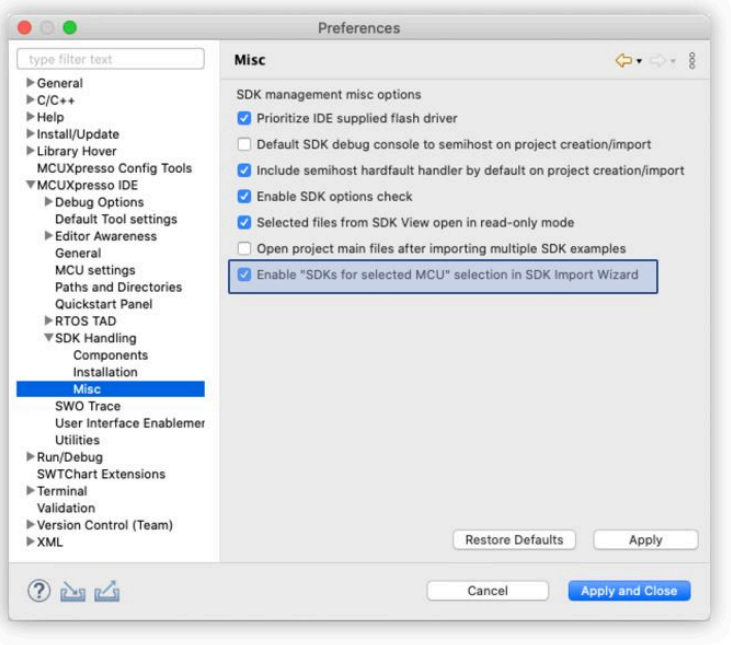


Figure 6.4. SDK Importer Force Manual SDK Selection

6.1.1 SDK Example Import Wizard: Basic Selection

The SDK Example Import Wizard consists of two pages offering basic and advanced configuration and selection options. The second configuration page is only available when a single example is selected for import. This is because examples may set specific options, and therefore changing settings globally is not sensible.

The first page offers all the available examples in various categories. These can be expanded to view the underlying hierarchical structure. The various settings and options are explained below:
Note: The project will be given a default name based on the MCU name, Board name and Example name. If this name matches a project within the workspace e.g. the wizard has previously been used to generate an example with the default name, then the error field will show a name clash and the ‘next’ and ‘finish’ buttons will be greyed out. To change the new example name, the blank ‘Project Name Suffix’ field can be used to quickly create a unique name but retain the original prefix e.g. add ‘1’.

MCUXpresso IDE will create a project with common default settings for your chosen MCU and board. However, the wizard offers the flexibility to select/change many build, library and source code options. These options and the components of this first Wizard page are described below.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

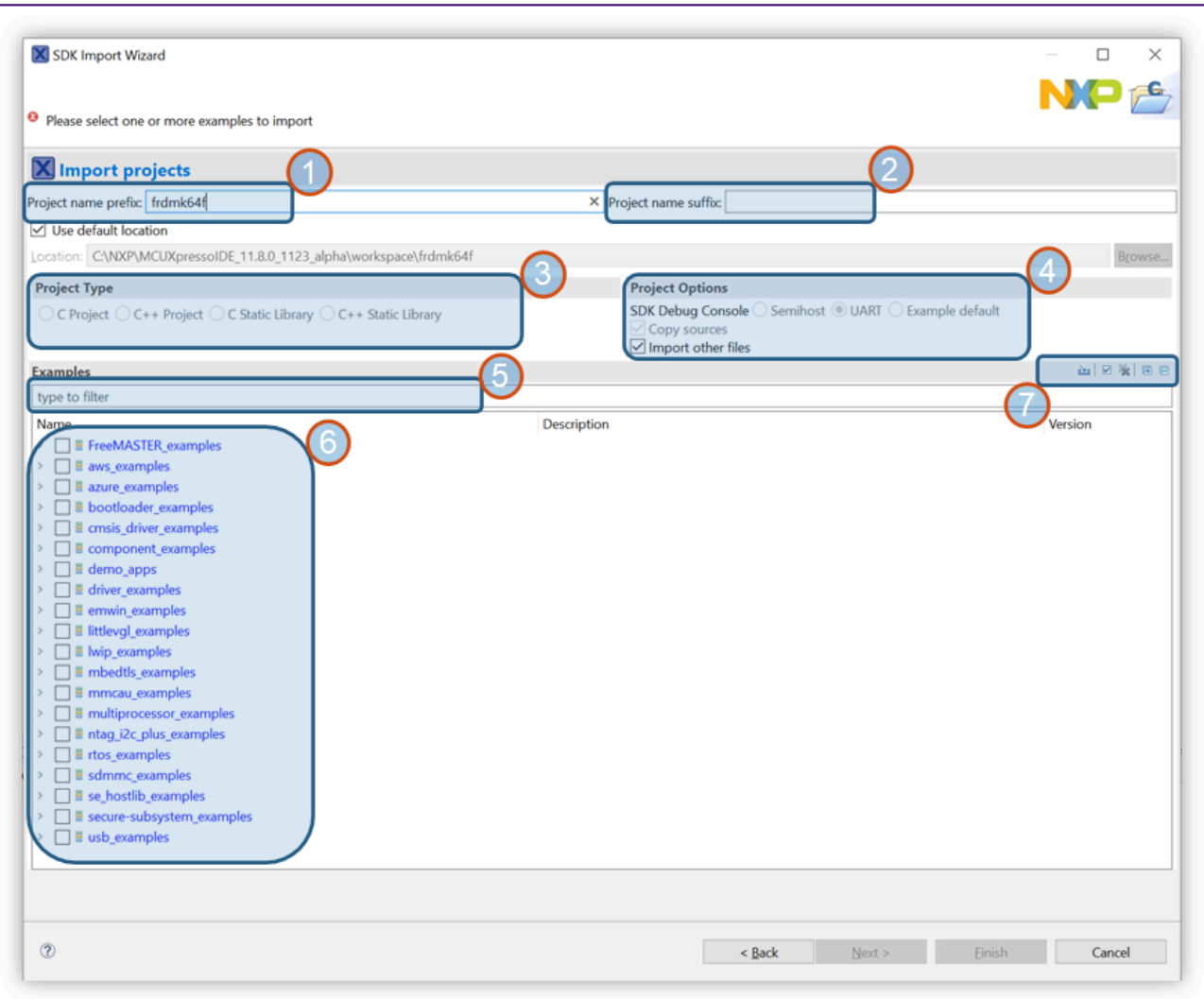


Figure 6.5. SDK Example Selection

1. Project Name: A project name is automatically created with the form: *boardname_examplename*
2. Project Suffix: An optional suffix to append to a project name can be entered here. This is particularly useful if you are repeating an import of one or more projects since an entry here can make all auto generated names unique for the current workspace...
3. Project Type: These will be set by the pre-set type of the example being imported. If more than one example is imported, then these options will appear greyed out.
4. Project Options:
   - 'SDK Debug Console': Once an example(s) has been selected, this option can be used to control IO between semihost console, UART or the examples default setting.
   - 'Copy sources': For unzipped SDKs, you can untick this option to create project containing source links to the original SDK files. This option should only be unticked with care, since editing linked example source will overwrite the original files!
   - 'Import other files': By default, non source files such as graphics are filtered out during import, check this box to import all files.
5. Examples Filter: Enter text into this field to find possible matches, for example enter 'LED' or 'bubble' to find examples present in many SDKs. This filter is case insensitive.
6. Examples: The example list broken into categories. **Note**: for some parts there will be many potential examples to import
7. Various options (from left to right):

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Opens a filer window to allow an example to be imported from an XML description. This is intended as a developer feature and is described in more detail below.
- Clear any existing filter
- Select (tick) all Examples
- Clear all ticked examples
- Open the example structure
- Close the example structure

Finally, if there is no error condition displayed, 'Finish' can be selected to finish the wizard, alternatively if only one example has been selected the option to select 'Next' to proceed to the Advanced options page is available (described in the next section).

Note: SDKs may contain many examples, 263 is indicated for the FRDM MK64 SDK example shown below. Importing many examples will take time ... Consider that each example may consist of many files and associated description XML. A single example import may only take a few seconds, but this time is repeated for each additional example. Furthermore, the operation of the IDE maybe impacted by a large number of projects in a single workspace, therefore it is suggested that example imports be limited to sensible numbers.

Note: Due to restrictions in the length of filenames accepted by the Windows version of the underlying GCC toolchain, it is recommended that the length of project names is kept to 56 characters or less. Otherwise you may see project build error messages regarding files not being found, particularly during the link step.

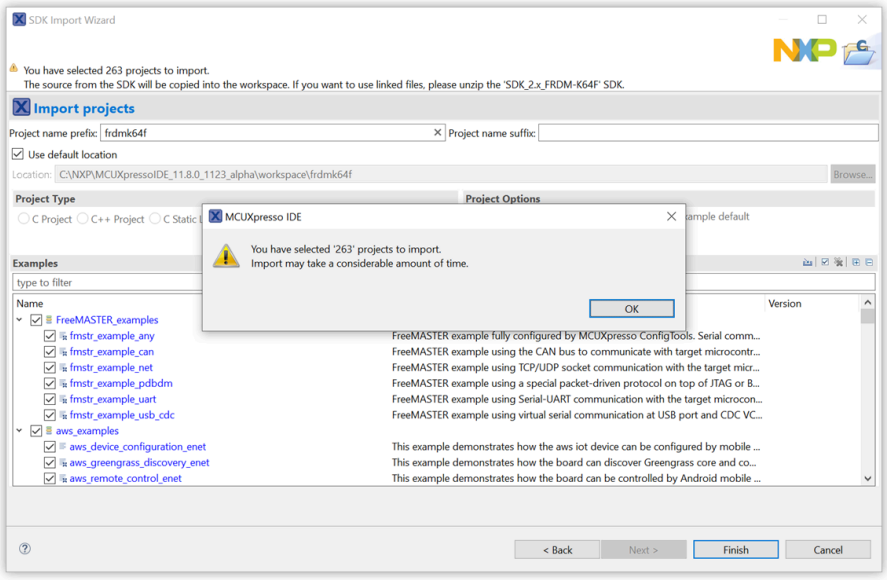


Figure 6.6. SDK Example Selection Many

6.1.2 SDK Example Import Wizard: Advanced options

The advanced configuration page (shown below) will take certain default options based on the examples selected; for example, a C project will pre-select Redlib libraries, whereas a *C++* project will pre-select NewlibNano.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

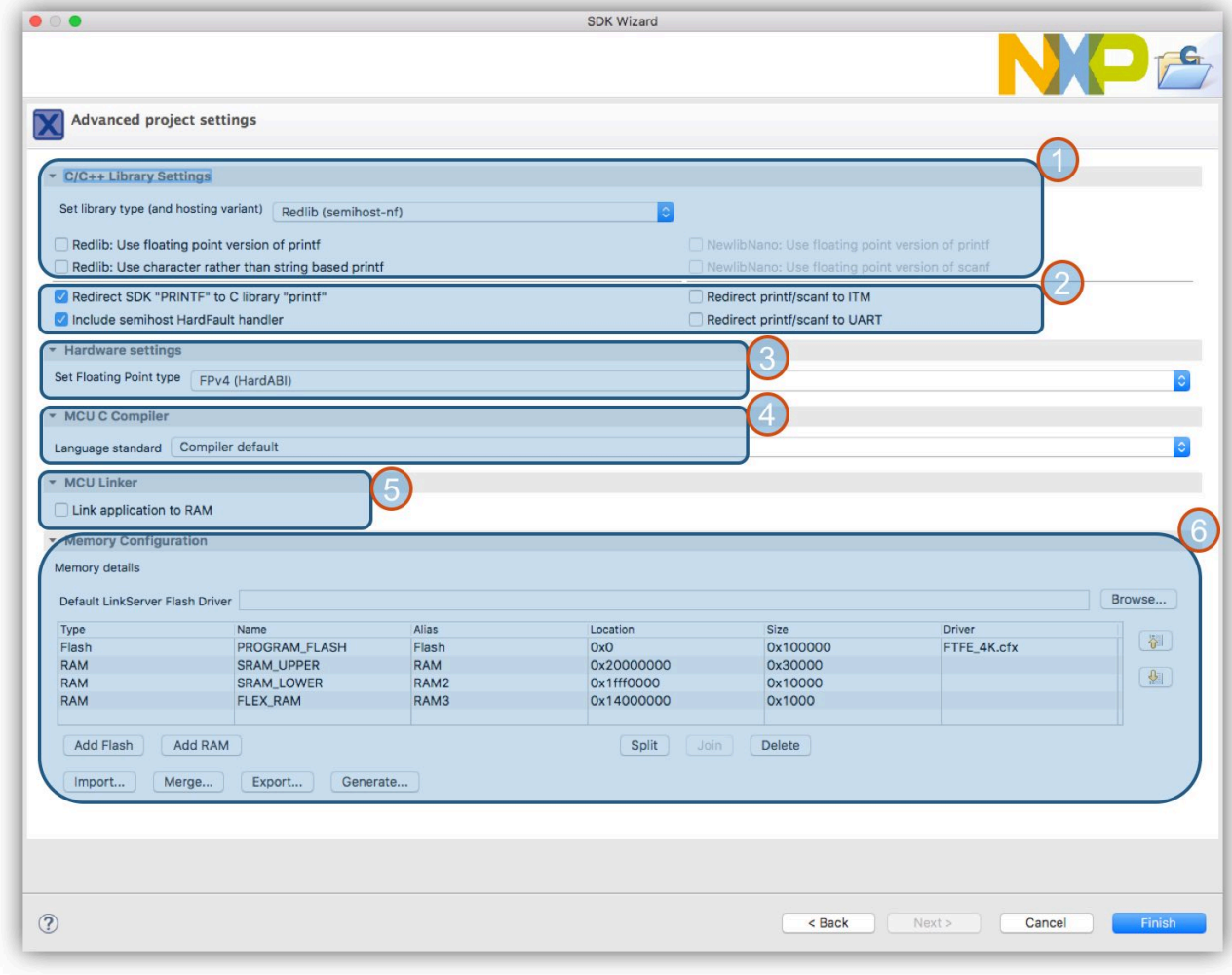


Figure 6.7. New Project Wizard advanced SDK settings

These settings closely match those in SDK New Project Wizard description. Therefore see SDK New Project Wizard:Advanced Options [58] for a description of these options. **Note**: Changing these advanced options may prevent an example from building or executing.

6.1.3 SDK Example Import Wizard: Import from XML fragment

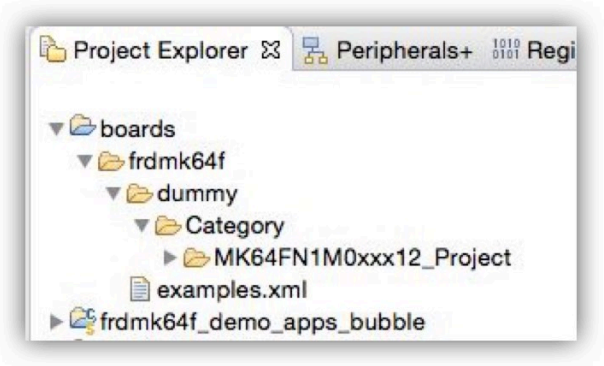
This option works in conjunction with the *'Project Explorer' -> Tools -> Generate Example XML* (and is also used to import project created by the MCUXpresso Config Tools Project Generator).

The functionality here is to merge existing sources within a selectable board package framework.

To create an XML "fragment" for an existing project in your workspace, right click on the project in the 'Project Explorer' (or just in the 'Project Explorer' view with no project selected) and choose *Tools->Generate examples.xml file*

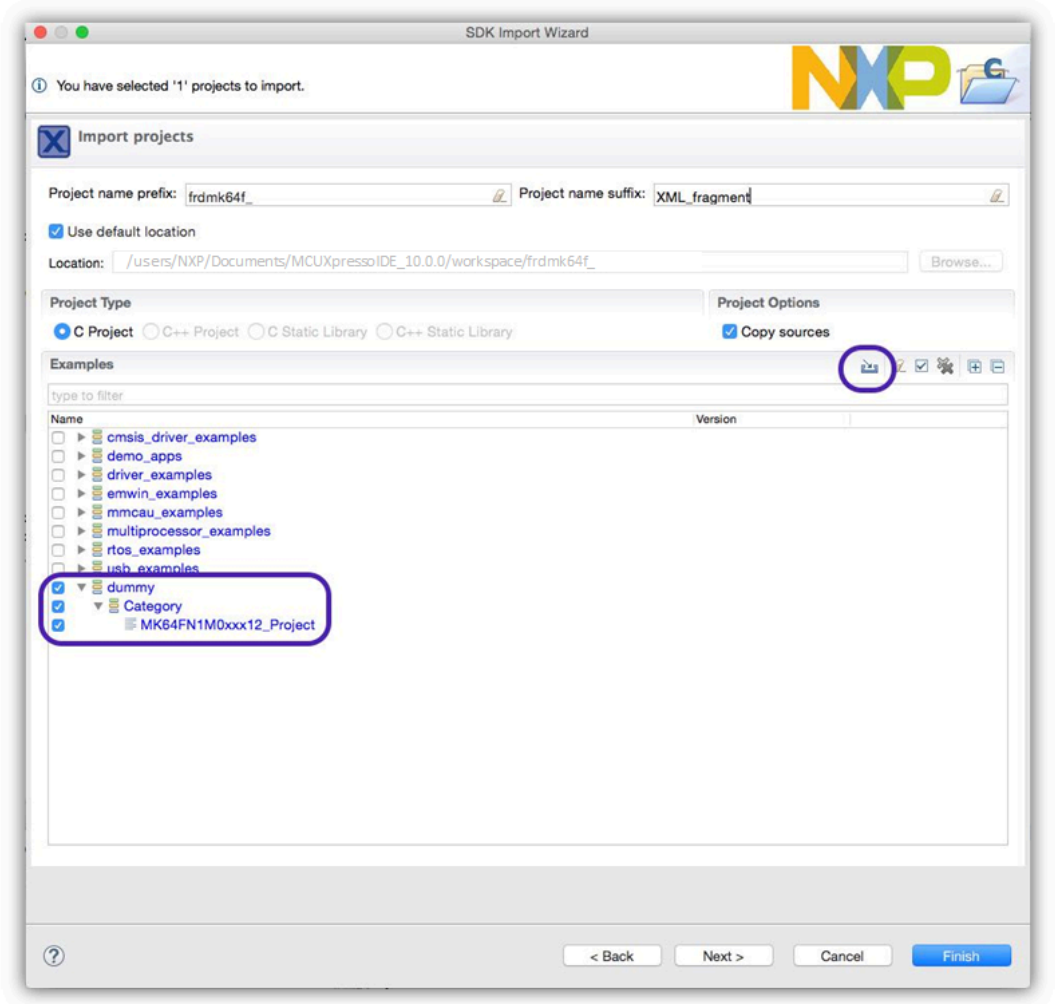
The selected project or all the projects in the workspace (if no projects are selected) will be converted into a fragment within a new folder created in the workspace itself:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



To create a project from a fragment, click on “Import SDK examples…” in the **Quickstart** Panel view:

Then select a board and then click on the button “Import from XML…” (highlighted below and described in the previous section). You will see the examples definitions from the external fragment in list of examples as shown and selected below.



Select the external examples you want to re-create and click on “Finish”. The project(s) will be created in the workspace.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

6.1.4 Importing Examples to non default locations

By default, imported example sources will be stored within the current MCUXpresso IDE workspace, **this is recommended since the workspace then contains both the sources and project descriptions**. However, the Import SDK Example Wizard allows a non default location to be specified if required. To ensure that each project's sources and local configuration are self contained when using non standard locations, the IDE will automatically create a sub directory inside the specified location using the *Project name prefix* setting. Single or multiple imported projects will then be stored within this location.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

7. SDK Project Component Management

Projects and examples created from SDKs contain a number of software components such as peripheral drivers and/or middleware. In previous versions of MCUXpresso IDE, the option to add components was only available when creating a new project and not possible for imported examples. MCUXpresso IDE version has the ability to easily add (or remove) SDK components to a previously created or imported example project via a new *Manage SDK components* wizard. To launch the Manage SDK Components wizard, simply select the chosen project in the Project explorer view and then click the *package* icon as indicated below:

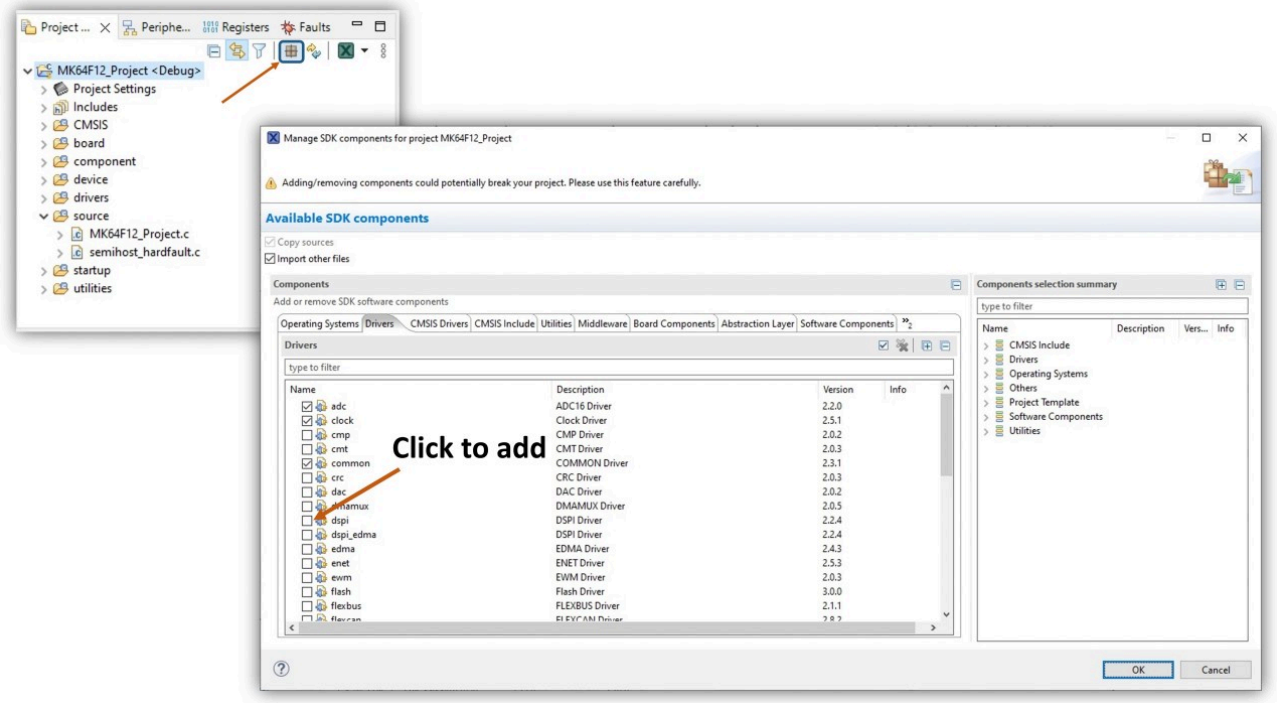


Figure 7.1. Manage SDK Components

Note: This powerful feature can add (or remove) SDK components **and their dependencies** at a source file level, relying on meta data contained within the SDK. However the following points should also be noted:

- The IDE can only maintain dependencies between SDK components. SDK component functions referenced from user-created files or from sources such as an SDK example's main() function will not be taken into account when determining the safe removal of components. Therefore, the IDE cannot always prevent users removing components that may actually be required for a successful project build.
- Defined symbols will not be removed when components are removed, therefore users should ensure only required symbols are present if components are removed. Failing to do this may lead to project build failures.

Various SDK Component Management options are available from *Preferences -> MCUXpresso IDE -> SDK Handling -> Components* .

7.1 SDK Project Component Management example

To demonstrate the use of this feature, the *dac* driver will be added to a project. To do this, launch the Manage SDK components wizard, and click on the *dac* driver component then click 'OK'.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Next, a dialogue will be presented listing all of the source files required by this component – as below.

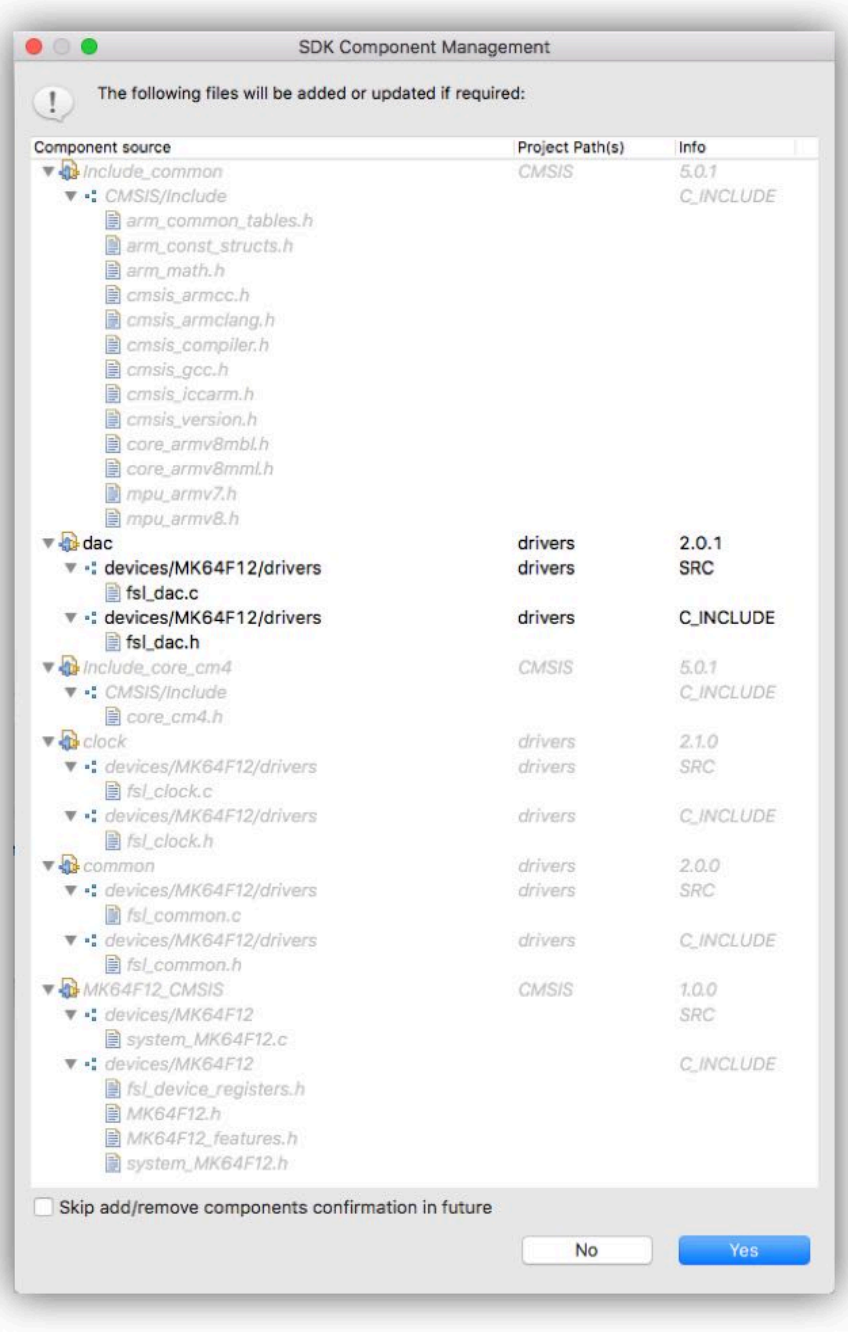


Figure 7.2. SDK Component Management

Note: Many of these files may already be included within your project.

Click ‘Yes’ to add these source files to your project.

Important Note: Since your project may contain edited or entirely new versions of the required source files, MCUXpresso IDE will perform a comparison between the new files to be included and any existing files already within the selected project.

Should a source file difference be found, then a dialogue as below will be launched:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

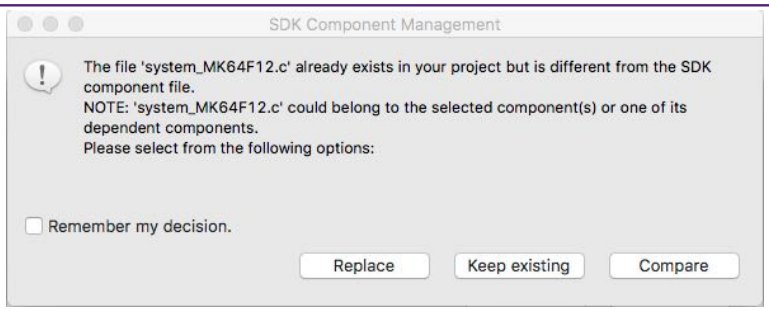


Figure 7.3. SDK Component Management file difference

From here you can choose from the following options:

- **Replace** click to overwrite the project's file from the SDK version.
- **Keep Existing** click to keep the existing project file unchanged.
- **Compare** click to compare the two files – this will launch the Eclipse file compare utility so the new SDK file can be compared with the projects copy.

In this example, we will click 'Compare' ...

Below, you can see that a user project source modification has been found:

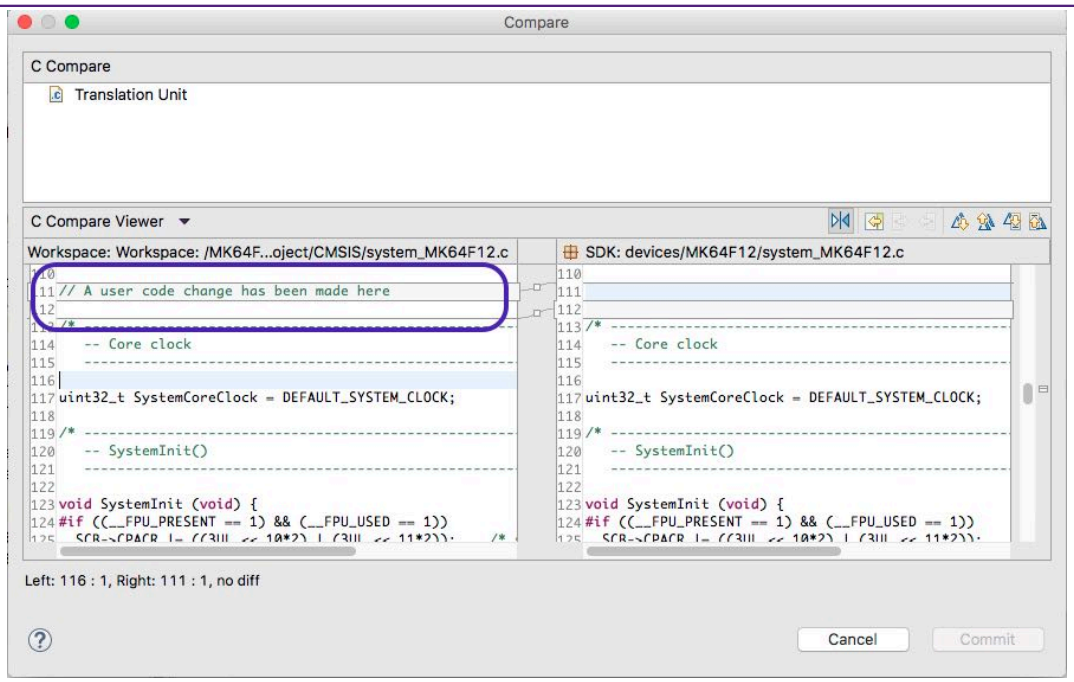


Figure 7.4. SDK Component Management file compare

The Compare utility allows any change to be examined and a decision made regarding which code lines to choose or ignore. When the decisions have been made, click 'Commit' to use these changes or 'Cancel' to return to the dialog and decide the action to take for the file.

Finally please note the application build sizes before the addition:

```
Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       13348 B         1 MB      1.27%
      SRAM_UPPER:        8444 B       192 KB      4.29%
      SRAM_LOWER:           0 GB        64 KB      0.00%
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
        FLEX_RAM:          0 GB         4 KB      0.00%
Finished building target: MK64FN1M0xxx12_Project.axf
```

Followed by the application sizes after the addition.

```
Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       13348 B         1 MB      1.27%
      SRAM_UPPER:        8444 B       192 KB      4.29%
      SRAM_LOWER:           0 GB        64 KB      0.00%
        FLEX_RAM:           0 GB         4 KB      0.00%
Finished building target: MK64FN1M0xxx12_Project.axf
```

These are exactly the same!

This is because although new source files have been added to the project, they will (probably) not be referenced by any code in the project and hence no new functions or data will be included in the final image. To make use of any new component, some of its new functionality must of course be referenced.

Note: Some middleware components such as USB, are not compatible with the Add/Remove component functionality and so will be hidden from the Add/Remove dialogue. The recommended approach if such components are required is to import an example including the component and modify as required. This restriction will be addressed in a future release.

Please also see Image Information [203] for details of how to explore the composition of an image in detail.

7.2 SDK Project Refresh

Using the above technology, MCUXpresso IDE projects can be refreshed with updated SDK components.

When new SDKs are released for a particular MCU/Board, many source files will be updated, bugs fixed, features added etc. If an existing SDK is replaced within MCUXpresso IDE by such a new SDK, any updated (or changed) source files or source file sections can optionally be added to existing project using an identical mechanism as described above.

To use this feature, simply select a project in the project explorer view and click to Refresh SDK Components as indicated below.

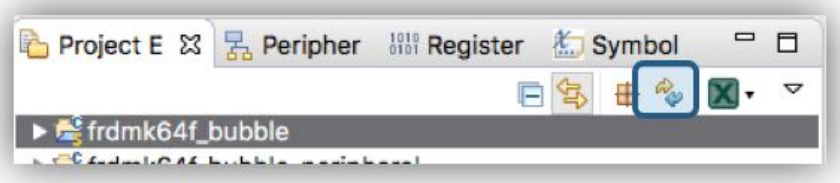


Figure 7.5. SDK Component Management Project Refresh

The SDK Component Management wizard will guide you through the update process.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

8. Open-CMSIS Component Management

MCUXpresso IDE integrated ARM CMSIS Plugins to explore Open-CMSIS packs and import (middleware) components into an Eclipse project. With this feature, the desired CMSIS-Pack can be installed via CMSIS-Pack Management (*Perspective -> Open Perspective -> Other -> CMSIS-Pack Manager*) and middleware components can be added into the project using RTE Configuration view.

8.1 Install a Pack

To start adding components, firstly open CMSIS-Pack Manager and install the needed packs.

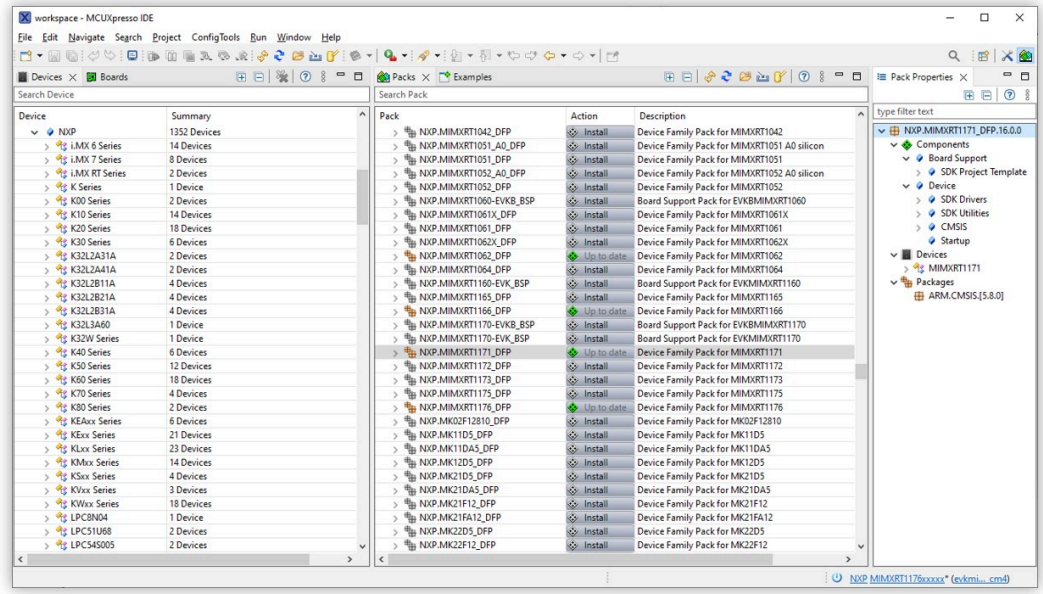


Figure 8.1. CMSIS-Pack Manager

From Packs view (toolbar) you can: Reload, Check for updates on Web, Import Packs from disk, etc. To install a pack, select one, click on **Install**, accept the license and once the pack is installed its status will be **Up to date** and a green icon will appear.

Note: First time when the CMSIS-Pack Manager is used, an index update will be done to populate the list of available packs on web.

Once the packs are installed, go back to **Develop** Perspective.

8.2 Add an Open-CMSIS-Pack component to a project

Create an NPW/SDK example for your device and add the support for Open-CMSIS by right-clicking on project entry in *Project Explorer -> SDK Management -> Add Open-CMSIS Components*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

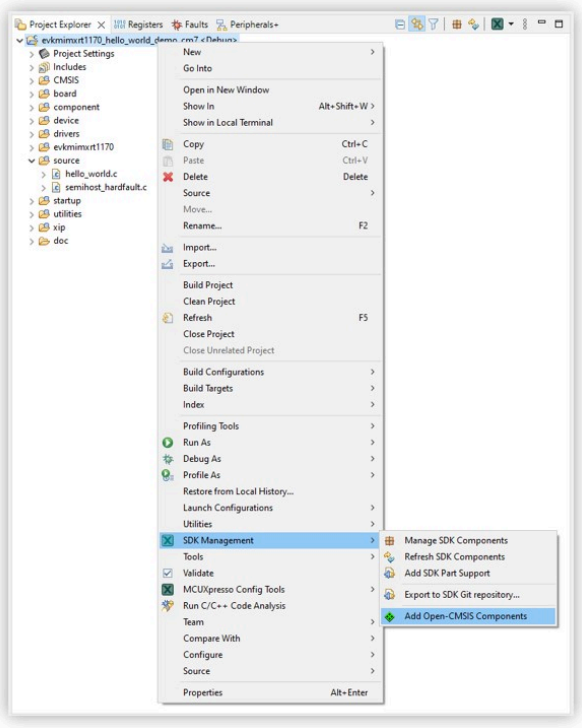


Figure 8.2. Add Open-CMSIS Components

RTE Configuration view will open and display all the available components from installed packs.

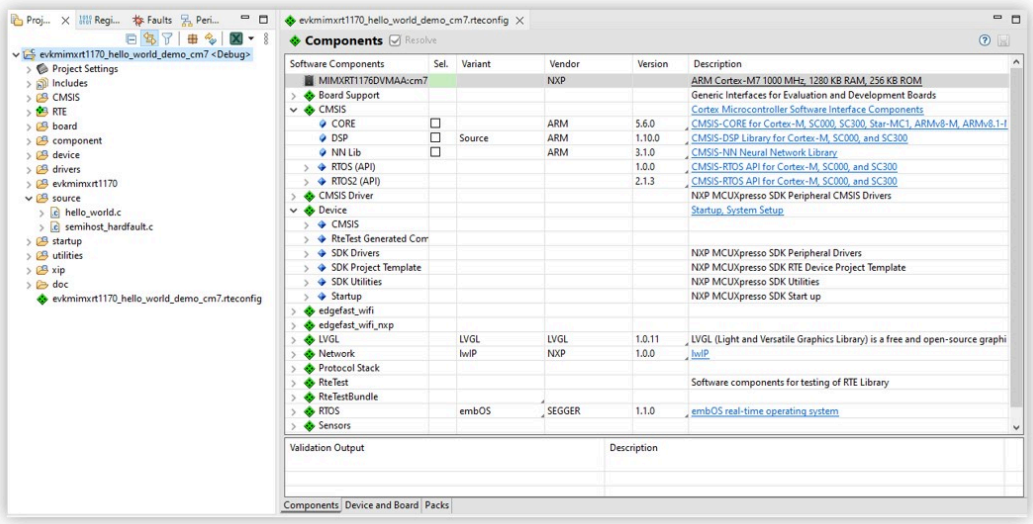


Figure 8.3. RTE Configuration view

8.3 Manage components inside the project

This feature:

- • Allows installation of multiple components, automatically checking and resolving dependencies.
  - Automatically generates required configuration, template and header files
  - Updates project configuration (e.g. build settings, compiler flags, include paths, linked libs, etc.)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

9. Creating New Projects using Preinstalled Part Support

For Creating project using SDKs please see Creating New Projects using installed SDK Part Support [53]

To explore the range of preinstalled parts/MCUs simply click ‘New project’ in the **Quickstart** panel. This will open a page similar to the figure below:

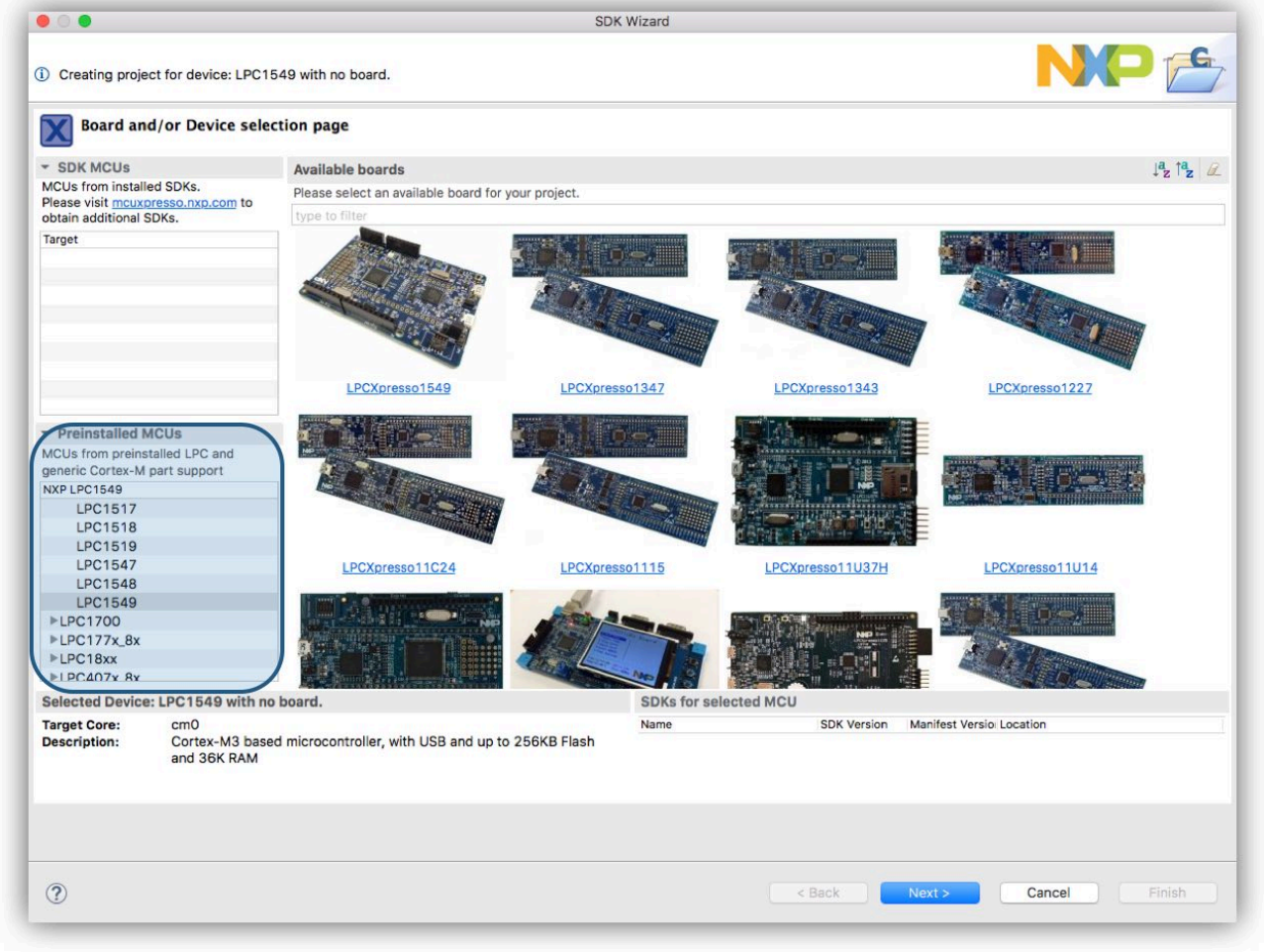


Figure 9.1. New Project Wizard Preinstalled

The list of preinstalled parts is presented on the bottom left of this window.

You will also see a range of related development boards indicating whether a matching board support library (LPCOpen or CodeBundles) is available.

For details of this page see: New Project Wizard details [53]

9.1 New Project Wizard

This wizard page provides a number of ways of quickly selecting the target for the project that you want to create.

In this description, we are going to create a project for an LPC4337 MCU. For this MCU an LPCOpen library is available, so we can locate this MCU using the board filter. **Note**: Boards will be displayed where either LPCOpen or CodeBundle projects exist.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: LPCOpen is described in section LPCOpen Software Drivers and Examples [85]

To reduce the number of boards displayed, we can simply type '4337' into the filter so only boards with MCUs containing '4337' will be displayed.

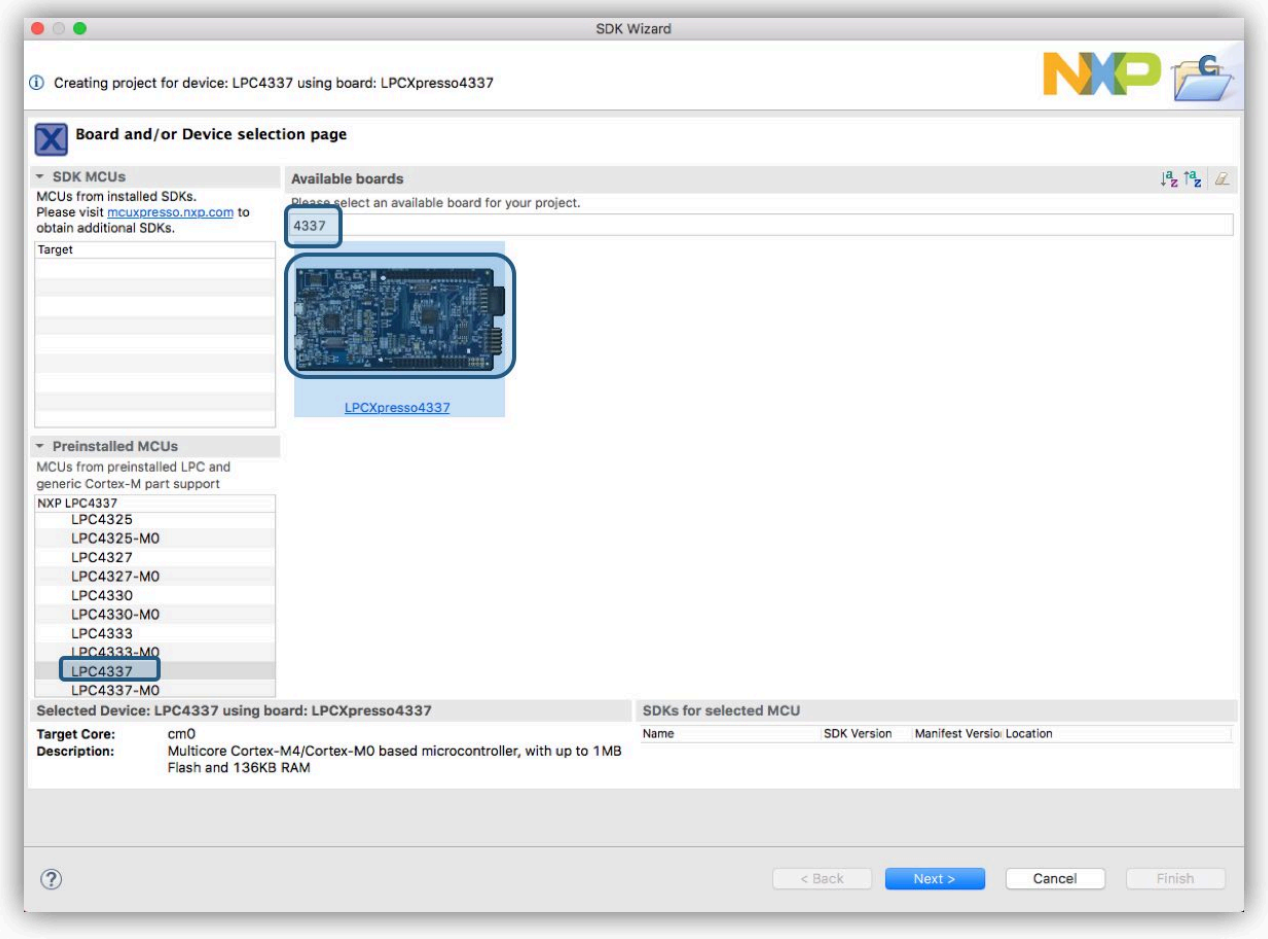


Figure 9.2. New Project Wizard selection for Preinstalled MCUs

When a board is selected as highlighted in the above figure, the matching MCU (part) is also selected automatically.

Note: if no matching board is available, the required MCU can be selected from the list of Preinstalled MCUs.

Note: Boards added to MCUXpresso IDE from SDKs will have an 'SDK' graphic superimposed on the board image. Boards without the SDK graphic indicate that a matching LPCOpen package (or Code bundle) is available for that board and associated MCU.

With a chosen board selected, now click 'Next' to launch the next level of wizards. These wizards for Preinstalled MCUs are very similar to those featured in LPCXpresso IDE and are described in the next section.

9.2 Creating a Project

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

This New Project wizard supports 2 types of projects:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Those targeting LPCOpen libraries
- Standalone projects

In addition, certain MCUs like the LPC4337 support multiple cores internally, for these MCUs, Multicore options will also be presented (as below):

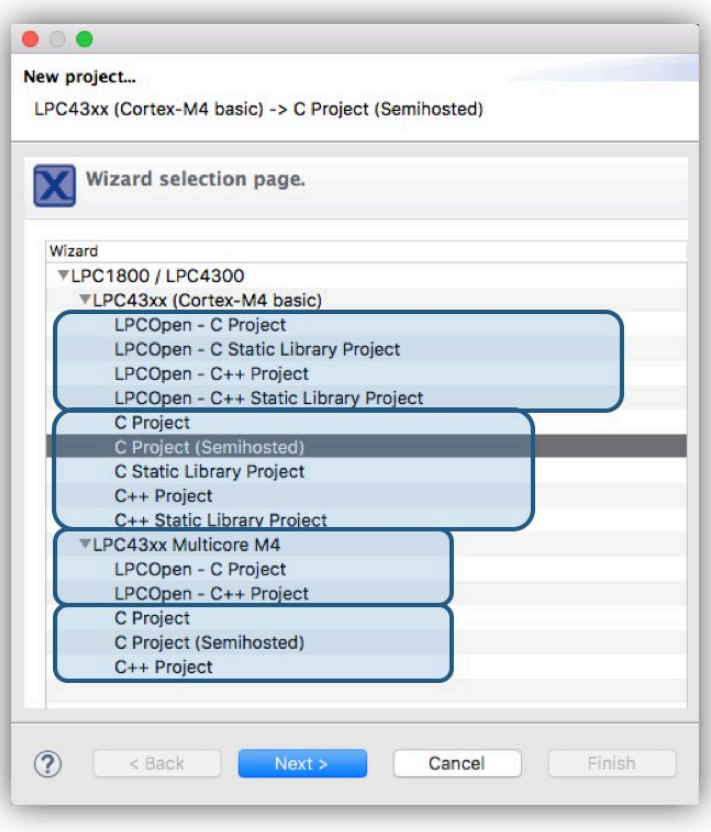


Figure 9.3. New project: wizard selection

You can now select the type of project that you wish to create (see below for details of Wizard types).

In this case, we will show the steps in creating a simple C 'Hello World' example project.

9.2.1 Selecting the Wizard Type

For most MCU families MCUXpresso IDE provides wizards for two forms of project: LPCOpen and non-LPCOpen. For more details on LPCOpen, see Software drivers and examples [85] . 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 an 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 an LED.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

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:

https://community.nxp.com/message/630594

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 MCUXpresso IDE. This is implemented using “semihosting” functionality. See the section on Semihosting [191] for more information.

9.2.2 Configuring the Project

Once you have selected the appropriate project wizard, you will be able to enter the name of your new project, this must be unique for the current workspace.

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 MCUXpresso IDE. **Note**: 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.

9.2.3 Wizard Options

The wizard will present a set of pages (that will vary based on the chosen MCU), many of these pages will typically require no user change since the common default values are already preset. The pages may include:

LPCOpen Library Project Selection

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

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

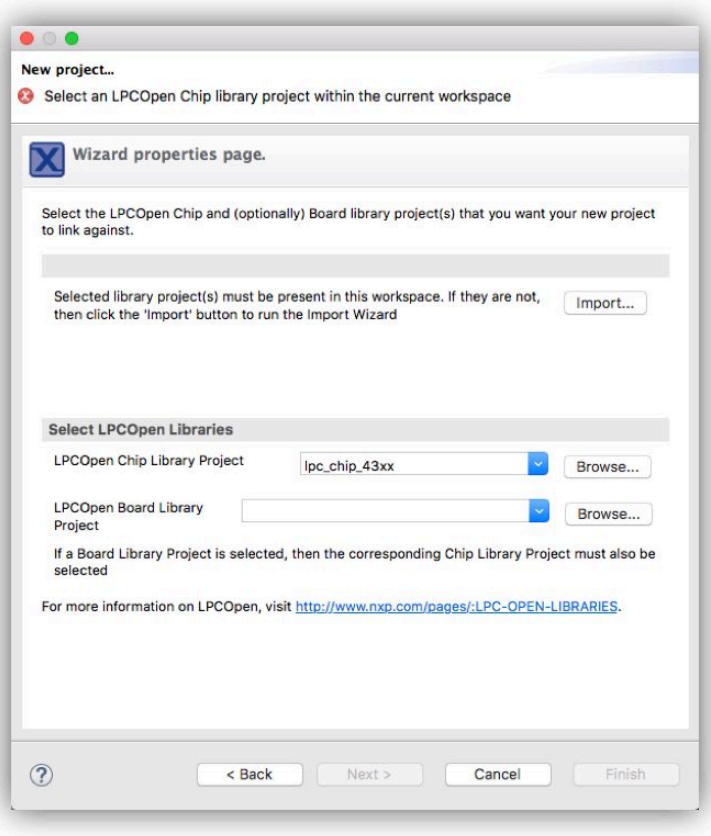


Figure 9.4. LPCOpen library selection

This page allows you to run an “Import wizard” to download the LPCOpen bundle for your target MCU/board from https://www.nxp.com/lpcopen 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 drop down next to the Browse button). **Note**: 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 drop down next to the Browse button). Although selection of a board library is optional, it is recommended that you do this in most cases.

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. MCUXpresso 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 MCUXpresso 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). MCUXpresso 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.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

The CMSIS library option within MCUXpresso IDE allows you to select which (if any) CMSIS-CORE library you want to link to from the project you are creating. **Note**: 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 MCUXpresso IDE, please see the FAQ at:

https://community.nxp.com/message/630589

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 [85]

CMSIS DSP Library Selection

ARM's **Cortex Microcontroller Software Interface Standard** (or **CMSIS**) specification also provides a definition and implementation of a DSP library. MCUXpresso 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 MCUXpresso IDE Examples.

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

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 MCUXpresso 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.

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.

Code Read Protect

NXP's Cortex based LPC 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. MCUXpresso IDE provides support for setting this memory location. See the section on Code Read Protection [215] for more information.

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:

https://community.nxp.com/message/630743

Disable Watchdog

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

https://community.nxp.com/message/630654

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

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".

Memory Configuration Editor

For certain MCUs such as the LPC18xx and LPC43xx, the wizard will present the option to edit the target memory configuration. This is because these parts may make use of external SPIFI Flash memory and hence this can be described here if required. For more information please see: LinkServer Flash Support [173] and also Memory Configuration and Linker Scripts [199]

Note: Memory configuration can of course also be edited after a project has been created.

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

For more information see C/C++ Library Support [188]

9.2.4 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 10.5 and Chapter 13.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

10. Importing Example Projects (from the file system)

MCUXpresso IDE supports two schemes for importing examples:

- From SDKs – using the **Quickstart** Panel -> Import SDK example(s). See Importing Examples Projects (from SDK) [62]
- From the filing system – using the **Quickstart** Panel -> Import project(s) from file System
  - this option is discussed below:



Drag and Drop

MCUXpresso IDE project(s) can be imported directly into a workspace by simply dragging a folder (or zip) containing MCUXpresso IDE projects onto the Project Explorer view. **Note**: this will import all projects within a folder (or zip). Projects can also be exported by dragging directly from the Project Explorer view onto a filer, or directly into another instance of the IDE. See Enhanced Project Sharing Features [45] for more information. Due to underlying Eclipse changes in Version 11.1.0, *Drag and Drop* can only be used to import projects, when one or more project already exists within a Workspace

Note: This option can also be used to import projects exported from MCUXpresso IDE. See Exporting Projects [88]

MCUXpresso IDE installs with a large number of example projects for preinstalled parts, that can be imported directly into a workspace: These are located at:

```
<install_dir>/ide/Examples
```

and consist of:

- CMSIS-DSPLIB
  - a suite of common signal processing functions for use on Cortex-M processor based devices.
- CodeBundles for LPC800 family
  - which consist of software examples to teach users how to program the peripherals at a basic level.
- FlashDrivers
  - example projects to create Flash driver used by LinkServer
- Legacy
  - a range of historic examples and drivers including CMSIS / Peripheral Driver Library
- LPCOpen
  - High quality board and chip support libraries for LPC MCUs, plus example projects

10.1 Code Bundles for LPC800 Family Devices

The LPC800 Family of MCUs are ideal for customers who want to make the transition from 8 and 16-bit MCUs to the Cortex M0/M0+. For this purpose, we've created Code Bundles which consist of software examples to teach users how to program the peripherals at a basic level. The examples provide register level peripheral access, and direct correspondence to the memory map in the MCU User Manual. Examples are concise and accurate explanations are provided within any readme and source file comments. Code Bundles for LPC800 family devices are made available at the time of the series product launch, ready for use with a range of tools including MCUXpresso IDE.

More information on code bundles together with latest downloads can be found at:

https://www.nxp.com/LPC800-Code-Bundles

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

10.2 LPCOpen Software Drivers and Examples

Note: LPCOpen is no longer under active development, new MCU's from NXP are supported by SDKs. Certain parts such as some members of the LPC54xxx families are available with both LPCOpen and SDK support.

LPCOpen is an extensive collection of free software libraries (drivers and middleware) and example programs that enable developers to create multifunctional products based on LPC microcontrollers. Access to LPCOpen is free to all LPC developers.

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/MCUXpresso IDE 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 MCUXpresso IDE

CMSIS / Peripheral Driver Library / code bundle software packages are still available, from within your *install_dir/ide/Examples/Legacy* folder. However, these should only be used for existing development work. When starting a new evaluation or product development, we would recommend the use of LPCOpen if available.

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

https://www.nxp.com/lpcopen

10.3 Importing an Example Project

To import an example project from the file system, locate the **Quickstart** panel and select 'Import projects from Filesystem'

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

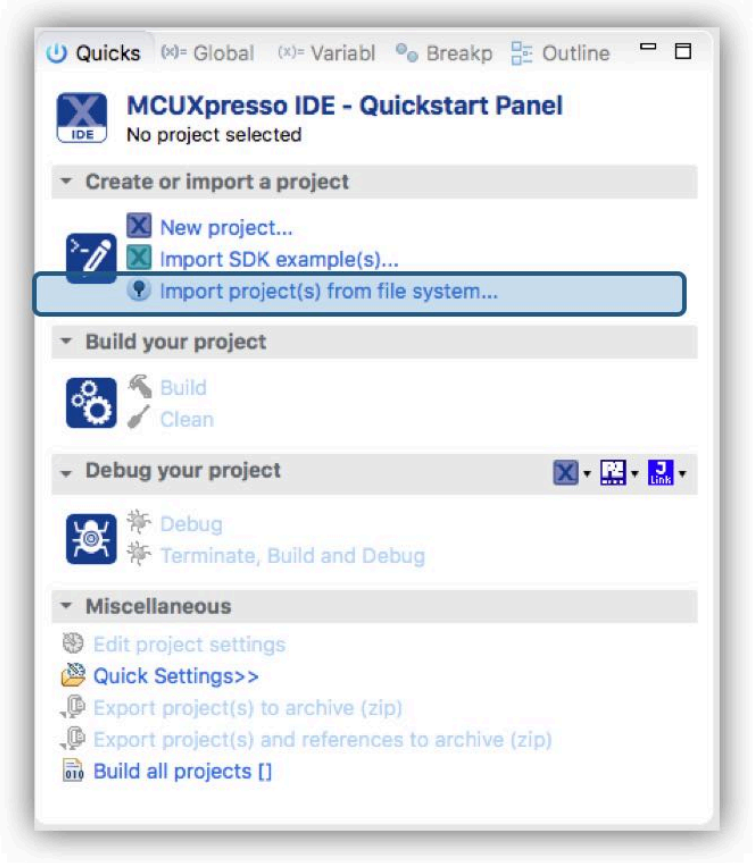


Figure 10.1. Importing project(s)

From here you can browse the file system.

© 2023 NXP Semiconductors. All rights reserved.

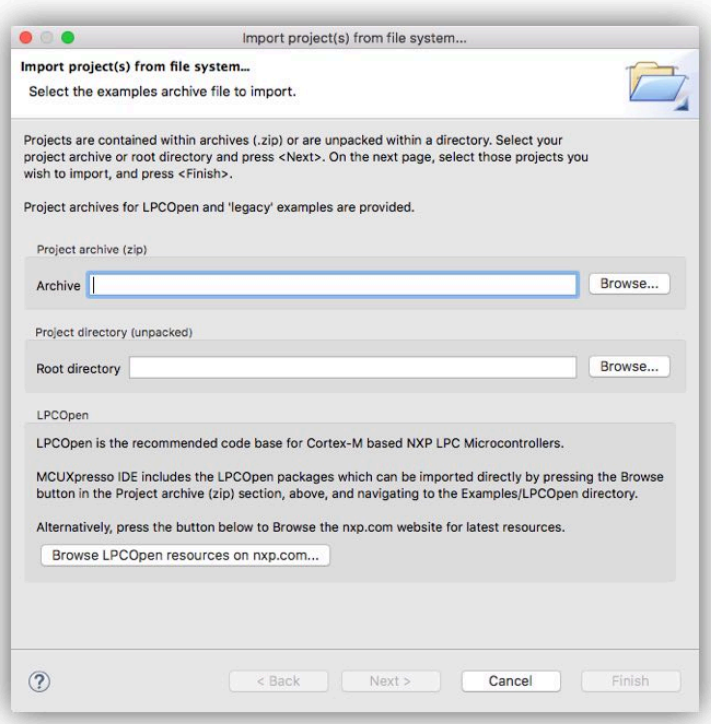


Figure 10.2. Importing examples

- **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 https://www.nxp.com/lpcopen or the supplied, but deprecated, sample code located within the Examples/Legacy subdirectory of your MCUXpresso 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 resources** to visit https://www.nxp.com/lpcopen and download an appropriate LPCOpen package for your target MCU. This option will automatically open a web browser onto a suitable links page.

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

10.3.1 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 Resources**, which will open a web browser window. Click on **LPC4300 Series**, and then locate **NXP LPCXpresso4337**, and then download **2.xx** version for LPCXpresso Toolchain (LPCOpen packages created for LPCXpresso IDE are compatible with MCUXpresso IDE).

Note: LPCOpen Packages for the LPC4337 are preinstalled and located at:

```
<install_dir>/ide/Examples/LPCOpen/...
```

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 10.3.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

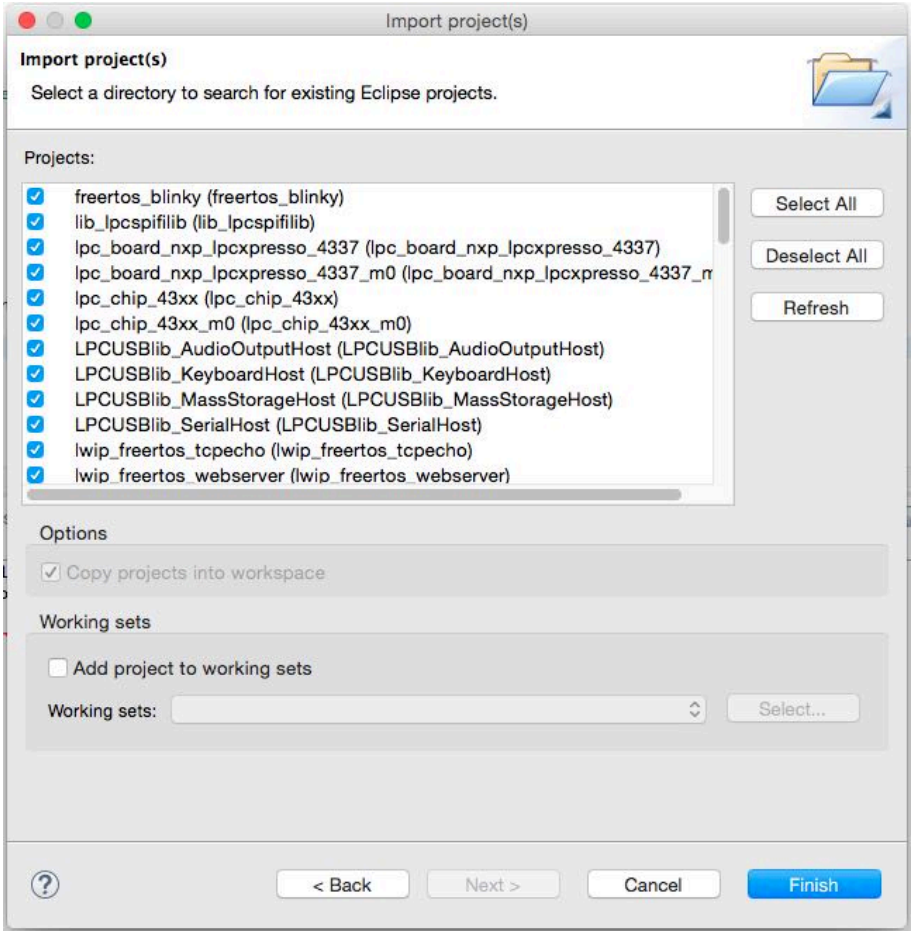


Figure 10.3. Selecting projects to import

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

Note: 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.

10.4 Exporting Projects

MCUXpresso IDE provides the following export options from the **Quickstart** panel:

- Export project(s) to archive (zip)
- Export project(s) and references to archive (zip)
  - choose this option to export project(s) and automatically also export referenced libraries

To export one or more projects, first select the project(s) in the **Project Explorer** then from the **Quickstart** Panel -> Export project(s) to archive (zip). This will launch a filer window. Simply select the destination and enter a name for the archive to be exported then click ‘OK’.

Also please see Enhanced Project Sharing Features [45] for information about dragging and dropping projects.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

10.5 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**: building a single project may also trigger a build of any associated or referenced project.

10.5.1 Build Configurations

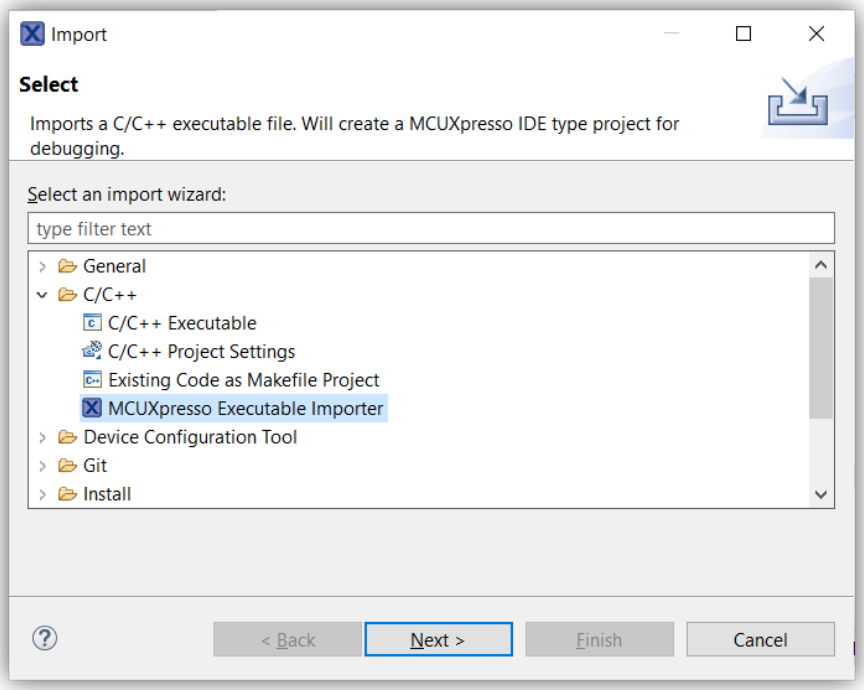
By default, each project will be created with two different “build configurations”: **Debug** and **Release**. Each build configuration will contain a distinct set of build options. Thus a **Debug** build will typically compile its code with optimizations disabled (`-O0`) and **Release** will compile 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 How do I switch between Debug and Release builds? [262]

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

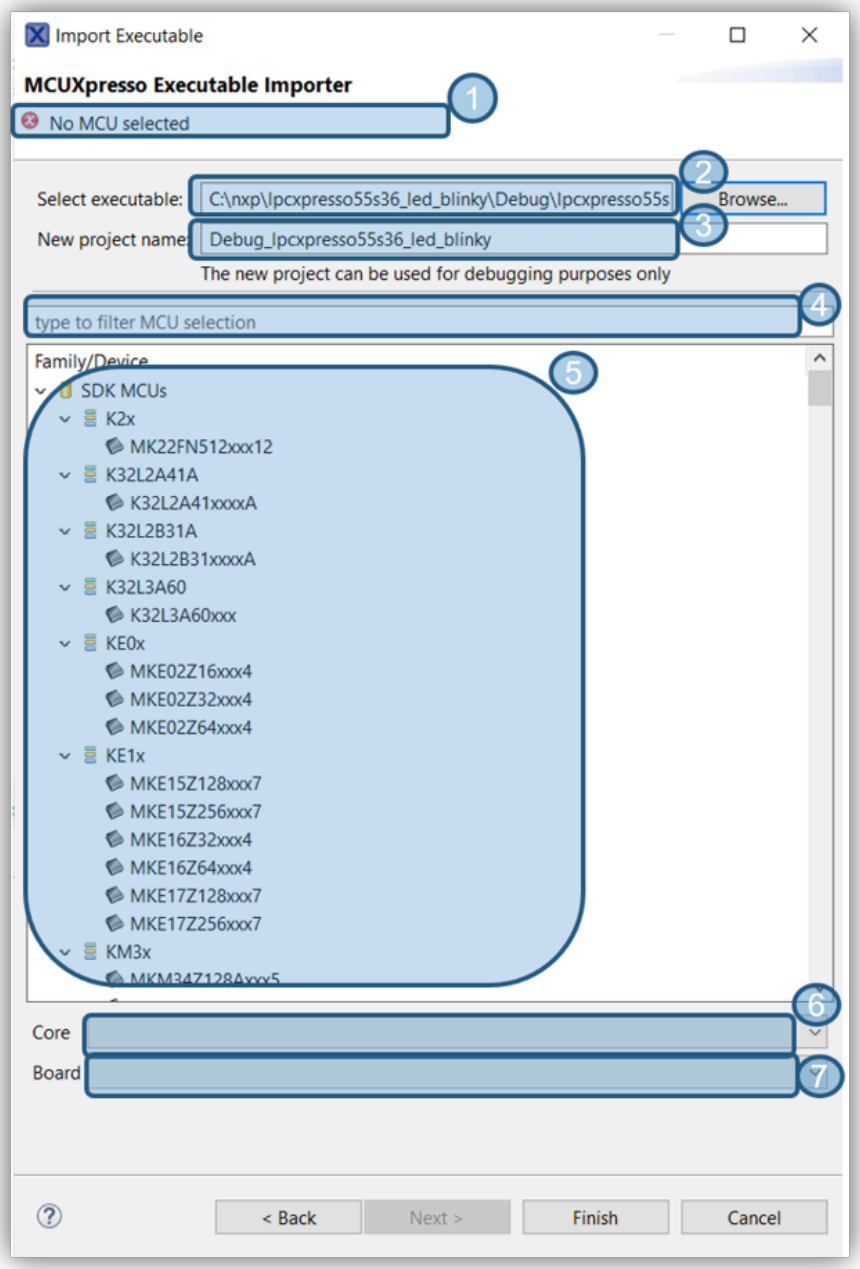
11. Importing Existing Executables

Existing executables can also be imported and further used for debugging with the MCUXpresso IDE. Importing an existing executable will generate a new project that can be used to attach or download the executable to the target; when debug is used, the code will first be downloaded to Flash and then debugging will start, while attach is used to debug an application that was already flashed (see Connecting to a running Target (attach) [124] for more details). It's important to note that the generated project cannot be used for rebuilding the executable. The newly created project will contain a symbolic link to the imported executable (the executable will not be copied or moved from its original location). The "MCUXpresso Executable Importer" can be accessed by going to "File" -> "Import" and then expanding the "C/C++" category. The wizard allows selection of files having "elf"/"axf" extension.



Once opened, several information is expected to be provided in order to finish the wizard. The fields are described below.

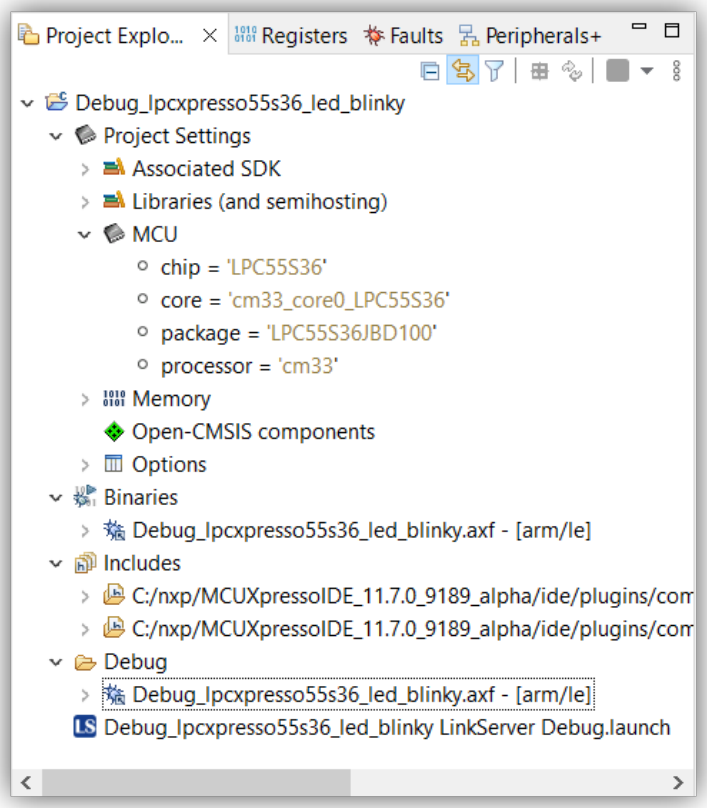
All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



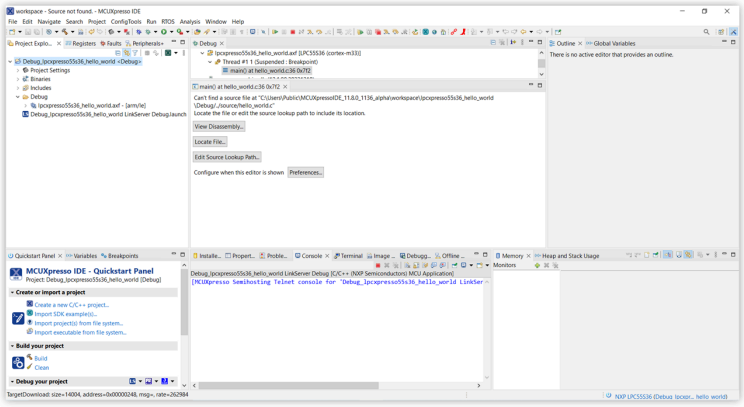
1. Wizard validation status. Validation errors will appear here. 2. Path to the C/C++ executable file. 3. Name to be assigned to the newly created project. 4. Text filter used to filter the available MCUs. 5. The IDE needs to associate an MCU to the newly created project. Please select one from the list. Note that the MCUs from the list are identified based on the installed SDKs and on the available preinstalled parts. 6. In case of a multicore device, the core to be used for debugging must be also selected. This can be done using the “Core” drop-down. 7. The “Board” drop-down allows selection of the actual board.

Once the wizard is finished, a new project will appear in Project Explorer, as illustrated in the picture below. See Debugging a Project [112] for details about how to start using the project for debugging.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

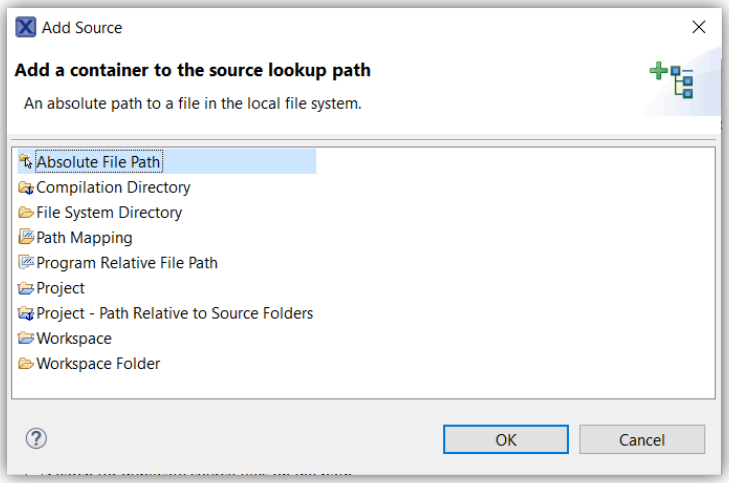


If the sources that were used to build the executable are still available in the original build folder then they will be accessible for source level debugging. However, since this is not usually the case you will need to specify the location of the sources in order to access them from your project.



At this point you can press “Locate File...” and add the source file location. Another option for source mapping is to press “Edit Source Lookup Path...” and then press the “Add” button, finally you should add a container to the source lookup path. This entry can later be edited or removed as needed.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

12. Debug Solutions Overview

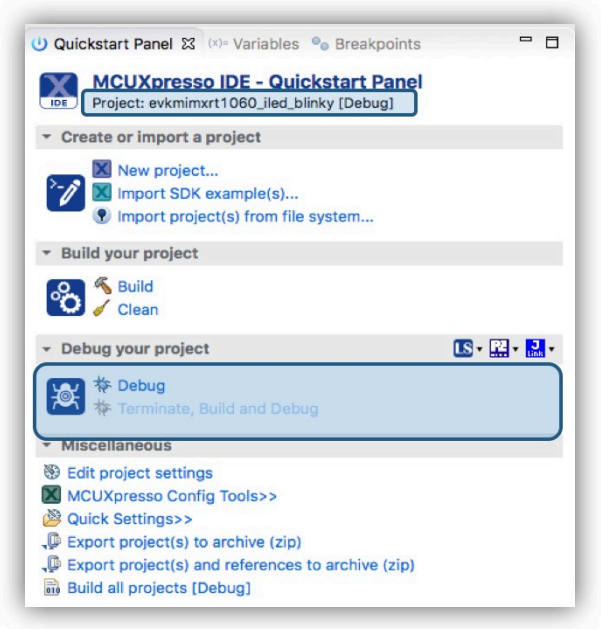
MCUXpresso IDE installs with built-in support for 3 debug (hardware) solutions; comprising the Native LinkServer (including CMSIS-DAP) [98] as used in LPCXpresso IDE. Plus support for both PEmicro [104] and SEGGER J-Link. [106] . This support includes the installation of all necessary drivers and supporting software.

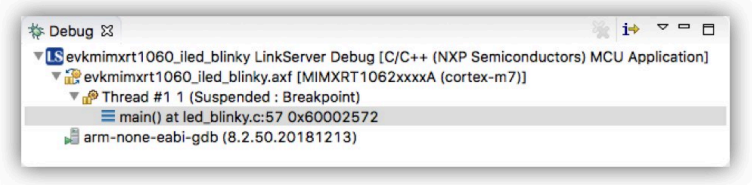
The rest of this chapter discusses these different Debug solutions. For general information on debugging please see the chapter Debugging a Project [112]

Note: Within MCUXpresso IDE, the debug solution used has no impact on project setting or build configuration. Debug operations for basic debug are also identical.

12.1 Starting a Debug Session

With a suitable board and debug probe connected (usually via USB), to start a debug session:

1. select a project to debug within the MCUXpresso IDE Project View
2. click **Debug** from within the MCUXpresso IDE **Quickstart** View
   - 

   - a *debug probe discovery* operation is automatically performed to display the available debug connections (i.e. the detected debug probes), including LinkServer, PEmicro and J-Link compatible probes.
3. select the required debug probe and click **OK**
   - at this stage a project launch configuration [95] is automatically created within the project containing debug specific configurations
   - if the debug connection is successful a Debug view will appear typically showing the project has stopped on main()
   - 

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Tip

Once a project has been debugged, the launch configuration will contain details of the debug probe used. Subsequent debug sessions will automatically select this probe if it is available.

From this point onwards, the low level debug operations are controlled by one of the debug solutions mentioned above.

However, from the user's point of view most common debug operations within the IDE will appear the same (or broadly similar), for example:

- Automatic inheritance of part knowledge
- Automatic downloading (programming) of generated image to target Flash memory
  - LinkServer/CMSIS-DAP Flash programming – see the chapter Introduction to LinkServer Flash Drivers [173]
- Automatic halt on main() [123]
- Setting breakpoints [131] and watchpoints [133]
- Stepping [116] (single, step in step out etc.)
- Viewing and editing local variables [151], registers [135], peripherals [139], memory [153]
- Viewing and editing global variables [144]
- Live global variables [144]
- Viewing disassembly [152]
- Semihosted IO [191]
- Instruction Trace is supported for all debug solutions, please see Instruction Trace Guide for more information
- GUI Flash Tool [167]
- SWO Trace is supported for all debug solutions including profiling, interrupt trace etc. please see SWO Trace Guide for more information
- Viewing details of execution faults via the Faults View [137] (automatically displayed for faults generated during LinkServer debug, a pause will be required for other debug solutions)

Additional documentation is also available covering:

- Power/Energy Measurement – please see Energy Measurement Guide
- FreeRTOS Debug – please see FreeRTOS Debug Guide
- Azure RTOS ThreadX Debug – please see Azure RTOS ThreadX Debug Guide
- Zephyr RTOS Debug – please see Zephyr RTOS Debug Guide
- MQX RTOS Debug – please see MQX RTOS Debug Guide

Note: In addition MCUXpresso IDE will dynamically manage each debug solutions connection requirements allowing multiple sessions to be started without conflict. For debug of Multicore MCUs please refer to the section Debugging MultiCore Projects [254]

It is important to note that certain operations such as the handling of features via Launch configurations [118] may be different for each debug solution. Furthermore, advanced debug features and capabilities may vary between solutions and even similar features may appear different within the IDE.

PEmicro and SEGGER debug solutions also provide a number of advanced features, details can be found at their respective web sites.

12.2 An Introduction to Launch Configuration Files

The debug properties of a project in MCUXpresso IDE are held locally within each project in **.launch** files (known as Launch Configuration files).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Launch configuration files are different for each debug solution (LinkServer, PEmicro, SEGGER) and contain the properties of the debug connection (SWD/JTAG, and various other configurations etc.) and can also include a debug probe identifier for automatic debug probe matching and selection.

If a project has not yet been debugged, for example a newly imported or created project, then the project will not have a launch configuration associated with it.

When the user first tries to debug a project, MCUXpresso IDE will perform a **Debug Probe Discovery** operation and present the user with a list of debug probes found. **Note**: The Debug Solutions searched can be filtered from this dialogue as highlighted, removing options that are not required will speed up this process.

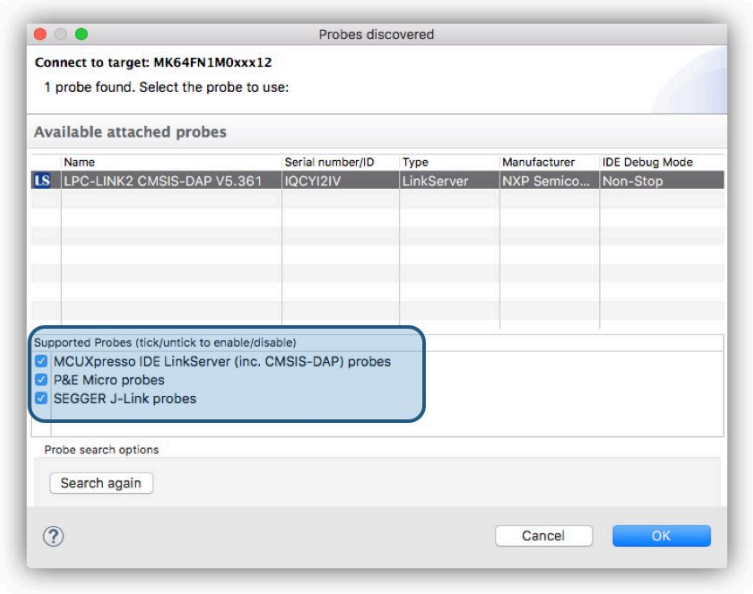


Figure 12.1. Debug Probe Discovery

Once the debug probe is selected and the user clicks ‘OK’, the IDE will automatically create a default launch configuration file for that debug probe (LinkServer launch configuration shown below).

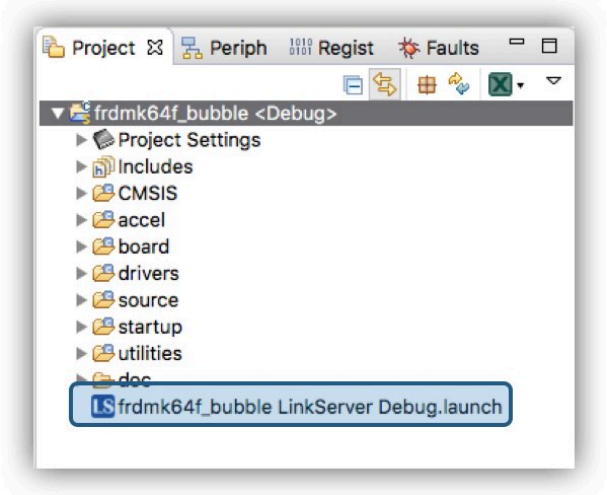


Figure 12.2. Launch Configuration Files

Note: a launch configuration will only be created for the currently selected build configuration.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

For many debug operations, these files won't require any attention and can essentially be ignored. However, if changes are required, these files should not be edited manually, rather their properties should be explored within the IDE.

The simplest way to do this is to click to expand the Project within the 'Project Explorer' pane, then simply double click a launch configuration file to automatically open the launch configuration *Edit Configuration* dialogue.

Note: This dialogue has a number of internal tabs, the *Debugger* tab (as shown below) contains the Debug main settings. See also the Project GUI Flash Tool [130]

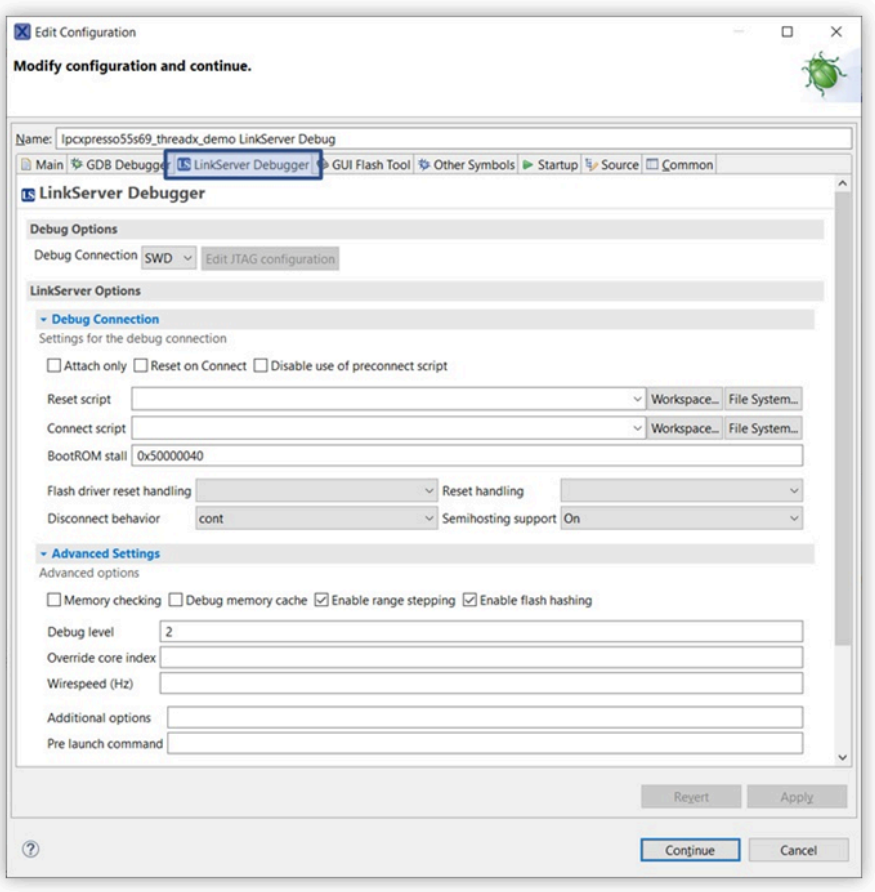


Figure 12.3. Launch Configuration

Some debug solutions support advanced operations (such as the recovering of badly programmed parts) from this view.

Note: Once a launch configuration file has been created, it will be used for the projects future debug operations. If you wish to use the project with a different debug probe, then simply delete the existing launch configuration and allow a new one to be automatically used on the next debug operation.



Tip

To simplify this operation, a probe discovery can be forced by holding the *SHIFT* key while launching a debug session from the **Quickstart** panel. If the new debug connection is completed, a new project launch configuration will be created replacing any existing launch configurations. Alternatively, the Debug shortcuts [123] are available to force the use of a particular debug solution.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Tip
When exporting a project to share with others, launch configurations should usually be deleted before export (along with other IDE generated folders such as build configuration folders (Debug/Release if present)).

For further information please see the section Launch Configurations [118]

12.3 LinkServer Debug Connections

MCUXpresso IDE's native debug connection (known as LinkServer) supports debug operation through the following debug probes:

- MCU-Link and MCU-Link Pro with CMSIS-DAP firmware
- Evaluation boards incorporating MCU-Link with CMSIS-DAP firmware
- LPC-Link2 with CMSIS-DAP firmware
- LPCXpresso V2/V3 Boards incorporating LPC-Link2 with CMSIS-DAP firmware
- CMSIS-DAP firmware installed onto on-board debug probe hardware (as shipped by default on LPCXpresso MAX and CD boards)
  - For more information on LPCXpresso boards see: https://www.nxp.com/lpcxpresso-boards
  - Additional driver may be required:
    - https://developer.mbed.org/handbook/Windows-serial-configuration
- CMSIS-DAP firmware installed onto on-board OpenSDA debug probe hardware (as shipped by default on certain Kinetis FRDM and TWR boards)
  - Known as DAP-Link and mBed CMSIS-DAP: https://www.nxp.com/opensda
  - Additional driver may be required:
    - https://developer.mbed.org/handbook/Windows-serial-configuration
- Other CMSIS-DAP probes such as Keil uLINK with CMSIS-DAP firmware: https://www2.keil.com/mdk5/ulink
- Legacy RedProbe+ and LPC-Link
- RDB1768 development board built-in debug connector (RDB-Link)
- RDB4078 development board built-in debug connector

Note: MCUXpresso IDE will automatically try to softload the latest CMSIS-DAP firmware onto LPC-Link2 or LPCXpresso V2/V3 boards. For this to occur, the DFU link on these boards must be set correctly. Please refer to the board's documentation for details.

12.4 LinkServer Debug Operation

When the user first tries to debug a project, MCUXpresso IDE will perform a Debug Probe Discovery operation and present the user with a list of debug probes found.

Note: To perform a debug operation within MCUXpresso IDE, select the project to debug within the 'Project Explorer' view and then click Debug from the **Quickstart** View.

If more than one debug probe is presented, select the required probe. For LinkServer compatible debug probes, you can select from Non-Stop (the default) or All-Stop IDE debug mode.

Non-Stop uses GDB's "non-stop mode" and allows data to be read from the target while an application is running. Currently this mechanism is used to support the Live Global Variable [144] and Live Heap [150] features.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

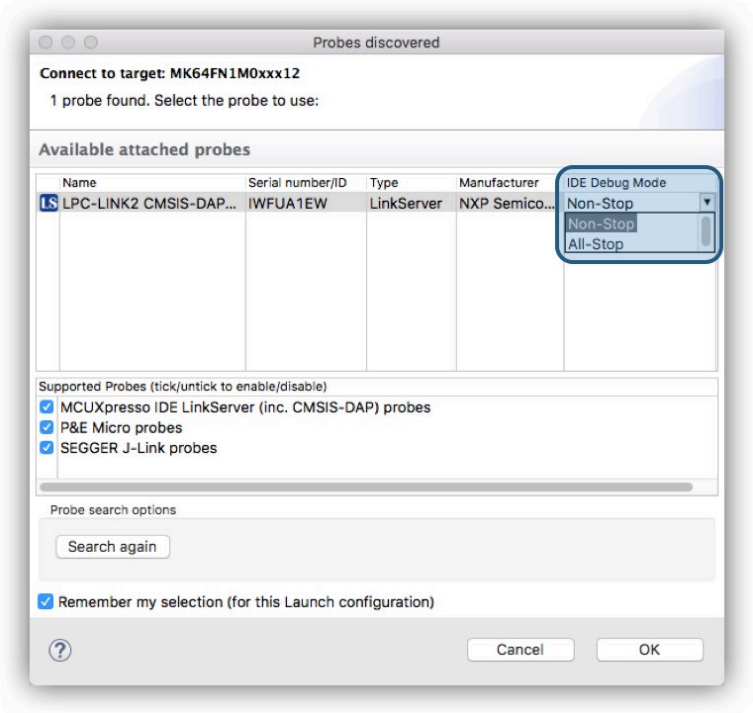


Figure 12.4. Debug Probe Discovery Non-Stop

Click ‘OK’ to start the debug session. At this point, the projects launch configuration files will be created. LinkServer Launch configuration files will contain the string ‘LinkServer’ and have an *LS* icon.

Note: If “Remember my selection” is left ticked, then the probe details will be stored within the launch configuration file, and this probe will be automatically selected on subsequent debug operations for this project.

For a description of some common debugging operations using supported debug probes see Common Debugging Operations [123]

MCUXpresso IDE defaults to the selection of “Non-Stop” mode when a LinkServer probe discovery operation is performed. This default can be changed from an MCUXpresso IDE Preference via:

Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options -> Miscellaneous

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

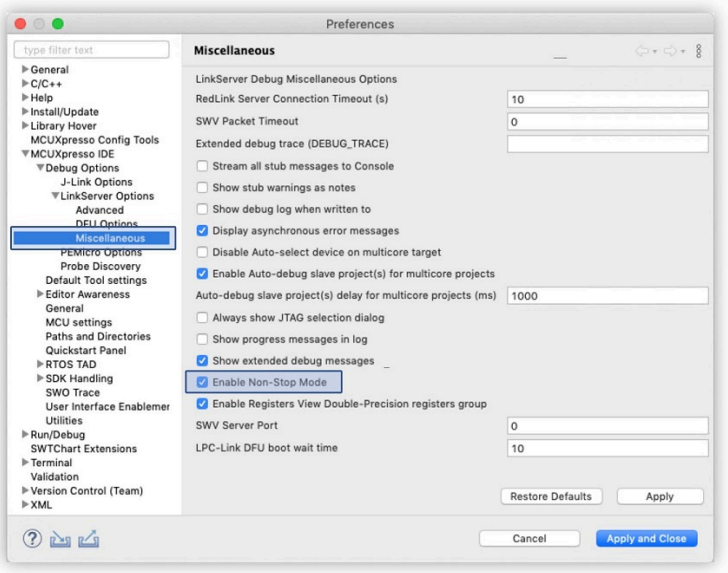


Figure 12.5. LinkServer Non Stop Preference

For a given project, the Non-Stop mode option is stored within the project's launch configuration. For projects that already have launch configurations, this option can be changed from the GDB Debugger tab as shown below.

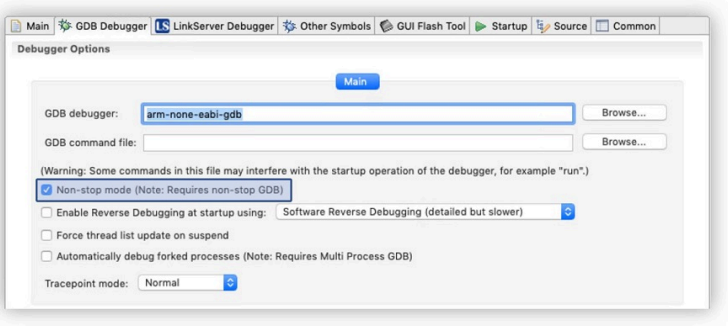


Figure 12.6. LinkServer Non Stop Control

12.4.1 LinkServer Debug Scripts

LinkServer debugging support a scripting language which is discussed in the section scripts [267].

A LinkServer debug connection has 3 potential *callouts* where scripts can be referenced typically to perform some non standard behaviour.

Connect Script a *Connect Script* overrides the default debug connection behaviour. Typically such scripts are used to prepare the debug target (MCU) for a debug operation that may otherwise fail due to some target setting that cannot be guaranteed post reset. A common requirement could be to ensure that RAM is available for Flash Programming operations. If required, a *Connect Script* is referenced within a LinkServer debug *Launch Configuration*.

Reset Script a *Reset Script* overrides the default debug reset behaviour. *Reset Scripts* are less commonly required than *Connect Scripts* but can be used to work around issues where a standard Reset may not allow debug operations to survive. If required, a *Reset Script* is referenced within a LinkServer debug *Launch Configuration*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

On rare occasions it may be useful to add a *Connect or Reset Script* to a project, see Project Sharing [45] for more information on how this can be done.

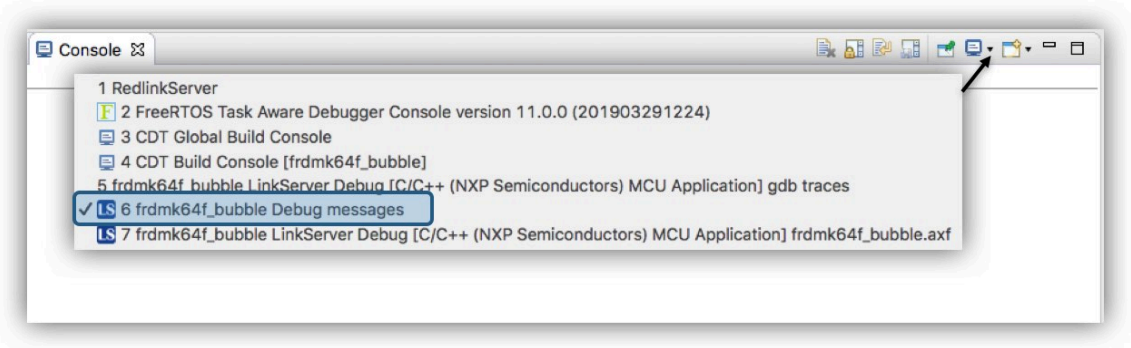
Preconnect Script a *Preconnect Script* is a little different. Such a scripts (if present) prepares the target MCU for an initial debug connection that may/would otherwise fail. *Preconnect Scripts* are not specified within a launch configuration, rather they are automatically invoked by the IDE for a given target based on built in intelligence. However, their use can be disabled by a checkbox within the project's *Launch Configuration*. On rare occasions it may be useful to add a *preconnect script* to a project – this can be done by placing a file called *LS_preconnect.scp* within the project's directory.

Note: In most circumstances, such scripts are supplied and referenced (via SDKs) automatically so no user intervention or action is required.

12.5 LinkServer Troubleshooting

12.5.1 Debug Log

On occasion, it can be useful to explore the operations of a debug session in more detail. The steps are logged into a console known as the Debug log. This log will be displayed when a Debug operation begins, but by default, will be replaced by another view when execution starts. The debug log is a standard log within the IDE's Console view. To display this log, select the Console and then click to view the various options (as below):



The debug log displays a large amount of information which can be useful in tracking down issues.

In the example debug log below, you can see that an initial Connect Script file has been run. Connect scripts are required for debugging certain parts and are automatically added to launch configuration files by the IDE if required. Next, the hardware features of the MCU are captured and displayed, this includes the number of breakpoints and watchpoints available along with details of various hardware components indicating what debug features might be available, for example Instruction Trace.

Further down in this log you will see the selection of a Flash driver (FTFE_4K), the identification of the part being debugged (in this case a K64), the programming progress and the speed of the Flash programming operation (in this case over 95 KB/sec).



Tip

a line similar to *flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)* will be displayed for LinkServer Flash programming operations. The size of the detected flash (in this example it is 1MB) and sector size (4KB) will be displayed here. The sector size may be important since multiples of this size represent valid base addresses for flash programming operations. For example, if the programming of more than one image is required, the second image must begin on a 4KB boundary beyond the end of any previously programmed image.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
MCUXpresso IDE RedlinkMulti Driver v11.1 (Nov 21 2019 14:13:54 - crt_emu_cm_redlink build 204)
Found part description in XML file MK64F12_internal.xml
Reconnected to existing LinkServer process.
============= SCRIPT: kinetisconnect.scp =============
Kinetis Connect Script
Connecting to Probe Index = 1
This probe = 1
This TAP = 0
This core = 0
DpID = 2BA01477
Assert NRESET
Reset pin state: 00
Power up Debug
MDM-AP APID: 0x001C0000
MDM-AP System Reset/Hold Reset/Debug Request
MDM-AP Control: 0x0000001C
MDM-AP Status (Flash Ready) : 0x00000032
Part is not secured
MDM-AP Control: 0x00000014
Release NRESET
Reset pin state: 01
MDM-AP Control (Debug Request): 0x00000004
MDM-AP Status: 0x0001003A
MDM-AP Core Halted
============= END SCRIPT ============================
Probe Firmware: LPC-LINK2 CMSIS-DAP V5.361 (NXP Semiconductors)
Serial Number:  IQCYI2IV
VID:PID:  1FC9:0090
USB Path: USB_1fc9_0090_314000_ff00
Using memory from core 0 after searching for a good core
debug interface type      = Cortex-M3/4 (DAP DP ID 2BA01477) over SWD TAP 0
processor type            = Cortex-M4 (CPU ID 00000C24) on DAP AP 0
number of h/w breakpoints = 6
number of flash patches   = 2
number of h/w watchpoints = 4
Probe(0): Connected&Reset. DpID: 2BA01477. CpuID: 00000C24. Info: <None>
Debug protocol: SWD. RTCK: Disabled. Vector catch: Disabled.
Content of CoreSight Debug ROM(s):
RBASE E00FF000: CID B105100D PID 04000BB4C4 ROM (type 0x1)
ROM 1 E000E000: CID B105E00D PID 04000BB00C Gen SCS (type 0x0)
ROM 1 E0001000: CID B105E00D PID 04003BB002 Gen DWT (type 0x0)
ROM 1 E0002000: CID B105E00D PID 04002BB003 Gen FPB (type 0x0)
ROM 1 E0000000: CID B105E00D PID 04003BB001 Gen ITM (type 0x0)
ROM 1 E0040000: CID B105900D PID 04000BB9A1 CSt TPIU type 0x11 Trace Sink - TPIU
ROM 1 E0041000: CID B105900D PID 04000BB925 CSt ETM type 0x13 Trace Source - Core
ROM 1 E0042000: CID B105900D PID 04003BB907 CSt ETB type 0x21 Trace Sink - ETB
ROM 1 E0043000: CID B105900D PID 04001BB908 CSt CSTF type 0x12 Trace Link - Trace funnel/router
NXP: MK64FN1M0xxx12
DAP stride is 4096 bytes (1024 words)
Inspected v.2 On chip Kinetis Flash memory module FTFE_4K.cfx
Image 'Kinetis SemiGeneric Nov  7 2019 19:12:49'
Opening flash driver FTFE_4K.cfx
Sending VECTRESET to run flash driver
Flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)
Closing flash driver FTFE_4K.cfx
Connected: was_reset=true. was_stopped=true
Awaiting telnet connection to port 3330 ...
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
GDB nonstop mode enabled
Opening flash driver FTFE_4K.cfx (already resident)
Sending VECTRESET to run flash driver
Flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)
Writing 26880 bytes to address 0x00000000 in Flash
00001000 done  15% (4096 out of 26880)
00002000 done  30% (8192 out of 26880)
00003000 done  45% (12288 out of 26880)
00004000 done  60% (16384 out of 26880)
00005000 done  76% (20480 out of 26880)
00006000 done  91% (24576 out of 26880)
00007000 done 100% (28672 out of 26880)
Sectors written: 7, unchanged: 0, total: 7
Erased/Wrote sector  0-6 with 26880 bytes in 276msec
Closing flash driver FTFE_4K.cfx
Flash Write Done
Flash Program Summary: 26880 bytes in 0.28 seconds (95.11 KB/sec)
Starting execution using system reset and halt target
Stopped (Was Reset)  [Reset from Unknown]
Stopped: Breakpoint #1
```

12.5.2 Flash Programming

Most debug sessions begin with the programming of Flash, followed by a reset of the MCU. **Note**: If flash programming should fail then the debug operation will be aborted.

Starting with MCUXpresso IDE version 11.1.0 – most LinkServer flash drivers now implement a *Verify Same* operation (via a flash hashing mechanism) for any flash sector that are unchanged from previous debug operations.

Below is a fragment of a debug log repeating the previous debug operation. The log reports the Sectors that were unchanged from the previous operation and the resultant overall speed of the flash operation – in this case the equivalent of a programming speed of 937 KB/sec.

```
...
Opening flash driver FTFE_4K.cfx (already resident)
Sending VECTRESET to run flash driver
Flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)
Writing 26880 bytes to address 0x00000000 in Flash
Sectors written: 0, unchanged: 7, total: 7
Erased/Wrote sector  0-6 with 26880 bytes in 28msec
Closing flash driver FTFE_4K.cfx
Flash Write Done
Flash Program Summary: 26880 bytes in 0.03 seconds (937.50 KB/sec)
Starting execution using system reset and halt target
Stopped (Was Reset)  [Reset from Unknown]
Stopped: Breakpoint #1
```

Note in the unlikely event of this feature causing problems, it can be disabled from a project LinkServer Launch Configuration by unchecking the *Enable Flash hashing* option. Alternatively, the feature can be disabled as a workspace preference via *MCUXpresso IDE -> Debug Options -> LinkServer Options -> Enable flash hashing*

Below is a brief discussion of the most common low level flash operations:

1. Sector Erase: internally Flash devices are divided into a number of sectors (or blocks), where a sector is the smallest size of Flash that can be erased in a single operation. A sector will be larger than a page (see below). Sectors are usually the same size for the whole Flash device,

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

however this is not always the case. A sector base address will be aligned on a boundary that is a multiple of its size. A sector erase is usually the first step in a flash programming sequence.

2. Page Program: internally Flash devices are divided into a number of pages, where a page is the smallest size that can be programmed in a single operation. A page will be smaller than a sector. A page base addresses will be aligned on a boundary that is a multiple of its size.
3. Mass Erase: a mass erase will reset all the bytes in Flash (usually to 0xff). Such an operation may clear any internal low level structuring such as protection of Flash areas (from programming).

The programming of an image (or data) comprises repeated operations of sector erase followed by a set of program page operations; until the sector is fully programmed or there is no more data to program.

One of the common problems when programming Kinetis parts relates to their use of Flash configuration block at offset 0x400. For more information please see: Kinetis MCUs Flash Configuration Block [216] . Flash sector sizes on Kinetis MCUs range from less than 1KB to 8KB, therefore the first Sector Erase performed may clear the value of this block to all 0xFFs, if this is not followed by a successful program operation and the part is reset, then it will likely report as ‘Secured’ and subsequent debugging will not be possible until the part is recovered.

Such an event can occur if a debug operation is accidentally performed to the ‘wrong board’ so a wrong Flash programmer is invoked.

Note: LinkServer mass erase operations will restore this Flash configuration block automatically for Kinetis parts. However, if a Kinetis device is mass erased by sector, this mechanism will be bypassed, therefore this operation should not be performed to Kinetis parts!

Should you need to recover a ‘locked’ part please see the section LinkServer GUI Flash Tool [167]

12.5.3 LinkServer executables

LinkServer debug operations rely on 3 main debug executables.

- **arm-none-eabi-gdb** – this is a version of GDB built to target ARM based MCUs
- **crt_emu_cm_redlink** – this executable (known as the debug stub) communicates with GDB through network sockets and passes low level commands to the LinkServer executable (also known as Redlink server)
- **redlinkserv** – this is the LinkServer executable and takes stub operations and communicates directly with the ARM Cortex debug hardware via the debug probe.

If a debug operation fails, or a crash occurs, it is possible that one or more of these processes may fail to shut down correctly. Therefore, if the IDE has no active debug connection but is experiencing problems making a new debug connection, ensure that none of these executables is running. To simplify this process an IDE button  is provided to kill all low level debug executables (for all debug solutions). Therefore should a debug operation fail or a crash occur, simply click this button before starting a new debug operation.

12.6 PEmicro Debug Connections

PEmicro software and drivers are automatically installed when MCUXpresso IDE installs. There is no need to perform any additional setup to use PEmicro debug connections.

Currently we have tested using:

- Multilink Universal (FX)
- Cyclone Universal (FX) (USB and Ethernet)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- PEmicro firmware installed into on-board OpenSDA debug probe hardware (as shipped by default on certain Kinetis FRDM and TWR boards)

Note: Some Kinetis boards ship with OpenSDA supporting PEmicro VCOM but with no debug support. To update this firmware visit the OpenSDA Firmware Update pages linked at: *Help -> Additional Resources -> OpenSDA Firmware Updates*

12.7 PEmicro Debug Operation

The process to debug via a PEmicro compatible debug probe is exactly the same as for a native LinkServer (CMSIS-DAP) compatible debug probe. Simply select the project via the 'Project Explorer' view then click Debug from the **Quickstart** panel and select the PEmicro debug probe from the Probe Discovery Dialogue.

If more than one debug probe is presented, select the required probe and then click 'OK' to start the debug session. At this point, the projects launch configuration files will be created. **Note**: PEmicro Launch configuration files will contain the string 'PE'.

MCUXpresso IDE stores the probe information, along with its serial number in the projects launch configuration. This mechanism is used to match any attached probe when an existing launcher configuration already exits.

To simplify debug operations, MCUXpresso IDE will automatically start PEmicro's GDB Server and select and dynamically assign the various ports needed as required. This means that multiple PEmicro debug connections can be started, terminated, restarted etc. all without the need for any user connection configuration. These options can be controlled if required by editing the PEmicro launch configuration file.

For more information see Common Debugging Operations [123]

Note: If the project already had a PEmicro launch configuration, this will be selected and used. If they are no longer appropriate for the intended connection, simply delete the files and allow new launch configuration files to be created.

Important Note: Low level debug operations via PEmicro debug probes are supported by PEmicro software. This includes, Part Support handling, Flash Programming, and many other features. If problems are encountered, PEmicro maintain a range of support forums at https://www.pemicro.com/forums/

Note: If a debug operation fails, or a crash occurs, it is possible that one or more debug processes may fail to shut down correctly. Therefore, if the IDE has no active debug connection but is experiencing problems making a new debug connection, ensure that none of these executables is running. To simplify this process an IDE button  is provided to kill all low level debug executables (for all debug solutions). Therefore should a debug operation fail or a crash occur, simply click this button before starting a new debug operation.

12.7.1 PEmicro Differences from LinkServer Debug

MCUXpresso IDE core technology is intended to provide a seamless environment for code development and debug.

When used with PEmicro debug probes, the debug environment is provided by the PEmicro debug server. This debug server does not 100% match the features provided by native LinkServer connections. However basic debug operations will be very similar to LinkServer debug.

For a description of some common debugging operations using supported debug probes see Common Debugging Operations [123]

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: LinkServer advanced features such as Power Measurement will not be available via a PEmicro debug connection. However, additional functionality maybe available using PEmicro supplied plugins.

12.7.2 PEmicro Software Updates

PEmicro support within MCUXpresso IDE is via an Eclipse plugin. The PEmicro update site is automatically added to the list of Available Software Update sites.

To check whether an update is available, please select:

Help -> Check for Updates

Any available updates from PEmicro will then be listed for selection and installation.

Note: PEmicro may provide news and additional information on their website, for details see https://www.pemicro.com

12.8 SEGGER Debug Connections

SEGGER J-Link software and documentation pack is installed automatically with the MCUXpresso IDE Installation for each host platform. No user setup is required to use the SEGGER debug solution within MCUXpresso IDE.

Currently we have tested using:

- J-Link debug probes (USB and Ethernet)
- J-Link firmware installed into on-board OpenSDA debug probe hardware (as shipped by default on certain Kinetis FRDM and TWR boards)
- J-Link firmware installed onto LPC-Link2 debug hardware and LPCXpresso V2/V3 boards
  - for details see https://www.segger.com/lpc-link-2.html
  - also for firmware programming see https://www.nxp.com/LPCSCRYPT

12.8.1 SEGGER software installation

Unlike other debug solutions supplied with MCUXpresso IDE, the SEGGER software installation is not integrated into the IDE installation, rather it is a separate SEGGER J-Link installation on your host.

The installation location will be similar to:

```
On Windows: C:/Program Files/SEGGER/JLink
On Mac: /Applications/SEGGER/JLink
On Linux: /opt/SEGGER/JLink
```

Note: SEGGER J-Link package is available in two flavors. The 64-bit version is currently installed and used by MCUXpresso IDE on all operating systems. The legacy 32-bit Windows package was used by older IDE versions but starting with MCUXpresso IDE v11.5.0, the 64-bit package is shipped. The installation folder for the 32-bit Windows version is usually:

```
On Windows: C:/Program Files (x86)/SEGGER/JLink
```

MCUXpresso IDE automatically locates the required executable and it is remembered as a Workspace preference. This can be viewed or edited within the MCUXpresso IDE preferences as below.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

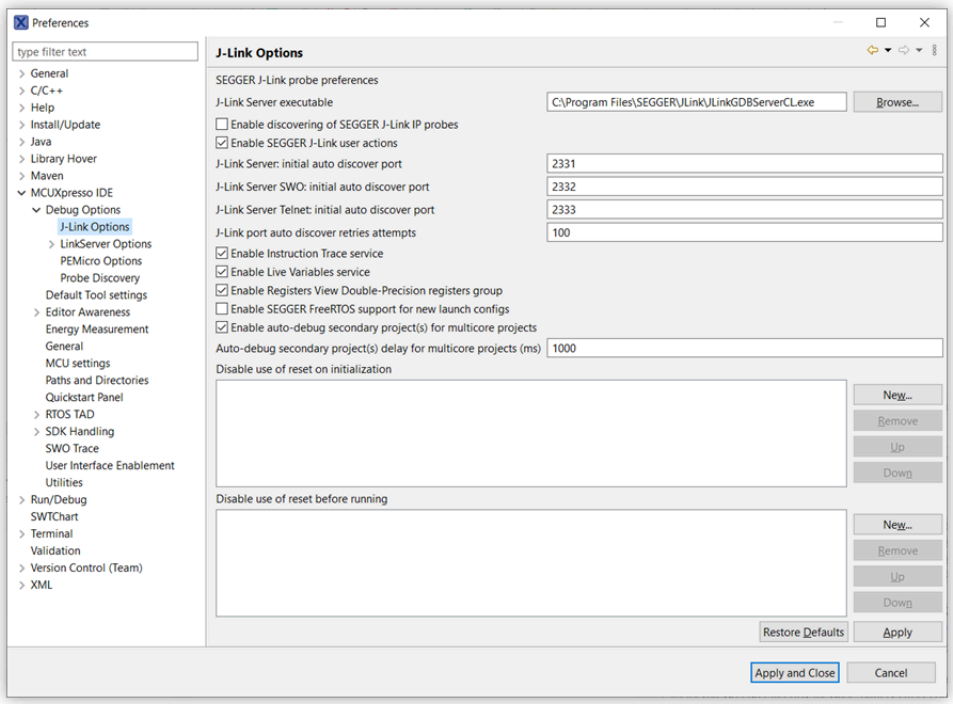


Figure 12.7. Segger Preferences

Note: this preference also provides the option to enable scanning for SEGGER IP probes (when a probe discovery operation is performed). By default, this option is disabled.

From time to time, SEGGER may release later versions of their software, which the user could choose to manually install. For details see https://www.segger.com/downloads/jlink

MCUXpresso IDE will continue to use the SEGGER installation path as referenced in a project's workspace unless the required executable cannot be found (for example, the referenced installation has been deleted). If this occurs:

1. The IDE will automatically search for the latest installation it can find. If this is successful, the Workspace preference will automatically be updated
2. If a SEGGER installation cannot be found, the user will be prompted to locate an installation

To force a particular workspace to update to use a newer installation location simply click the *Restore Default* button.

To permanently select a particular SEGGER installation version, the location of the SEGGER GDB Server can be stored in an environment variable.

For example, under Windows you could set:

```
MCUX_SEGGER_SERVER="C:/Program Files (x86)/SEGGER/JLink_V630k/jLinkGDBServerCL.exe"
```

This location will then be used, overriding any workspace preference that maybe set.

SEGGER software un-installation

If MCUXpresso IDE is uninstalled, it will not remove the SEGGER J-Link installation. If this is required, then the user must manually uninstall the SEGGER J-Link tools.

Note: If for any reason MCUXpresso IDE cannot locate the SEGGER J-Link software, then the IDE will prompt the user to either manually locate an installation or disable the further use of the SEGGER debug solution.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

12.9 SEGGER Debug Operation

The process to debug via a J-Link compatible debug probe is exactly the same as for a native LinkServer (CMSIS-DAP) compatible debug probe. Simply select the project via the 'Project Explorer' view then click Debug from the **Quickstart** Panel and select the SEGGER Probe from the Probe Discovery Dialogue.

If more than one debug probe is presented, select the required probe and then click 'OK' to start the debug session. At this point, the projects launch configuration files will be created. **Note**: SEGGER Launch configuration files will contain the string 'JLink'.

To simplify debug operations, MCUXpresso IDE will automatically start SEGGER's GDB Server and select and dynamically assign the various ports needed as required. This means that multiple SEGGER debug connections can be started, terminated, restarted etc. all without the need for any user connection configuration. These options can be controlled if required by editing the SEGGER launch configuration file.

In MCUXpresso IDE, SEGGER Debug operations default to using the SWD Target Interface. When debugging certain multicore parts such as the LPC43xx Series, the JTAG Target Interface must be used to access the internal Secondary MCUs. To select JTAG as the Target Interface, simply edit the SEGGER launch configuration file and select JTAG.

For more information see Common Debugging Operations [123]

Note: If the project already had a SEGGER launch configuration, this will be selected and used. If an existing launch configuration file is no longer appropriate for the intended connection, simply delete the files and allow new launch configuration files to be created.



Tip

If *Reset before running* is set in the Launch configuration, then a default intelligent reset will be used. This reset automatically supports running from Flash or RAM. A specific reset type can optionally be set from the free form text field if required, please consult SEGGER's documentation for available reset types.

Important Note: Low level debug operations via SEGGER debug probes are supported by SEGGER software. This includes, Part Support handling, Flash Programming, and many other features. If problems are encountered, SEGGER's provide a range of support forums at https://forum.segger.com/

12.9.1 SEGGER Differences from LinkServer Debug

MCUXpresso IDE core technology is intended to provide a seamless environment for code development and debug. When used with SEGGER debug probes, the debug environment is provided by the SEGGER debug server. This debug server does not 100% match the features provided by native LinkServer connections. However basic debug operations will be very similar to LinkServer debug.

For a description of some common debugging operations using supported debug probes see Common Debugging Operations [123]

Note: LinkServer features such as Power Measurement will not be available via a SEGGER debug connection. However, additional functionality maybe available using external SEGGER supplied applications.

12.10 SEGGER Troubleshooting

When a debug operation to a SEGGER debug probe is performed, the SEGGER GDB server is called with a set of arguments provided by the launch configuration file. The command and resulting output are logged within the IDE SEGGER Debug Console. The console can be viewed as below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Figure 12.8. Segger Server

The command can be copied and called independently of the IDE to start a debug session and explore connection issues.

Below is the shortened output of a successful debug session to a Kinetis K64 Board.

```
[05-1-2023 11:26:24] Executing Server: "C:\Program Files\SEGGER\JLink\JLinkGDBServerCL.exe"  /
-SettingsFile "C:\Users\MCUXpresso\Documents\MCUXpressoIDE_11.7.0\workspace\ /
frdmk64f_bubble_peripheral\Debug\frdmk64f_bubble_peripheral JLink Debug SettingsFile.jlink"  /
-nosilent -swoport 2332 -select USB=174505240  -telnetport 2333 -singlerun -endian little /
-noir -speed 4000   -port 2331    -vd -device MK64FN1M0xxx12 -if SWD -halt -reportuseraction  /
SEGGER J-Link GDB Server V7.84a Command Line Version

JLinkARM.dll V7.84a (DLL compiled Dec 22 2022 16:11:39)

Command line: -SettingsFile C:\Users\MCUXpresso\Documents\MCUXpressoIDE_11.7.0\workspace\  /
frdmk64f_bubble_peripheral\Debug\frdmk64f_bubble_peripheral JLink Debug SettingsFile.jlink /
-nosilent -swoport 2332 -select USB=174505240 -telnetport 2333 -singlerun -endian little   /
-noir -speed 4000 -port 2331 -vd -device MK64FN1M0xxx12 -if SWD -halt -reportuseraction
-----GDB Server start settings-----
GDBInit file:                  none
GDB Server Listening port:     2331
SWO raw output listening port: 2332
Terminal I/O port:             2333
Accept remote connection:      localhost only
Generate logfile:              off
Verify download:               on
Init regs on start:            off
Silent mode:                   off
Single run mode:               on
Target connection timeout:     0 ms
------J-Link related settings------
J-Link Host interface:         USB
J-Link script:                 none
J-Link settings file:          C:\Users\MCUXpresso\Documents\MCUXpressoIDE_11.7.0\workspace\ /
frdmk64f_bubble_peripheral\Debug\frdmk64f_bubble_peripheral JLink Debug SettingsFile.jlink
------Target related settings------
Target device:                 MK64FN1M0xxx12
Target device parameters:      none
Target interface:              SWD
Target interface speed:        4000kHz
Target endian:                 little

Connecting to J-Link...
J-Link is connected.
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
Device "MK64FN1M0XXX12" selected.
Firmware: J-Link Pro V4 compiled Sep 22 2022 15:00:37
Hardware: V4.00
S/N: 174505240
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.26 V
Listening on TCP/IP port 2331
Connecting to target...
InitTarget()
Found SW-DP with ID 0x2BA01477
DPIDR: 0x2BA01477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[2]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x24770011)
AP[1]: JTAG-AP (IDR: 0x001C0000)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
[0][0]: E000E000 CID B105E00D PID 000BB00C SCS-M7
[0][1]: E0001000 CID B105E00D PID 003BB002 DWT
[0][2]: E0002000 CID B105E00D PID 002BB003 FPB
[0][3]: E0000000 CID B105E00D PID 003BB001 ITM
[0][4]: E0040000 CID B105900D PID 000BB9A1 TPIU
[0][5]: E0041000 CID B105900D PID 000BB925 ETM
[0][6]: E0042000 CID B105900D PID 003BB907 ETB
[0][7]: E0043000 CID B105900D PID 001BB908 CSTF
Connected to target
Waiting for GDB connection...Connected to 127.0.0.1
Reading common registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, SP, /
LR, PC, XPSR
Connected to 127.0.0.1
Reading common registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, SP, /
LR, PC, XPSR
Read 4 bytes @ address 0x000047E6 (Data = 0x46BD3714)
Read 4 bytes @ address 0x000047E6 (Data = 0x46BD3714)
Read 4 bytes @ address 0x00000E14 (Data = 0x687B6078)
Reading 64 bytes @ address 0x2002FE40
Read 4 bytes @ address 0x00000E14 (Data = 0x687B6078)
Reading 64 bytes @ address 0x2002FE40
Received monitor command: reset
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
AfterResetTarget()
Resetting target
Downloading 16016 bytes @ address 0x00000000 - Verified OK
Downloading 8496 bytes @ address 0x00003E90 - Verified OK
Downloading 16 bytes @ address 0x00005FC0 - Verified OK
J-Link: Flash download: Bank 0 @ 0x00000000: Skipped. Contents already match
Writing register (PC = 0x     1d4)
Read 4 bytes @ address 0x000001D4 (Data = 0xF002B672)
Read 4 bytes @ address 0x000001D4 (Data = 0xF002B672)
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
Reading common registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, SP, /
LR, PC, XPSR
Read 4 bytes @ address 0x000001D4 (Data = 0xF002B672)
Reading 64 bytes @ address 0x00000F00
Read 2 bytes @ address 0x00000F22 (Data = 0xF107)
Received monitor command: semihosting enable
Semi-hosting enabled (Handle on breakpoint instruction hit)
Received monitor command: exec SetRestartOnClose=1
Executed SetRestartOnClose=1
Received monitor command: reset
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
AfterResetTarget()
Resetting target
Setting breakpoint @ address 0x00000F22, Kind = 2, Type = THUMB, BPHandle = 0x0001
Starting target CPU...
...Breakpoint reached @ address 0x00000F22
Reading common registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, SP, /
LR, PC, XPSR
Removing breakpoint @ address 0x00000F22, Size = 2
Read 4 bytes @ address 0x00000F22 (Data = 0x031CF107)
Reading 64 bytes @ address 0x2002FFC0
Read 4 bytes @ address 0xE00FFFF4 (Data = 0x00000010)
...
```

Note: If a SEGGER debug operation is not successful, the IDE will generate an error dialogue, the *'Details'* button can be clicked to display a copy of the SEGGER server log. One possible reason for a SEGGER debug operation to fail is due to a Device name mismatch. MCUXpresso IDE will try to supply the expected Device name to SEGGER server, however on rare occasions this may not be the name expected. The SEGGER launch configuration Device entry can be populated via a drop down list or via a user supplied device name.

If required, additional server options can be set within the SEGGER launch configuration. For example to capture logging information to a file, you can set the additional server option:

```
-log $(CWD)/my.log
```

where *$(CWD)* represents the current working directory of the debug connection, i.e. the dynamically created project build configuration folder.

Note: If a debug operation fails, or a crash occurs, it is possible that one or more debug processes may fail to shut down correctly. Therefore, if the IDE has no active debug connection but is experiencing problems making a new debug connection, ensure that none of these executables is running. To simplify this process an IDE button  is provided to kill all low level debug executables (for all debug solutions). Therefore should a debug operation fail or a crash occur, simply click this button before starting a new debug operation.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

13. Debugging a Project

This chapter describes many of the common debug features supported by the debug solutions within MCUXpresso IDE. Please also refer to the chapter Debug Solutions Overview [94] for more details of the supported debug solutions and management of debug operations.

13.1 Debugging Overview

A debug operation requires a physical connection between the host computer and the target MCU via a debug probe. The debug probe translates the high level commands provided by MCUXpresso IDE into the appropriate low level operations supported on the target MCU.

This connection to the debug probe is usually made via USB to the host computer (although IP probes from PEmicro and SEGGER are also supported). Some debug probes such as LPC-Link2 or SEGGER J-Link *Plus* are separate physical devices, however many LPCXpresso, Freedom, Tower, EVK boards also incorporate a built in debug probe accessed by one of the development boards USB connections.

Note: If a separate debug probe is used, you must ensure that the appropriate cables are used to connect the debug probe to the target board, and that the target is *correctly* powered.

Typically, an on board debug probe connection will also provide power to the development board and target MCU. In contrast, an external debug probe will not usually power the target, and a second connection (often USB) will be required to provide power to the board and MCU. Some external debug probes such as the LPC-Link2 can also provide power to the target board – this is enabled by connecting the link *JP2*. For other debug probes, refer to their supplied documentation.

External debug probes will usually provide superior features and performance compared to on-board debug probes, however please note that LPCXpresso V2 and V3 boards incorporate a full featured LPC-Link2 debug probe.

Note: Some LPCXpresso development boards have two USB connectors fitted. Make sure that you have connected the lower connector marked DFU-Link. Many Freedom and Tower boards also have two USB connectors fitted. Make sure that you have connected to the one marked ‘OpenSDA’ - this is usually (but not always) marked on the board. If in doubt, the debug processor used on these designs is usually a Kinetis K20 MCU, it is approximately 6mm square. The USB nearest this MCU will be the OpenSDA connection.

13.1.1 Debug Launch

To debug a project on your target MCU, simply highlight the appropriate project in the ‘Project Explorer’, and then in the **Quickstart** Panel click on the large **Debug**, as in Figure 13.1, alternatively click the blue bug icon  to perform the same action.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

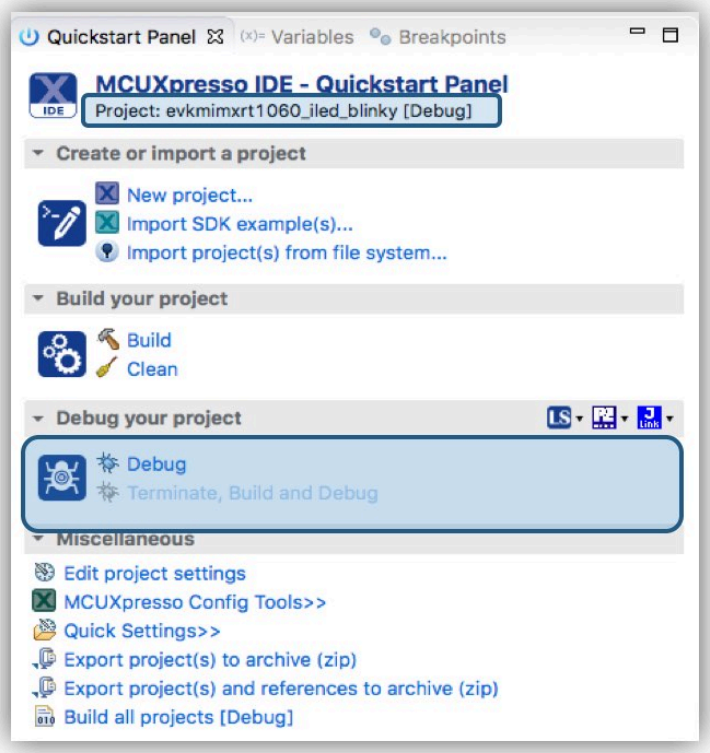


Figure 13.1. Launching a debug session

Note: The green bug icon should not be used because this invokes the standard Eclipse debug operation and so skips certain essential MCUXpresso IDE debug steps.

For a newly created project a debug operation will perform a number of steps. By default, it will first build the selected project and (assuming there are no build errors) launch a debug probe discovery operation (see next section) to allow the user to select the required debug probe. A launch configuration file will automatically be created with default options (per build configuration) and will be associated with the project. Like a project's build configuration, launch configuration files control what occurs each time a debug operation is performed. Please see the section An Introduction to Launch Configuration files [95] for more information.

Note: This default behaviour can be changed by editing the Workspace preference located at *Preferences -> Run/Debug -> Launching -> Build (if required) before launching*. For individual projects, the *Main* tab of the launch configuration allows the workspace preference to be overridden.

By default, once a debug probe has been selected (and 'OK' clicked) the binary contents of the .axf file will automatically be downloaded to the target via the debug probe connection. Typically, projects are built to target MCU Flash memory, and in these cases, a suitable Flash driver will automatically be selected to perform the Flash programming operation. Next a default breakpoint will be set on the first instruction in `main()`, the application will be started (by performing or simulating a processor reset), and code will be executed until the default breakpoint is hit. See the section on Breakpoints [131] for additional information.

13.1.2 Debug Probe Selection Dialog (Probes Discovered)

The first time you debug a project, the IDE will perform a probe discovery operation and display the discovered Debug Probes for selection. This will show a dialogue listing all supported probes that are attached to the host computer. In the example shown in Figure 13.2, a LinkServer (LPC-Link2), a PEmicro Multilink and also a J-Link (OpenSDA) probe have been found.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

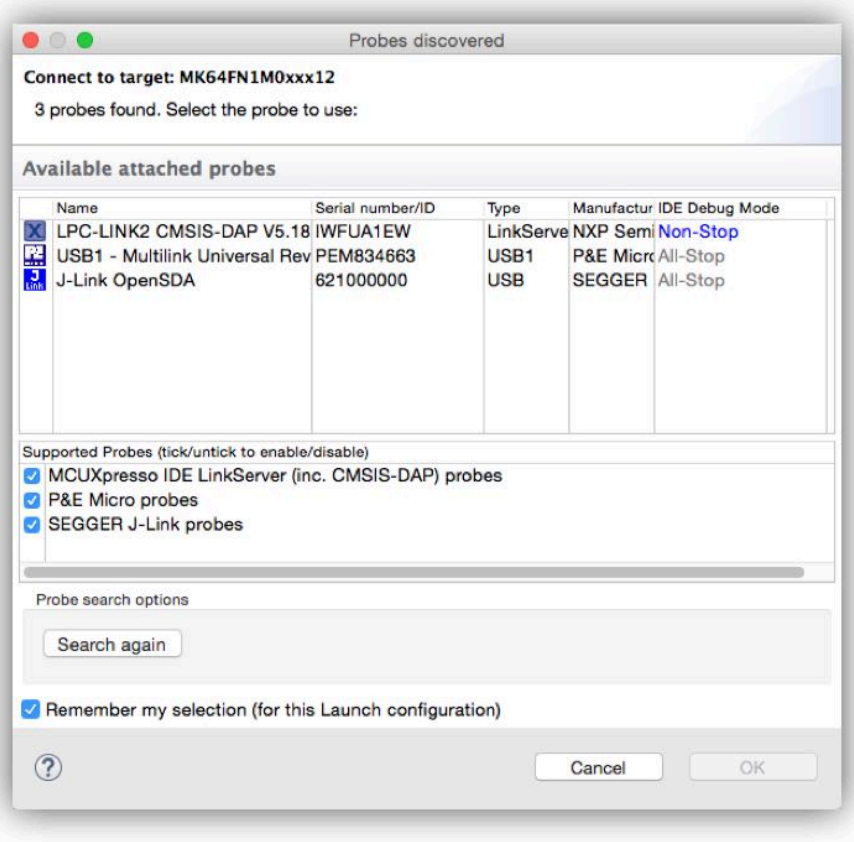


Figure 13.2. Attached probes: debug emulator selection

Note: if only one probe is found, it will be selected automatically, so simply click OK or hit return to use the probe displayed.

MCUXpresso IDE supports unique debug probe association.

Debug probes can return an ID (Serial number) that is used to associate a particular debug probe with a particular project. Some debug probes will always return the same ID, however debug probes such as the LPC-Link2 will return a unique ID for each probe – in our example **IWFUA1EW**.

For any future debug sessions, the stored probe selection will be automatically used to match the project being debugged with the previously used debug probe. This greatly simplifies the case where multiple debug probes are being used.

However, if a debug operation is performed and the previously remembered debug probe cannot be found, then a debug probe discovery operation will be performed from within the same family e.g. LinkServer, PEmicro or SEGGER.

See also debug shortcuts [123]

Sometimes a probe discovery will find no debug probes and return a dialogue as below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

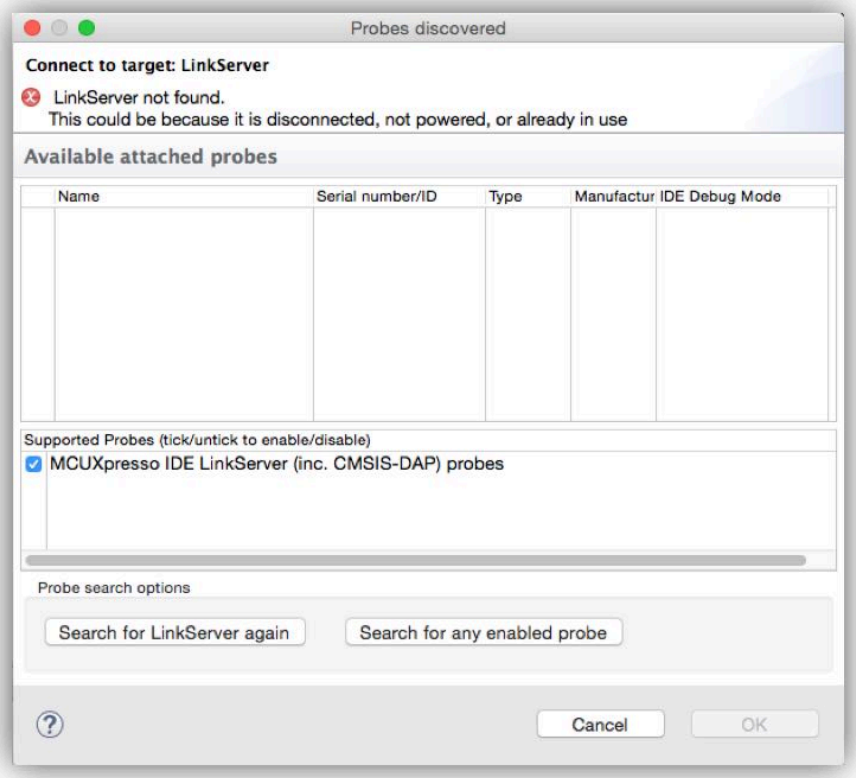


Figure 13.3. LPC-Link2 no longer connected

This might have been because you had forgotten to connect the probe, in which case simply connect it to your computer and select **Search again**. If you are using a different debug probe from the same family of debug probes, simply select the new probe and this will replace the previously selected probe.

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).
- If you wish to debug a project using a different family of debug probe(s), then the simplest option is to delete the launch configuration files associated with the project and start a debug operation. Please see the section "An Introduction to Launch Configuration files [95] for more information. Please also see Debug Shortcuts. [123]

Firmware version check on MCU-Link / MCU-Link Pro probes

For MCU-Link and MCU-Link Pro probes, the **Probes Discovered** will provide an indication if a newer firmware version is available.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

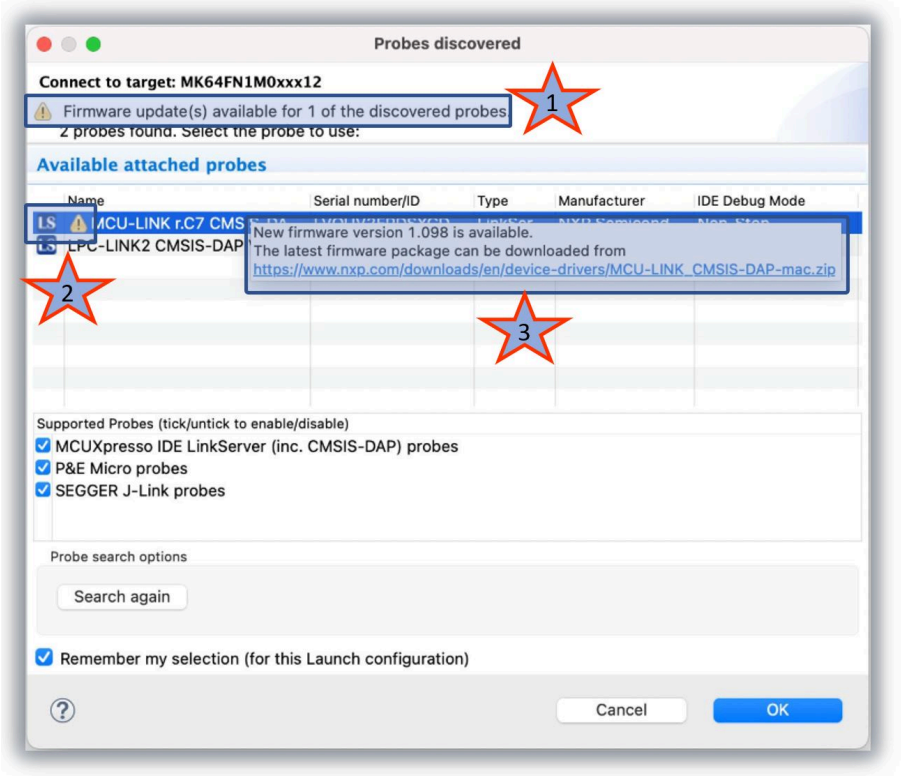


Figure 13.4. MCU-Link available firmware update indication

1. Once the Probes Discovered opens, a warning will indicate if there are debugger probes that would require firmware update.
2. Each discovered probe will warn if a firmware update is available.
3. A tooltip (on each probe marked by a warning icon) will specify the firmware's download location and its version.

Note For details on how to update the MCU-Link firmware, follow the link from https://www.nxp.com/design/microcontrollers-developer-resources/mcu-link-debug-probe:MCU-LINK and https://www.nxp.com/design/microcontrollers-developer-resources/mcu-link-pro-debug-probe:MCU-LINK-PRO.

13.1.3 Controlling Execution

When you have started a debug session a default breakpoint [131] is set on the first instruction in `main()`, the application is started (by simulating or performing 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 13.1, which are displayed on the global toolbar. The call stack is shown in the Debug View, as in Figure 13.5.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

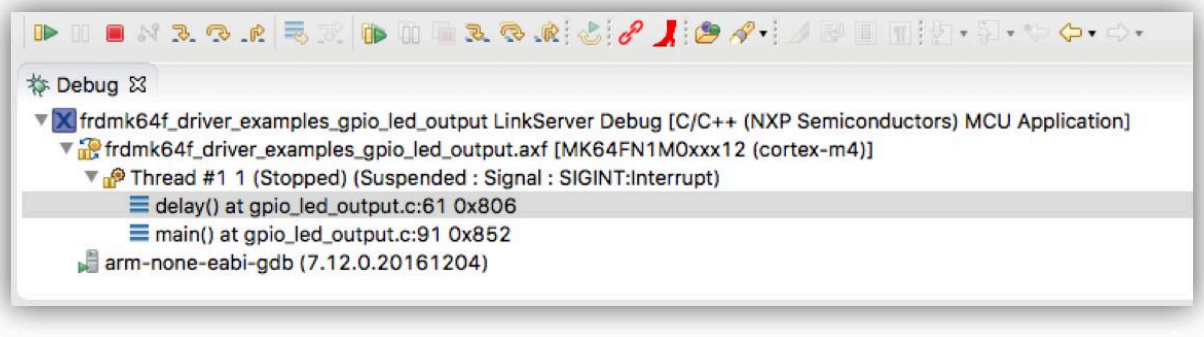


Figure 13.5. Debug controls and Debug Call Stack

Table 13.1. Program execution controls

Button	Description	Keyboard Shortcut
	Restart program execution (from reset)	
	Run/Resume the program	F8
	Pause Execution of the running program	
	Terminate the debug Session	Ctrl + F2
	Clean up debug	
	Run, Pause, Terminate all debug sessions	
	Step over a C/C++ line	F6
	Step into a function	F5
	Return from a function	F7
	Step in, over, out all debug sessions	
	Show disassembled instructions	



Tip

Clean up debug will kill all debug processes associated with LinkServer, PEmicro and SEGGER debug connections. This may be necessary if the IDE is restarted with a connected debug session or if a crash occurs – and will remove any failed or orphaned debug processes. **Note**: a warning will be issued with the option to cancel before any action is performed since this will kill all connected debug sessions.

Note: The debug controls for 'all' debug sessions will perform identically to their single session counterparts if only one debug session exists.

Note: Typically a user will only have a single active debug session. However if there is more than one debug session, the active session can be chosen by clicking within the debug call stack within the Debug view. All debug views will reflect the selected session.

Setting a breakpoint

To set a breakpoint, simply double-click on the left 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.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Stopping debugging

To stop debugging just press the **Terminate/Stop** button. This action will disconnect MCUXpresso IDE from the target (board). The subsequent behaviour is controllable by the disconnect behaviour. [129]

Pause debugging

Typically, debugging is paused due to the action of a breakpoint [131] or watchpoint [133] since these will be set to observe the target when an event of interest has occurred. However, the pause button can be used to pause the target at an instant of time.

To pause debugging

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 MCUXpresso IDE (as shown in Figure 3.4).

13.2 Launch Configurations

Launch Configuration files will be automatically created within the root directory of a project the first time a debug operation is performed. They will typically be named:

```
{projname}{debug solution}Debug.launch
{projname}{debug solution}Release.launch
```

A file will be created for the build variant being debugged, and is used to store the settings for the debug connection for that build configuration.

Normally, there is no need to edit launch configurations, as the default settings created by the IDE will be suitable. However, in some circumstances, you may need to manage them – typically under direction from an FAQ. In such cases this can be done via the “Launch Configurations” entry on the context sensitive menu available from the Project Explorer view...

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

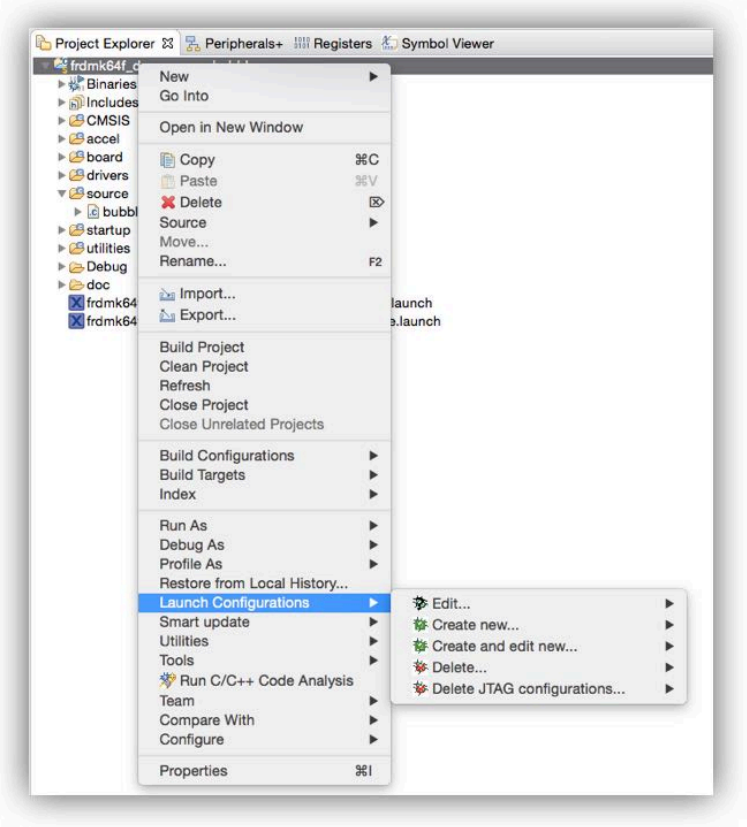


Figure 13.6. Create a Launch Configuration

Note: to view the contents or edit an existing launch configuration file, you can also simply double click to open an edit view.

A number of options are available here:

Edit...

- Allows various debug settings to be modified
  - Typically not required since the default options will be correct for most debug operations

Create new...

- Create a launch configuration for a particular debug solution, if they do not already exist.
  - Normally you will not need this option as it is carried out automatically the first time that you debug your project. However, if you want the flexibility to debug a project with different debug solutions for example, LinkServer and SEGGER, then both sets of launch configurations can be created. On the next debug operation, the user can select the launch configuration to use for that session.

Create and edit new...

- Allows new launch configurations to be created and immediately opened for editing.

Delete...

- Allows the launch configurations for the selected project (or projects) to be deleted.
- This can be useful as it allows you to put the debug connection settings back to the default after making modifications for some reason, or if you are moving your project to a new version of the tools, and want to ensure that your debug settings are correct for this version of the tools.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Delete JTAG Configuration...

- Allows the JTAG configuration files for the selected project (or projects) to be deleted. These files are stored in the Debug/Release subdirectories.

13.2.1 Editing a Launch Configuration (LinkServer)

WARNING: - Modifying the default settings for a launch configuration can prevent a successful debug connection from being made.

After selecting the “Edit...” or “Create and edit New” launch configuration menu entry, you will then see a new dialog box pop up, which looks similar to the following...

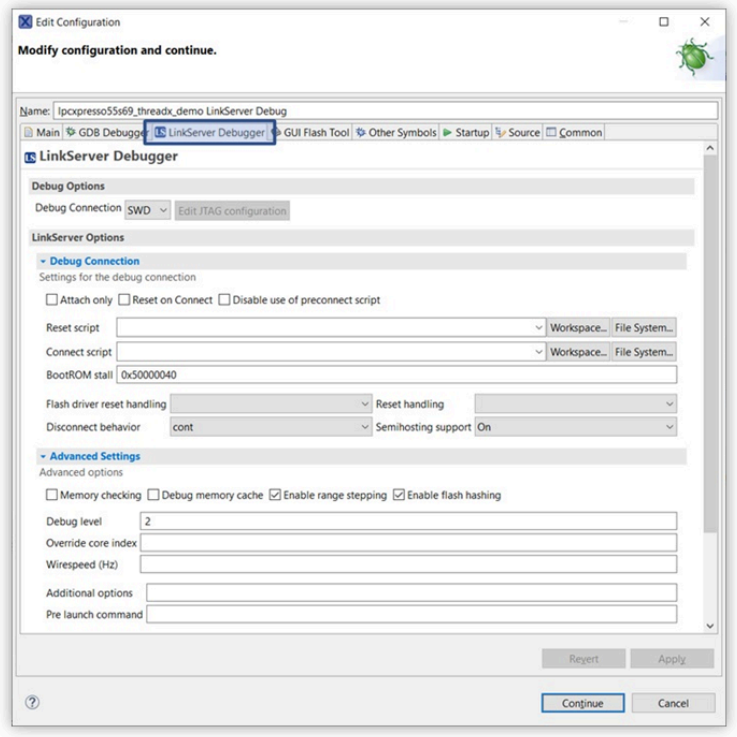


Figure 13.7. Edit a Launch Configuration

Most settings that you may need to modify can be found in the Debugger tab, in the Target configuration sub-tab (as shown in the above screenshot).

Some examples of modifications that you may need to make in particular circumstances are:

- Changing the initial breakpoint [131] on debug startup
  - When the debugger starts, it automatically sets an initial (temporary) breakpoint on the first statement in main(). If desired, you can change where this initial breakpoint is set, or even remove it completely.
- Modifying the Debugger connect behaviour
  - via a Connect Script e.g. kinetisconnect.scp
- Modifying the Debugger reset behaviour.
  - *Flash driver reset handling* is used to run the RAM-loaded flash driver, while *Reset handling* is used to start the image loaded into flash/RAM. Possible choices:
    - *SOFT*: Maintain current software environment but change the SP and PC. Note that the values for SP and PC are read from the first two words of the binary image, so *SOFT* is not applicable in case of flash images which begin with a bootheader / configuration block.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- *VECTRESET*: Execute a hardware reset of the core & catch the vectored ‘reset’ event. ARMv8-M cores lack this reset mechanism, so *SOFT* reset will be used instead.
- *SYSRESETREQ*: Execute a hardware restart of the system & catch the vectored ‘reset’ event.
- *Default*: For *Flash driver reset handling* it usually means *VECTRESET*, except for specific parts. For *Reset handling* it means *SYSRESETREQ* for flash and *VECTRESET* for RAM.

- If the *Reset script* input field is not empty, the specified reset script is executed instead of the selected *Reset handling*.
- *BootROM stall*: Temporarily stall the boot loader post initialization on a read watchpoint to restablish debug control. Useful for ARMv8-M cores which lack vector catch.

- Connecting to a target via JTAG rather than SWD
  - if supported by the target, you can edit the Debug type
- Connecting to a running target
  - set Attach only to True (see also debug shortcuts) [123]
- Changing the debug stub connection parameters
  - By default, the IDE attempts to start the GDB stub when initiating a debug connection. As result, the launch configuration is automatically created with the “Automatically start debug server” checkbox enabled. If the GDB stub is already running, and the GDB client must connect to that instance, it must be unchecked.
  - Controls associated with the host name and with the network port number that will be used for listening to GDB client connections are grayed-out by default. These parameters can be changed by disabling “Automatically start debug server”.

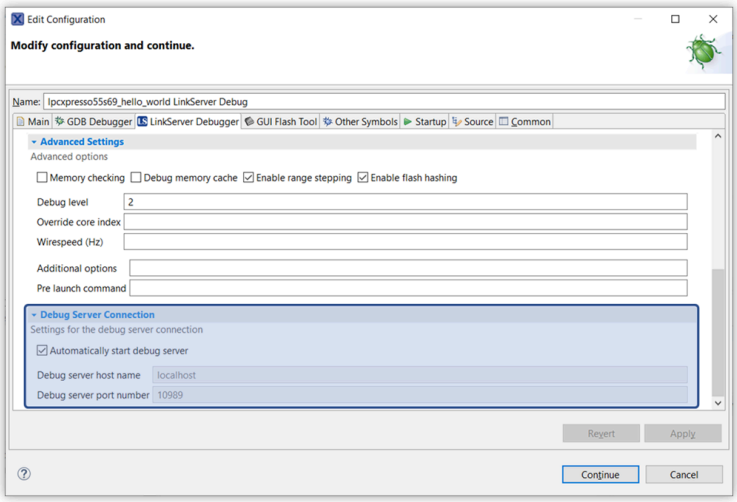


Figure 13.8. Debug Server Connection

Note that the Debug Server Connection parameters can be also changed using the Preferences page. All launch configurations will be created using the values specified there. Existing launch configurations will not be affected by any change in the Preferences page. Go to **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options** in order to change any of the debug server-related parameters.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

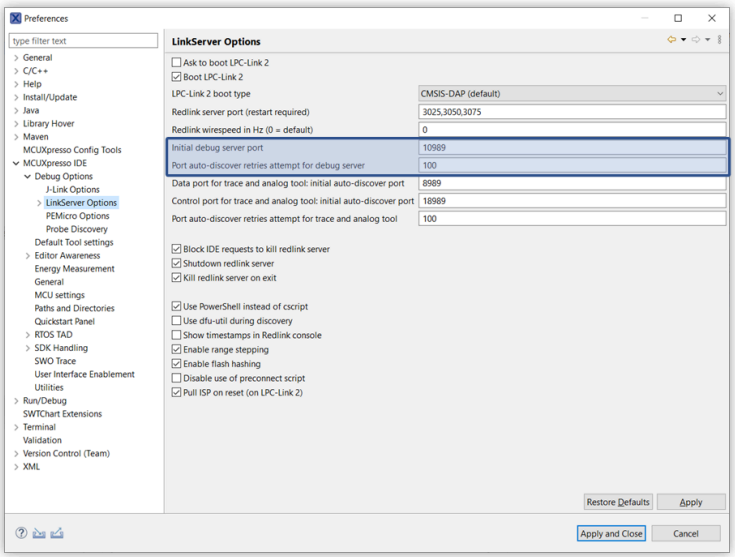


Figure 13.9. Debug Server Connection

Target Boot Configuration

- Changing target boot configuration
  - LinkServer debug configuration has a section for Target Boot Control (highlighted below):

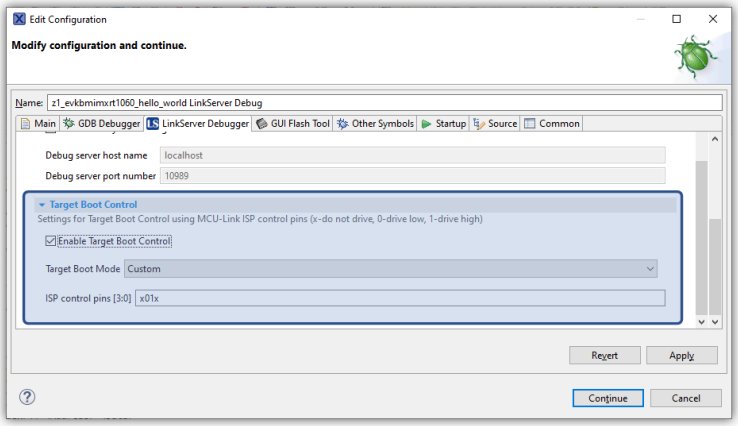


Figure 13.10. Target Boot Control

- The main purpose of this feature is to configure how the device boots on the reset requests issued during the debug session. This feature requires an **MCU-Link debug probe** with the ISPx boot control features implemented.
- By default, the target boot control is disabled when the launch configuration is created. If a specific target boot configuration is required, you need to enable the target boot control and change the value present in the edit box and/or select the boot mode from the list. At this moment, there are no predefined target boot mode entries in the list. The combo box has two generic entries:
  - **Custom** entry: at least one ISP control pin is configured to be driven (custom configuration)
  - Empty entry: no ISP control pin is driven (default).
- When the target boot control is enabled, the selected configuration is stored on the probe and a wire reset is issued to apply it at the very beginning of the debug session. Also, it overrides the **Pull ISP on reset** option from **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options**.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- **Note**: The on-board MCU-Link probes have support for up to 4 ISP control pins, being able to drive up to 4 target device pins, but this depends on how the target board is designed. MCU-Link Pro and the MCU-Link base probes have one ISP control pin (ISP0).
- **WARNING**: Depending on the selected target boot mode (ISP control pins state), debug session failures might occur in cases where code exists in the source of the boot media (e.g. in flash) and that code prevents the debugger from gaining control.



Tip

Multiple launch configurations are supported for each build configuration. Multiple launch configurations may be created using standard Eclipse functionality – for example from the main menus, select *Run -> Debug Configurations* and double click on the C/C+ entry. Alternatively you can clone an existing launch configuration. Once this has been done, a debug operation will present the user will a list of available launch configurations. Simply double click the required launch configuration to start the debug session.

13.3 Common Debug Operations and Launch Configurations

Where possible MCUXpresso IDE attempts to provide a common debug experience regardless of the debug solution being used. However some debug tasks require launch configuration modifications and these will be different for each debug solution. In this section, some common debug operations are discussed for each debug solution.

13.3.1 Debug Quickstart Shortcuts

MCUXpresso IDE Quickstart panel incorporates Debug shortcut buttons. These buttons invoke actions **only** from their respective debug solutions.

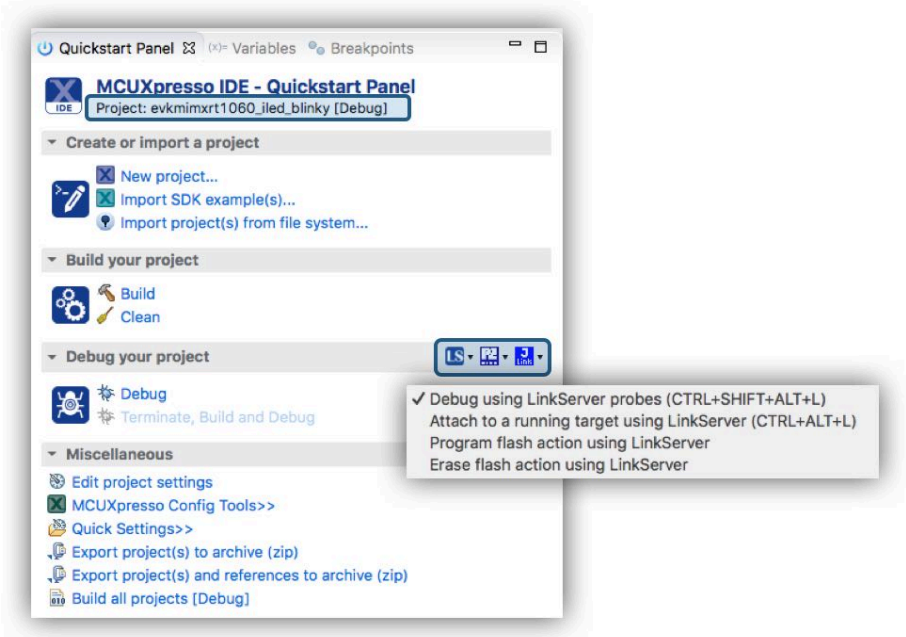


Figure 13.11. Debug Shortcuts (LinkServer shown)

Each button provides the same 4 options for each debug solution:

Debug (default) : make a **Debug** connection to the chosen debug probe. A launch configuration will be created if not present. The attach mode will be set to **False**. **Note**: a normal debug

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

operation will inherit a launch configuration attach setting, whereas this operation will force attach mode to False. If a launch configuration already exists, its attach setting will be set to **False**, no other changes will be made.

Attach : make an **Attach** connection to a LinkServer compatible debug probe. A launch configuration will be created if not present. The attach mode will be set to **True**. The launch configuration will be given a *A* decorator to show that Attach is the set configuration.  button. If a launch configuration already exists, its attach setting will be set to **True**, no other changes will be made.

Program Flash : perform the launch configuration *Program action*, by default this will program the 'project' into flash. The selected project will be built if required and a default launch configuration will be created if one is not present.

Erase Flash : perform the launch configuration *Erase action*, by default this will erase the flash memory via a mass erase. A default launch configuration will be created if one is not present.

Note: the selected action will be remembered for subsequent shortcut uses, the tooltip will show the action to be performed.



Tip

If an attach operation is performed, the created launch configuration will have Attach set to True. Therefore any subsequent debug operations will be in Attach Mode, until either the launch configuration is edited to set Attach to false, or the Debug short cut is used again to force the attach mode to false.

13.3.2 Connecting to a running Target (attach)

A typical debug session will begin by downloading code to Flash and then debugging from main() onwards. However, to explore an already running system a debug connection (attach) can be made to the target MCU without affecting the code execution (at least until the user chooses to halt the MCU!).

Note: Source level debug of a running target is only possible if the sources of the project to be attached exactly match the binary code running on the target.

Important Note: Please be sure to read and understand the section on semihosted printf and debugging [192] and also the implications in the related section on library selection [188]

LinkServer

Edit the project launch configuration by double clicking on the launch config file, select the Debugger tab and Target configuration view, then set the 'Attach only' setting to True as below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

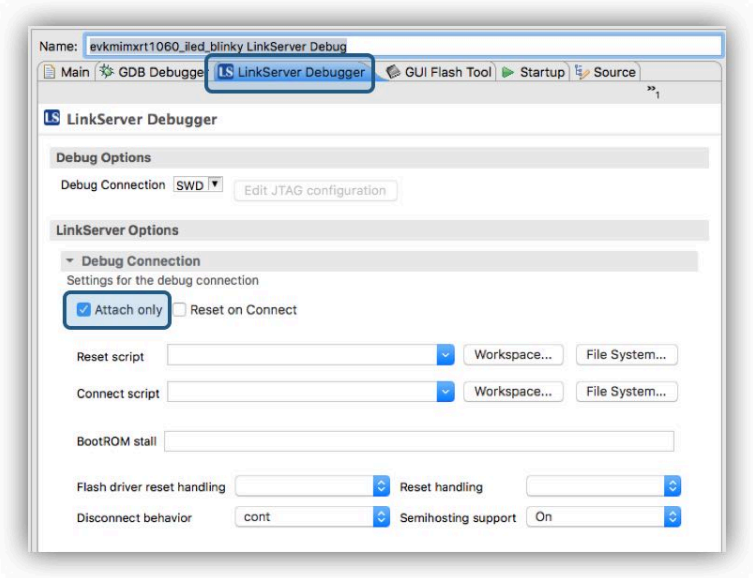


Figure 13.12. Debug Launch Attach Mode

When a debug connection is made, the target will continue running until it is paused. However, if the IDE Debug Mode is set to Non-Stop (the default) then Global variables values can be explored and displayed.

Other operations such as ITM console IO will also function. See the LinkServer SWO Trace Guide for further information.

PEmicro

Edit the project launch configuration by double clicking on the launch config file, select the Startup tab, then set the ‘Attach to a running target’ check box as below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

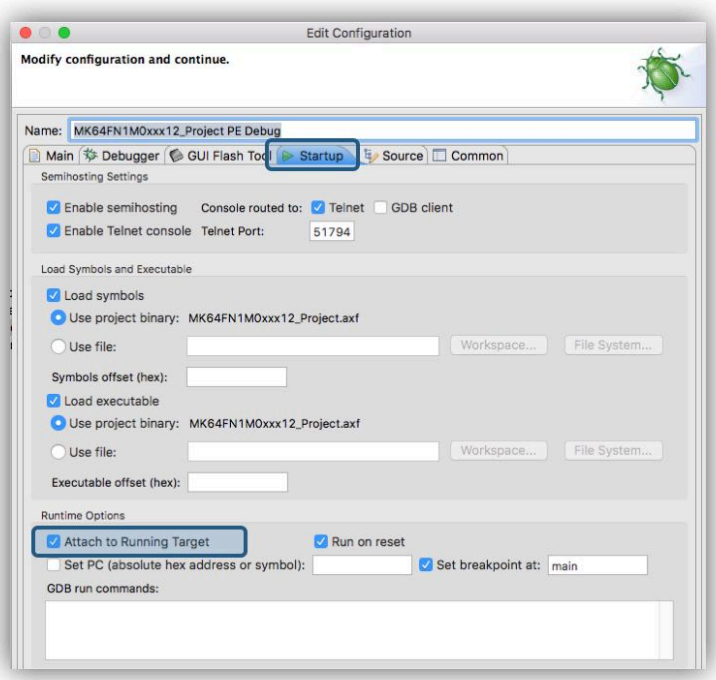


Figure 13.13. Debug Launch Attach mode PEmicro

When a debug connection is made, the target will continue running until it is paused.

SEGGER JLink

Edit the project launch configuration by double clicking on the launch config file, select the Debugger tab, then set the ‘Attach to a running target’ check box as below:

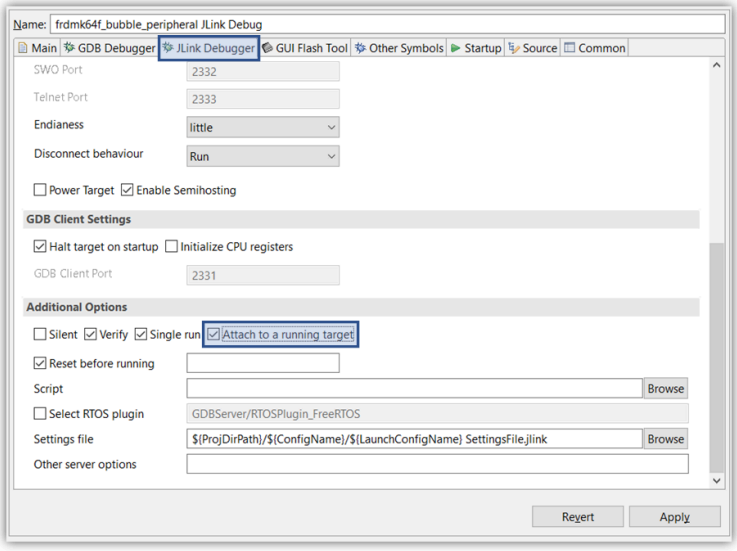


Figure 13.14. Debug Launch Attach Segger

When a debug connection is made, the target will continue running until it is paused.

13.3.3 Controlling the initial Breakpoint (on main)

When the debugger starts, it automatically sets an initial (temporary) breakpoint on the first statement in main(). If desired, you can change where this initial breakpoint is set, or even remove

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

it completely. One common requirement is to debug an application from startup. The entry point (startup) in a standard example application can be identified by a symbol called ResetISR, a breakpoint can be set on this symbol to halt execution at the first instruction within an application.

LinkServer

To debug from the start of the image, edit the project launch configuration by double clicking on the launch config file, select the Debugger tab, replace main with ResetISR

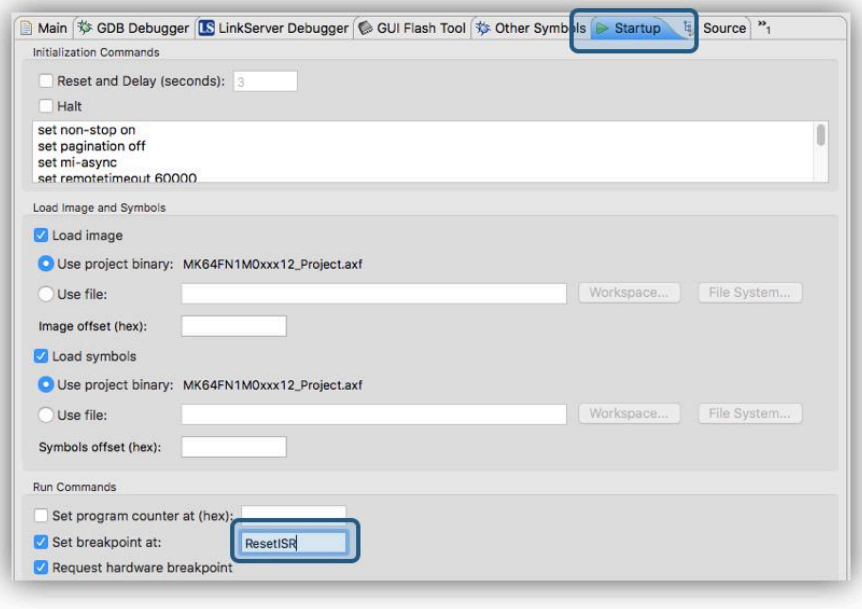


Figure 13.15. Debug Launch ResetISR

When a debug connection is made, the target should halt at this symbol.

To disable the initial breakpoint, uncheck the option ‘Stop on startup at...’. To restore the original behaviour, replace the symbol ResetISR with main, and check the option ‘Stop on startup at...’. Alternatively, you could delete the launch configuration and allow the IDE to create a new one.

PEmicro

Edit the project launch configuration by double clicking on the launch config file, select the Startup tab, replace main with ResetISR

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

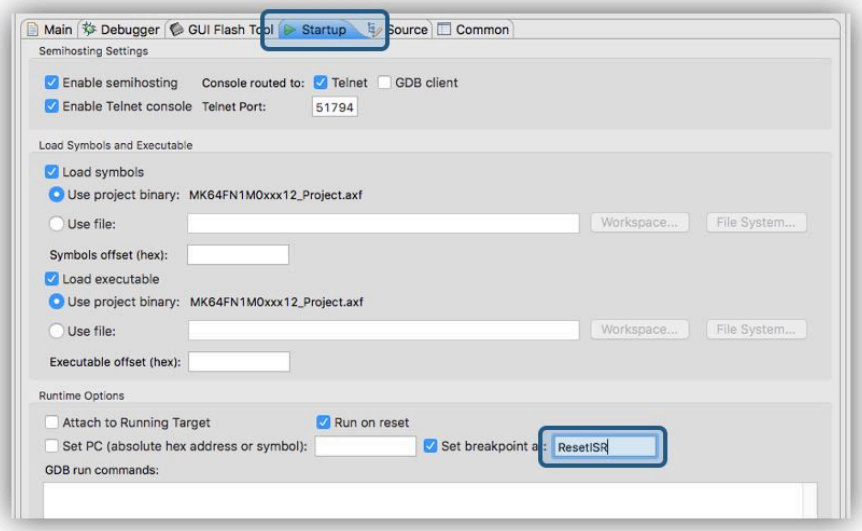


Figure 13.16. Debug Launch ResetISR PEmicro

When a debug connection is made, the target should halt at this symbol.

To disable the initial breakpoint, uncheck the option ‘Set breakpoint at...’. To restore the original behaviour, replace the symbol ResetISR with main, and check the option ‘Set breakpoint at...’. Alternatively, you could delete the launch configuration and allow the IDE to create a new one.

SEGGER JLink

Edit the project launch configuration by double clicking on the launch config file, select the Startup tab, replace main with ResetISR

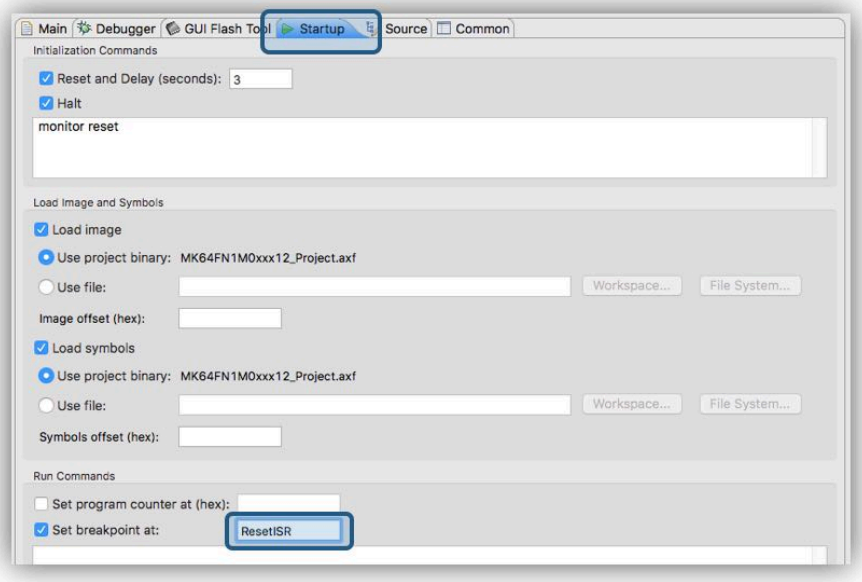


Figure 13.17. Debug Launch ResetISR Segger

When a debug connection is made, the target should halt at this symbol.

To disable the initial breakpoint, uncheck the option ‘Set breakpoint at...’. To restore the original behaviour, replace the symbol ResetISR with main, and check the option ‘Set breakpoint at...’. Alternatively, you could delete the launch configuration and allow the IDE to create a new one.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

13.3.4 Debugging Pre-loaded binaries (Add Symbols)

In a typical debug scenario, a project is built, programmed into flash and debugged. However, a common requirement may be to debug via a bootloader, or debug additional code preloaded (into flash) generated by another project(s).

For a good debug experience, symbolic information (and source) for additional project code is must be made available to the debug environment.

Symbolic information from additional projects can now easily be added via the *Other Symbols* tab on a project's launch configuration as shown below.

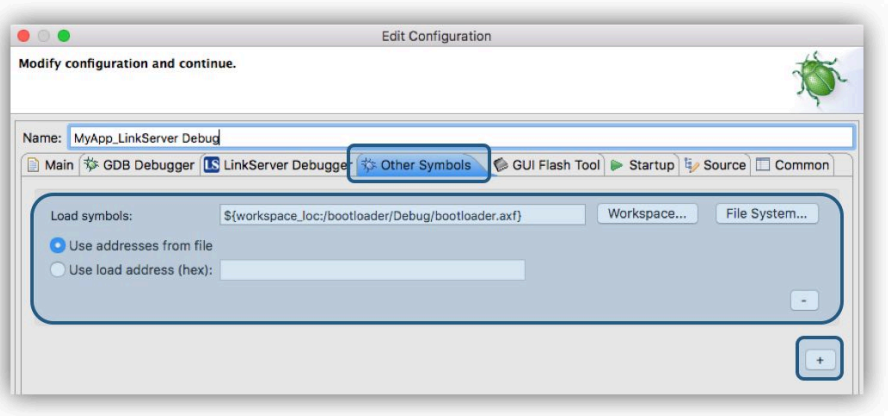


Figure 13.18. Debug Launch Additional Symbols

To add symbolic information from other projects, simply browse to their .axf files and either use either the default address or set a new base address for the image data. Use the + button to add further symbolic information.

13.3.5 Disconnect Behaviour

Once the user has completed a debug session, the debugger connection can be terminated via the IDE's Terminate button! The exact behaviour of the target will depend on the particular debug solution.

LinkServer

For LinkServer, the launch configuration contains a set of options to control what the target should do when terminated. The default option is for the target to continue running from the current PC value, however this can be changed by selecting a new setting within the launch configuration.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

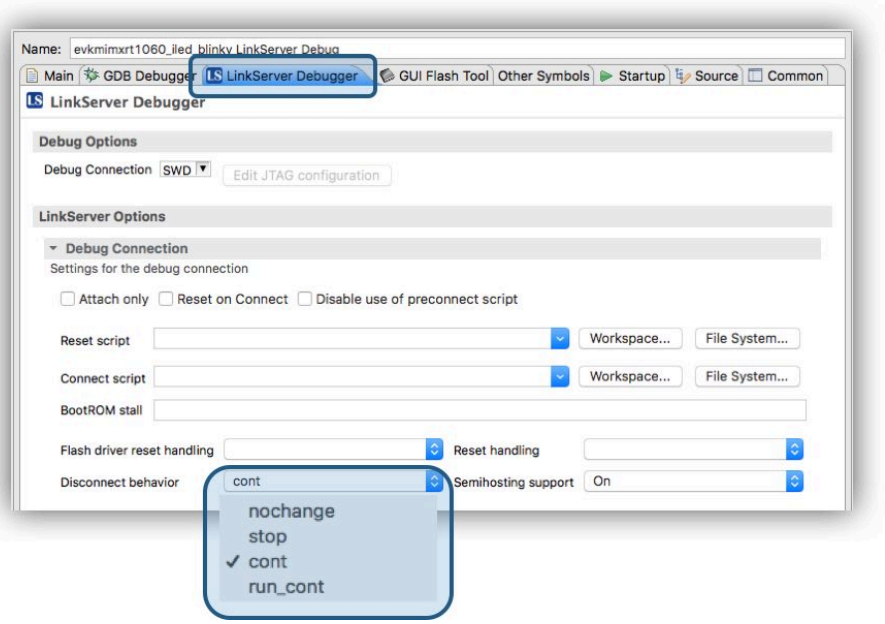


Figure 13.19. Debug Launch Disconnect Mode

Where:

- **nochange** - will leave the target in its current state
- **stop** - will leave the target in debug state i.e. halted
- **cont** - the default, will either start the image from its current PC value or leave it running
- **run cont** - will reset the target and let it run

PEmicro

The Terminate button will force the target to halt. Alternatively, for PEmicro debug the IDE supports another option – to disconnect and force the target to run. This can be achieved via the IDE's disconnect button.

SEGGER JLink

The target will *Run* on disconnect by default. The launch configuration option, *Disconnect behaviour* can be changed to *Halt* causing the target to halt on disconnect.

13.3.6 Project Flash Programming

Launch configuration dialogues now contain a GUI Flash Tool tab. This along with the Advanced GUI Flash Tool [167] and Debug Shortcuts [123] provide access to the flash programming capabilities each of the supported debug solutions.

For each debug solution, the options will vary slightly but the presentation is broadly the same as shown below. These options are self describing.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

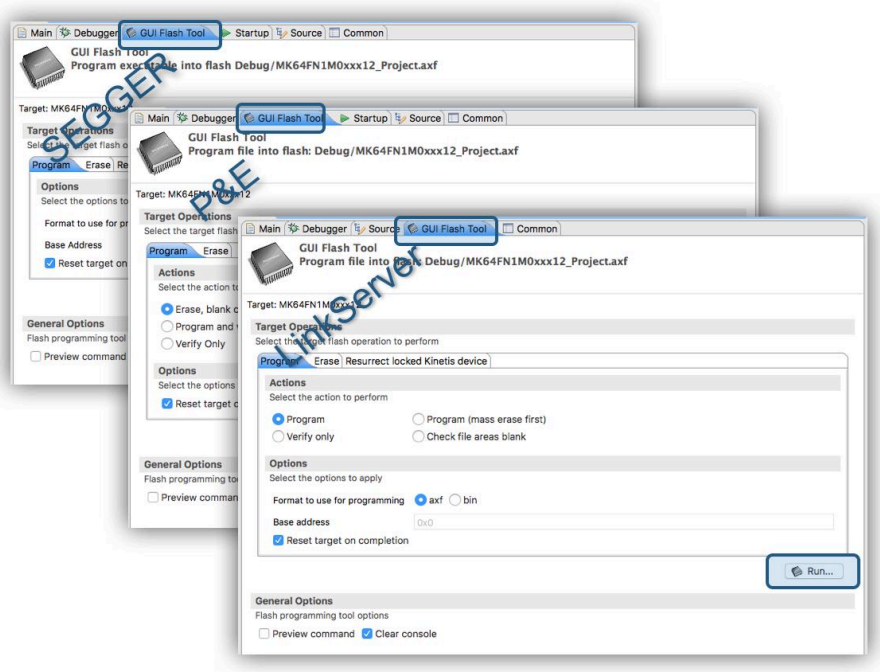


Figure 13.20. Debug Launch Flash Programming

To perform the selected operation, simply click the *Run* button.

Important Note: By default, a launch configuration will be created with *Program* as the default Program action, and *Mass Erase* as the default Erase action. When settings are changed by the user they will be stored within that project's launch configuration and will remain until manually changed (or the launch configuration is deleted). When Debug Shortcuts [123] are used, they will action the current settings within the selected projects launch configuration (or if none exists, create a new default launch configuration) - therefore if the *Program* action is set to *Verify*, a Verify will be performed as the Program action.

13.4 Breakpoints

When viewing source (or disassembly) during a debug session, you can toggle breakpoints by simply clicking/double clicking in the left most side of the source view, typically shown as a light blue column. This is also where the breakpoint symbol is shown when one is set. This can be done when the target is paused or running.

Breakpoints (and Watchpoints) are also displayed, and can be deleted or disabled in the Breakpoints View. If you are using the "Develop" perspective, then by default it will be in the bottom left of the MCUXpresso IDE window tabbed with the Quickstart and other views

If you have closed the Breakpoint view at some point, then you can re-open it using the "Window -> Show view" menu or 'Window -> Perspective -> Reset Perspective".

13.4.1 Breakpoint Types

At a basic level there are 2 types of breakpoints:

- Hardware: these are limited in quantity but can be set on ROM (Flash) or RAM. These breakpoints are provided by the debug hardware built into to the CPU.
- Software: these are implemented by a software instruction *BKPT* and can in normal circumstances only be placed on addresses within RAM (since the underlying code must be changed). These breakpoints can be applied in any quantity and are invisibly placed (and removed) by the debugger.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Usually the debugger will automatically decide the best breakpoint to use for a particular memory type or circumstance and this is invisible to the user.

Simplistically, software breakpoints will be placed in RAM and Hardware breakpoints are placed in ROM (Flash).



Tip

On some systems, a bootloader may copy code from ROM into RAM for execution – if a symbol within this code is breakpointed – such as main(), then the debugger may select a software breakpoint since it knows that main() will reside in RAM. A problem can arise if the software breakpoint is set by the debugger before the bootloader has relocated the code. If this occurs, any software breakpoint will be overridden by the relocated code. MCUXpresso IDE includes support for plain load images, [217] - to ensure this problem does not arise in this case, MCUXpresso IDE will force a hardware breakpoint onto main(). This will not be overridden since this breakpoint type makes no changes to memory.

13.4.2 Breakpoints Resources

When debugging code running from Flash memory, the debugger is limited on how many breakpoints it can set at any time by the number of hardware breakpoint units provided by the ARM CPU within the MCU.

Note: Code located in RAM can use a different breakpoint mechanism offering the capability of essentially unlimited breakpoints.

Typically, the number of hardware breakpoints/watchpoints that can be set are as follows:

```
Cortex-M0/M0+ (LPC) - 4 breakpoints, 2 watchpoints
Cortex-M0/M0+ (Kinetis) - 2 breakpoints, 1 watchpoints
Cortex-M3/M4/M7 - 6 breakpoints, 4 watchpoints
```

ARM does provide a level of implementation flexibility, so always consult your MCU documentation.

If you try to set too many breakpoints/watchpoints when debugging, then the precise behaviour will depend on the debug solution you are using. For LinkServer an error of the form below will be generated.

```
15: Target error from Set break/watch
Unable to set an execution break - no resource available.
```

To fix the problem, simply remove the excess breakpoint(s).

Also remember that a breakpoint will be (temporarily) required for the initial breakpoint set by default on the function main() when you initially debug your application. A breakpoint may also be required (temporarily) when single stepping code.

Note: When the target is paused, any number breakpoints may be set within the source or disassembly views of the IDE, however only when the target is Resumed (Run) will the low level debug hardware attempt to set the required breakpoints. Therefore it is possible to request many more breakpoints that are supported by the target MCU leading to the error described above.

13.4.3 Skip All Breakpoints

You can use the “Skip all breakpoints” button  in the Breakpoints view (or on the main toolbar) to temporarily disable all breakpoints. This can be particularly useful on parts with only a few

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

breakpoints available, particularly when you want to reload your image, which will typically cause the default breakpoint on main() to be temporarily set again automatically by the tools.

13.5 Watchpoints

Watchpoints are Breakpoints for Data and are often referred to as Data Breakpoints. Watchpoints are a powerful aid to debugging and work by allowing the monitoring of global variables, peripheral accesses, stack depth etc. The number of watchpoints that can be set varies with the MCU family and implementation.

Watchpoints are implemented using watchpoints units which are data comparators within the debug architecture of an MCU/CPU and sit close to the processor core. When configured they will monitor the processor's address lines and other signals for the specific event of interest. This hardware is able to monitor data accesses performed by the CPU and force it to halt when a particular data event has occurred.

The method for setting Watchpoints is rather more hidden within the IDE than some other debugging features. One of the easiest ways to set a Watchpoint is to use the Outline View, which by default this will be located within the IDE **Quickstart** panel.

From this view you can locate global and static variables then simply select **Toggle Watchpoints**.

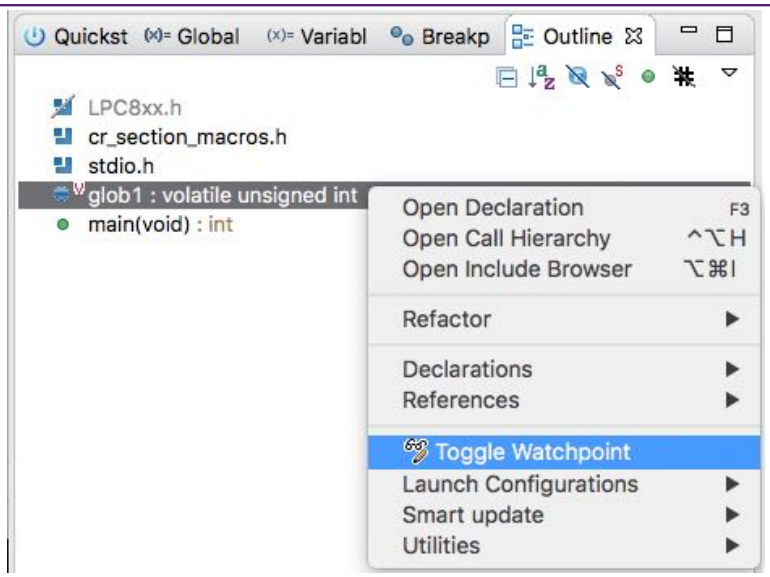


Figure 13.21. Toggle Watchpoint

Once set, they will appear within the Breakpoint pane alongside any breakpoints that have been set.

Watchpoints can be configured to halt the CPU on a Read (or Load), Write (or Store), or both. Since watchpoints 'watch' accesses to memory, they are suitable for tracking accesses to global or static variables, and any data accesses to memory including those to memory mapped peripherals.

Note : To easily distinguish between Breakpoints and Watchpoints within the Breakpoint view, you can choose to group entries by Breakpoint type. From within the Breakpoints view, click the Eclipse Down Arrow Icon Menu, then you can select to Group By Breakpoint Types as shown below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

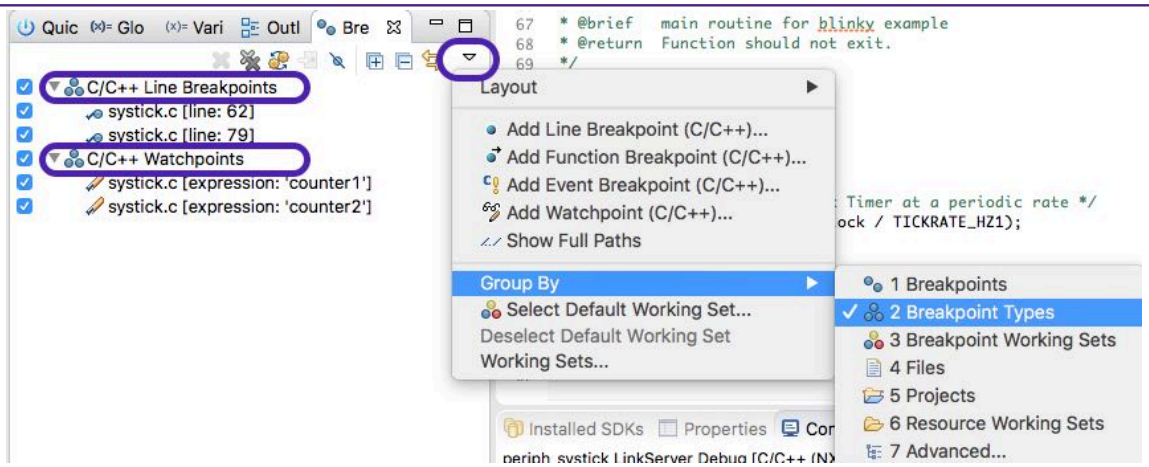


Figure 13.22. Watchpoints View

As you can see from the above graphic, the option to set a Watchpoint is also available directly from the Breakpoint view. When set from here, you will be offered an unpopulated dialogue – simply entering an address will cause a watchpoint to be created, monitoring accesses to that location.

Another place to set Watchpoints within the IDE is from the context sensitive menu within a Memory view.

Note: Watchpoint resources are shared with other debug features, in particular an SWO Data Watch item will require a dedicated watchpoint unit to monitor the value.

Note: Due to the way watchpoints are implemented, any monitored access will be performed by the CPU before a halt occurs (unlike instruction breakpoints – which halt the CPU before the underlying instruction executes). When a watchpoint is hit you will see some 'skid' beyond the instruction that performed the watched data access. If the instruction after the data access changes program flow (e.g. a branch or function return), then the IDE may not show the instruction or statement that caused the CPU to halt.

Note: Application initialisation performed by the C library may write to monitored memory locations, therefore you may see your application halting during startup if watchpoints have been set on initialised global data.

13.5.1 Using Watchpoints to monitor stack depth

Watchpoints provide a very simply way of monitoring stack depth when an application is running.

Stacks on ARM based processors use a Full Descending scheme and so have the potential to descend into areas of memory used for other purposes (typically holding global data or the heap). Establishing the maximum depth of an applications stack can be a challenge especially since any memory corruption due to excessive stack use may not be immediately apparent. Watchpoints may be used to monitor and trap the stack exceeding a particular depth during execution enabling positive reassurance that the true stack depth is understood.

The graphic below shows the use of the breakpoint view feature *Add Watchpoint (C/C++) ...* where an address has been selected to watch for the Stack reaching 0x10007D00.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

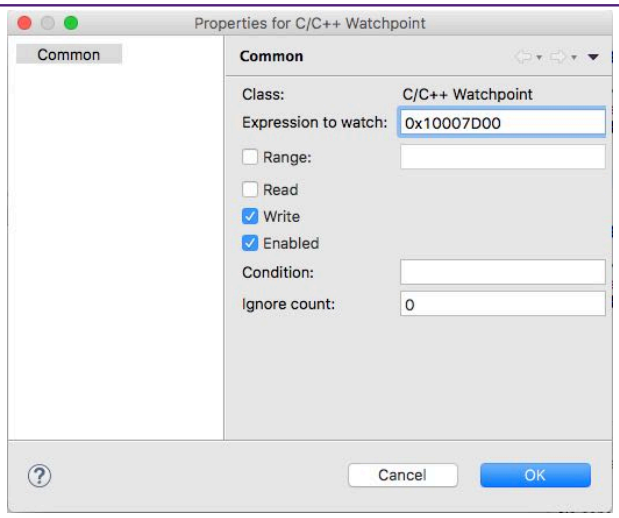


Figure 13.23. Watchpoint on Stack Depth

13.6 Registers

The Register view, by default located next to the Project Explorer view, will display the internal ARM CPU registers when the core is halted i.e. when there is an active debug connection but the target is paused. The contents of the registers view will vary depending on the nature of the ARM CPU inside the MCU being debugged, however the base register set will be available for all MCUs.

The Register list as displayed is made up from, the Basic Register set (Core Registers), Fault and Status Registers, Pseudo Registers, and finally Floating point Registers (for Cortex M4/M7 etc.). Since the register set for many MCUs is large, individual register groups can now be hidden if required to reduce screen usage.

Note: For many debug tasks, the values of the CPU registers will be of little concern, however when debugging at the disassembly level (and single stepping), these values can be a powerful debugging aid. For an in depth understanding of the ARM register set for the CPU within your NXP MCU, please consult the documentation available from ARM.



Tip

Even when operating in LinkServer None Stop mode, registers cannot be read or written when the target is executing and the register display may appear blank.

13.6.1 Basic Register set (Core Registers)

The basic register set comprises the CPU's 16 32 bit core registers (r0 – r15), plus the program status register, certain registers have a special function:

- r13 – SP Stack Pointer, this holds the address of the last entry on the stack
- r14 – LR Link Register, this holds the return address for a BL (branch with link) instruction
- r15 – PC Program Counter, this holds the address of the instruction (to be) executed
- xpsr – program status register, this combines the Application (APSR), Interrupt (IPSR) and Execution (EPSR) program status registers, reflecting the state of the CPU
- flags – set by certain instructions performing arithmetic operations (contained within the APSR)

The register set (for a Cortex M4 CPU) is displayed below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

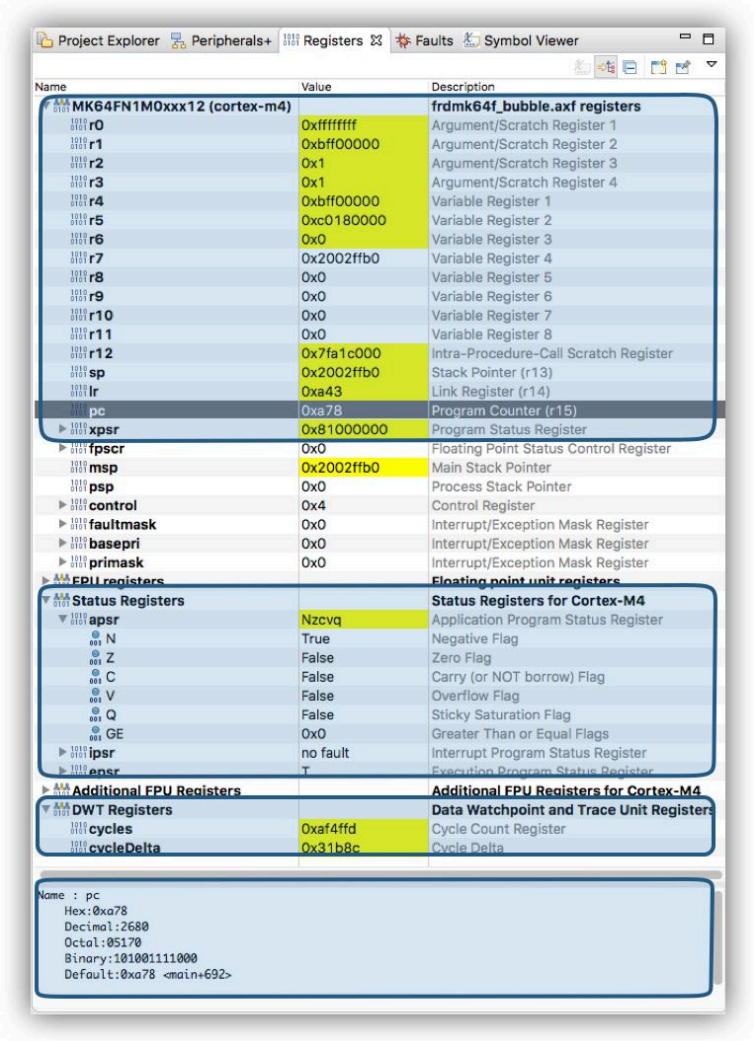


Figure 13.24. Registers View

Note: in this graphic the floating point registers have been hidden

Four blocks of registers are highlighted within the graphic

- registers r0 – r15 and the xpsr (the components of this are shown below in the status registers)
- status registers apsr ipsr and epsr, these registers together combine to form the xpsr
  - certain bit fields such as the CPU flags are expressed alpha-numerically in this view
- Cycles is a memory mapped register that increments for each core clock tick. CycleDelta is a pseudo register that records the cycles since the last pause (see more below).
- details view displays the selected register in various formats

When paused, all of these registers can be read (or written). The ability to write values to the registers set is a powerful debug feature but should be used with care.

CycleDelta

CycleDelta holds the number of core clock ticks that have occurred since the last time the CPU was paused. For example, if you run from the default breakpoint on main to a breakpoint, *cycledelta* will contain the number of clock ticks that occurred while executing this section of code. If a *step* is performed, the *cycledelta* will be the number of clock ticks for code being stepped. If *stepping* at the instruction level, this value will often be 1 because many instructions will execute within a single clock cycle.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Vectpc

In previous versions of MCUXpresso IDE the pseudo register VectPC was used to display a value when the CPU has experienced a Hard Fault. This functionality has been replaced by the Faults View [137].

13.7 Faults

During application development, errors within a program or algorithm may lead to a CPU fault (Hard Fault). These faults include:

- usage fault – such as a divide by zero
- bus fault – such as abort triggered by a memory controller
- mem manage – such as a fault triggered by a memory protection unit

Such errors can be difficult to locate, so to aid the debugging of such problems MCUXpresso IDE incorporates a *Faults view*.

If a fault occurs, the new Faults view will automatically appear and the CPU will halt (LinkServer). The view offers a set of features including identifying the nature of the fault, the location (link) of the code that caused the fault, the location (link) of the function that called the 'fault' function.

Note: for non LinkServer debug probes, a fault may leave the application running within the default fault handler (usually implemented as a *while(1)*), hence a pause might be necessary to see that a fault has occurred.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

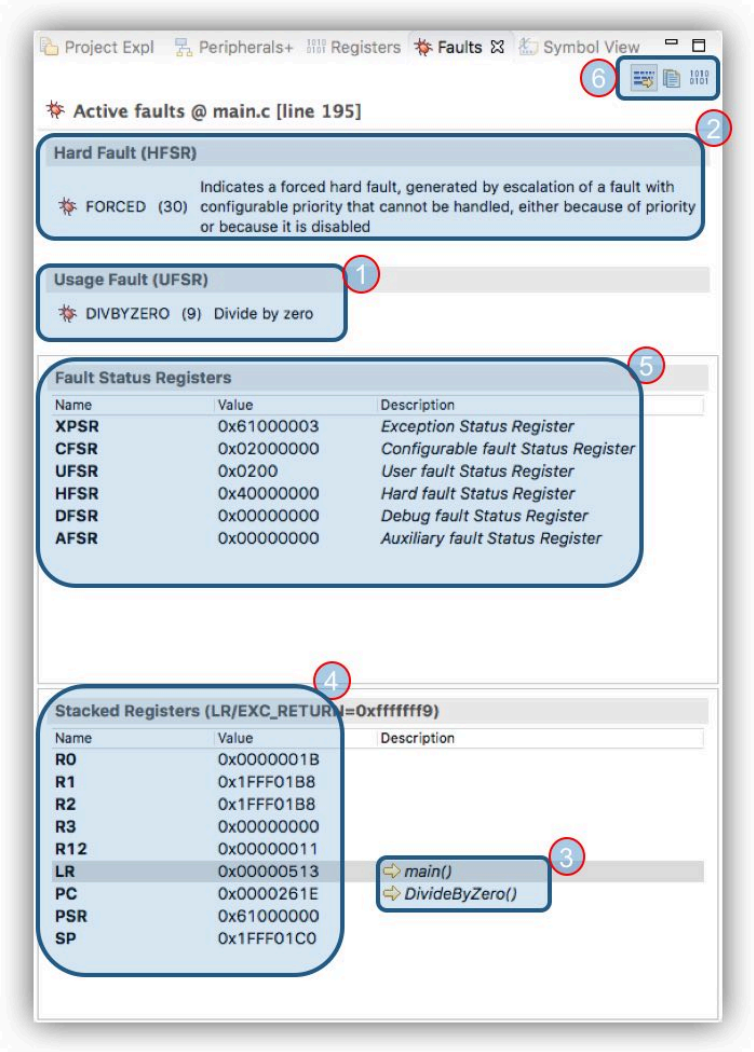


Figure 13.25. Faults View major features

This view will be titled with the source file and line number that caused the error. The view contains the following features:

1. The Fault that occurred – in this example a Usage Fault of type Divide by Zero
   - certain faults may need to be enabled within the CPU, for example Divide by Zero is enabled in the Cortex M4 Configuration and Control register
2. The Action that was taken – in this example a Hard Fault was generated
3. Links to the source of the fault function and its caller function, located from stacked registers
4. values of the registers automatically stacked on entry to the fault handler
5. fault status registers that may offer further information
6. additional options including:
   - button to cause disassembly to be opened in parallel with sources (3)
   - button to copy the fault details to the clip board
   - button to display all fault registers and descriptions rather than the

In some circumstances, a hard fault might be caused early on during the initialisation of the system before the breakpoint on main() is hit. This may mean that the fault is triggered before the debugger can take action to display the faults view. If this happens, try setting a breakpoint in the startup code – this might then allow your code to load without the hard fault being triggered. You should then be able to single step / run until the cause of the hard fault is hit. You will then see Faults View displayed.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Tip

if a repeated fault occurs that is difficult to debug, instruction trace could be enabled (when supported by the MCU) and the captured trace dumped when the fault is trapped. Looking back at the captured instructions should help find the reason for the fault condition. Please see the MCUXpresso IDE Instruction Trace guide for more information.

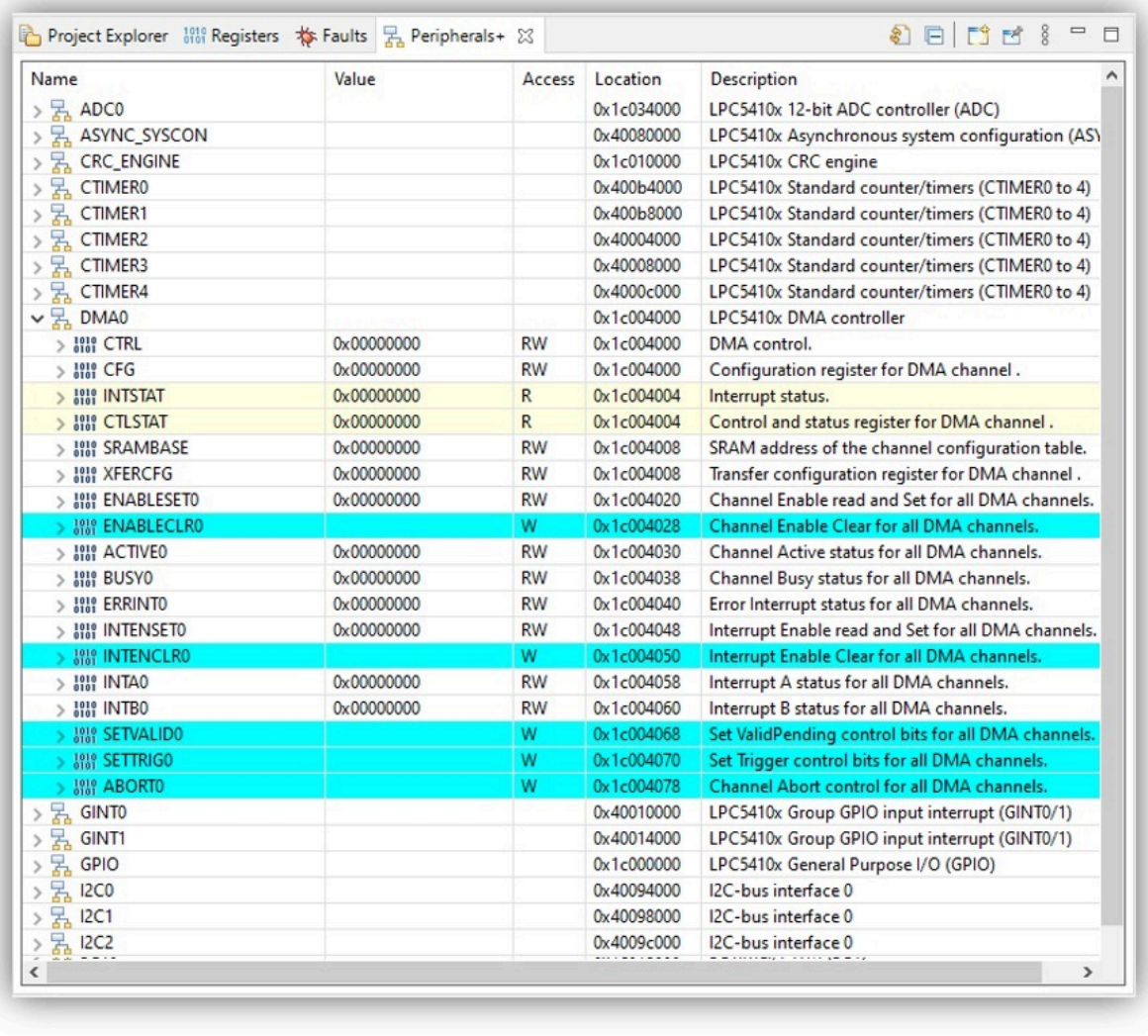
Note: Typically a Fault on an embedded system will be fatal, however this view will also assist for users developing and testing fault handlers for recoverable fault situations.

13.8 Peripherals

Peripherals is a generic term referring to both core peripherals, for example the System Timer (SysTick) and SOC/MCU peripherals such as an ADC or UART. In both instances these hardware blocks are exposed within the MCUs address space (known as memory mapped peripherals) and so can be interrogated by accesses to their specific memory locations.

MCUXpresso IDE's debug support (whether built in or provided by an SDK) includes knowledge of an MCU's peripheral set, this is available via the Peripherals tab within the Project Explorer pane (once a debug connection is made).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Name	Value	Access	Location	Description
ADC0			0x1c034000	LPC5410x 12-bit ADC controller (ADC)
ASYNC_SYSCON			0x40080000	LPC5410x Asynchronous system configuration (AS
CRC_ENGINE			0x1c010000	LPC5410x CRC engine
CTIMER0			0x400b4000	LPC5410x Standard counter/timers (CTIMER0 to 4)
CTIMER1			0x400b8000	LPC5410x Standard counter/timers (CTIMER0 to 4)
CTIMER2			0x40004000	LPC5410x Standard counter/timers (CTIMER0 to 4)
CTIMER3			0x40008000	LPC5410x Standard counter/timers (CTIMER0 to 4)
CTIMER4			0x4000c000	LPC5410x Standard counter/timers (CTIMER0 to 4)
DMA0			0x1c004000	LPC5410x DMA controller
CTRL	0x00000000	RW	0x1c004000	DMA control.
CFG	0x00000000	RW	0x1c004000	Configuration register for DMA channel .
INTSTAT	0x00000000	R	0x1c004004	Interrupt status.
CTLSTAT	0x00000000	R	0x1c004004	Control and status register for DMA channel .
SRAMBASE	0x00000000	RW	0x1c004008	SRAM address of the channel configuration table.
XFERCFG	0x00000000	RW	0x1c004008	Transfer configuration register for DMA channel .
ENABLESET0	0x00000000	RW	0x1c004020	Channel Enable read and Set for all DMA channels.
ENABLECLR0		W	0x1c004028	Channel Enable Clear for all DMA channels.
ACTIVE0	0x00000000	RW	0x1c004030	Channel Active status for all DMA channels.
BUSY0	0x00000000	RW	0x1c004038	Channel Busy status for all DMA channels.
ERRINT0	0x00000000	RW	0x1c004040	Error Interrupt status for all DMA channels.
INTENSET0	0x00000000	RW	0x1c004048	Interrupt Enable read and Set for all DMA channels.
INTENCLR0		W	0x1c004050	Interrupt Enable Clear for all DMA channels.
INTA0	0x00000000	RW	0x1c004058	Interrupt A status for all DMA channels.
INTB0	0x00000000	RW	0x1c004060	Interrupt B status for all DMA channels.
SETVALID0		W	0x1c004068	Set ValidPending control bits for all DMA channels.
SETTRIG0		W	0x1c004070	Set Trigger control bits for all DMA channels.
ABORT0		W	0x1c004078	Channel Abort control for all DMA channels.
GINT0			0x40010000	LPC5410x Group GPIO input interrupt (GINT0/1)
GINT1			0x40014000	LPC5410x Group GPIO input interrupt (GINT0/1)
GPIO			0x1c000000	LPC5410x General Purpose I/O (GPIO)
I2C0			0x40094000	I2C-bus interface 0
I2C1			0x40098000	I2C-bus interface 0
I2C2			0x4009c000	I2C-bus interface 0

Figure 13.26. Peripherals View

In this view each peripheral is listed along with its value, base address, access and a brief description. The view also exposes the inner peripheral registers and offers bit field enumerations to greatly simplify both reading existing configurations and setting new values.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

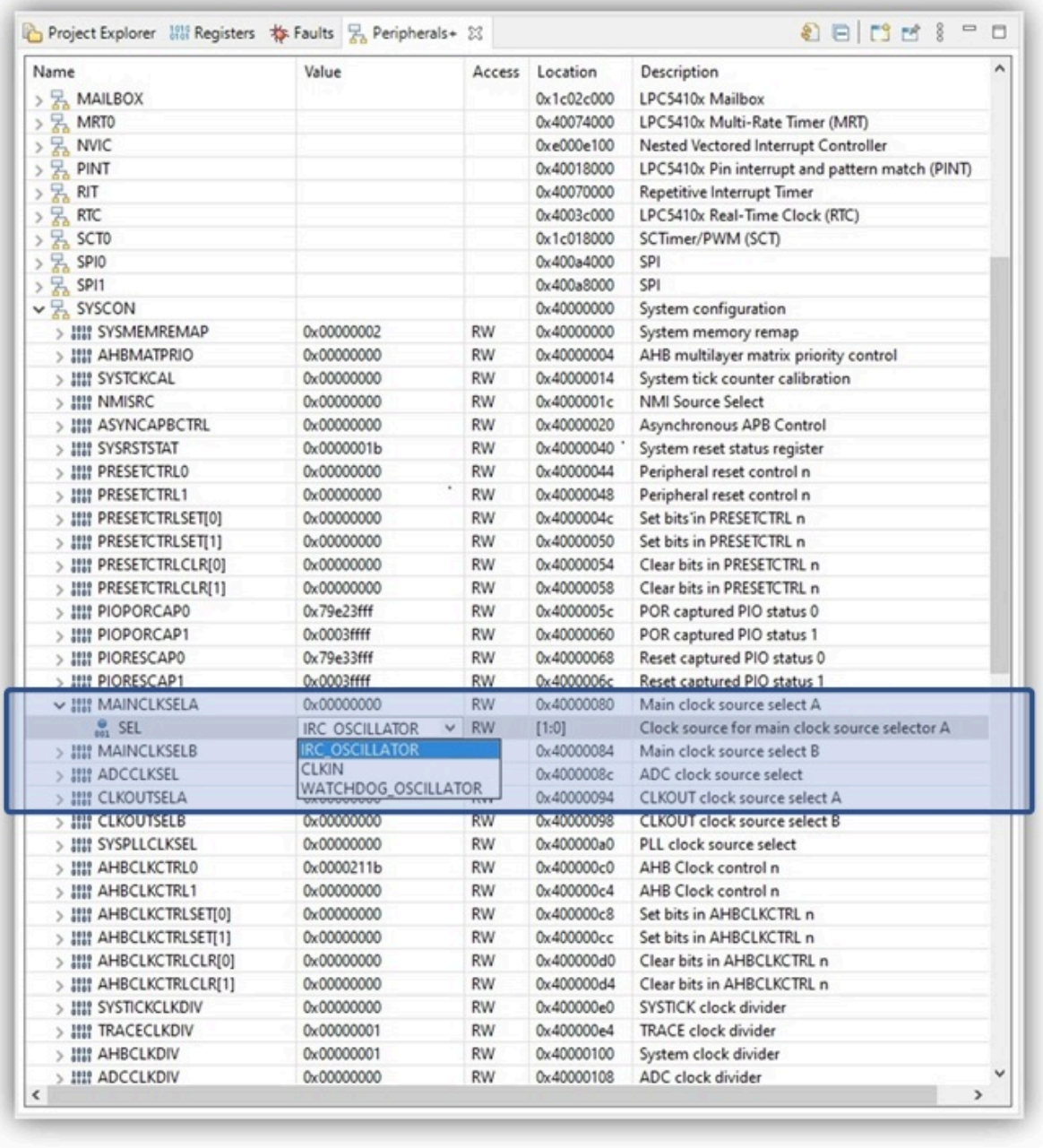


Figure 13.27. Peripheral Register View Modifying Bit Field Value

Important Note: When an MCU powers up, many peripherals will be unavailable because they are unpowered/not clocked. Attempting to access a peripheral in this state will result in failure, and the peripheral view will simply display them in red. Certain peripherals may be partially available, unavailable sections will again display in red. Entries that have changed, will display in yellow.



Tip

Even when operating in LinkServer None Stop mode, peripherals can not be read or written when the target is executing. The main peripheral display may appear blank when the target is executing regardless of LinkServer mode.

Warning: It is **strongly** advised that only peripherals that are well understood are accessed in this manner since attempting to view certain peripherals can break a debug connection or perform other unexpected actions. MCUXpresso IDE's debug features cannot offer protection from such occurrences.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

The view also lists in the main menu the device memory regions. If these memory regions are selected, a standard hex memory display will be created. Memory regions are not peripherals in the normal sense but are included here so their memory space can be easily displayed.

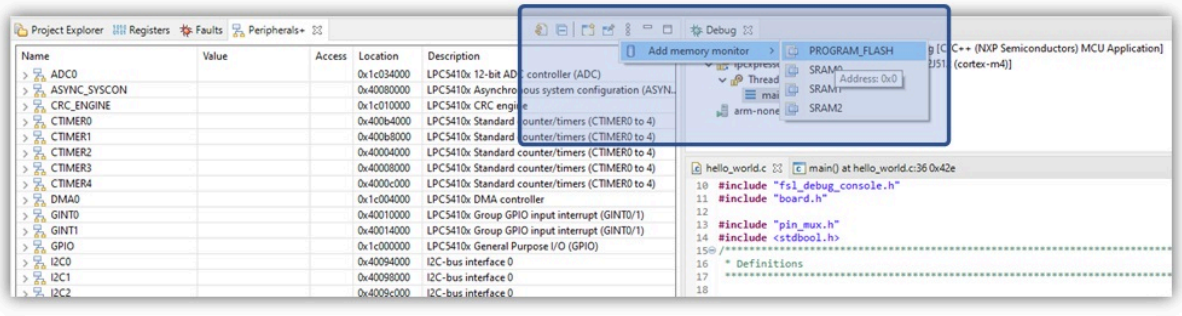


Figure 13.28. Peripherals View Memory Regions

13.8.1 Custom SVD File

User can specify a custom SVD file location inside of a project. This can be achieved from *Project Properties -> Run/Debug Settings -> MCU Settings -> SVD Selection*. The file will be used by Peripherals+ view when debugging that project.

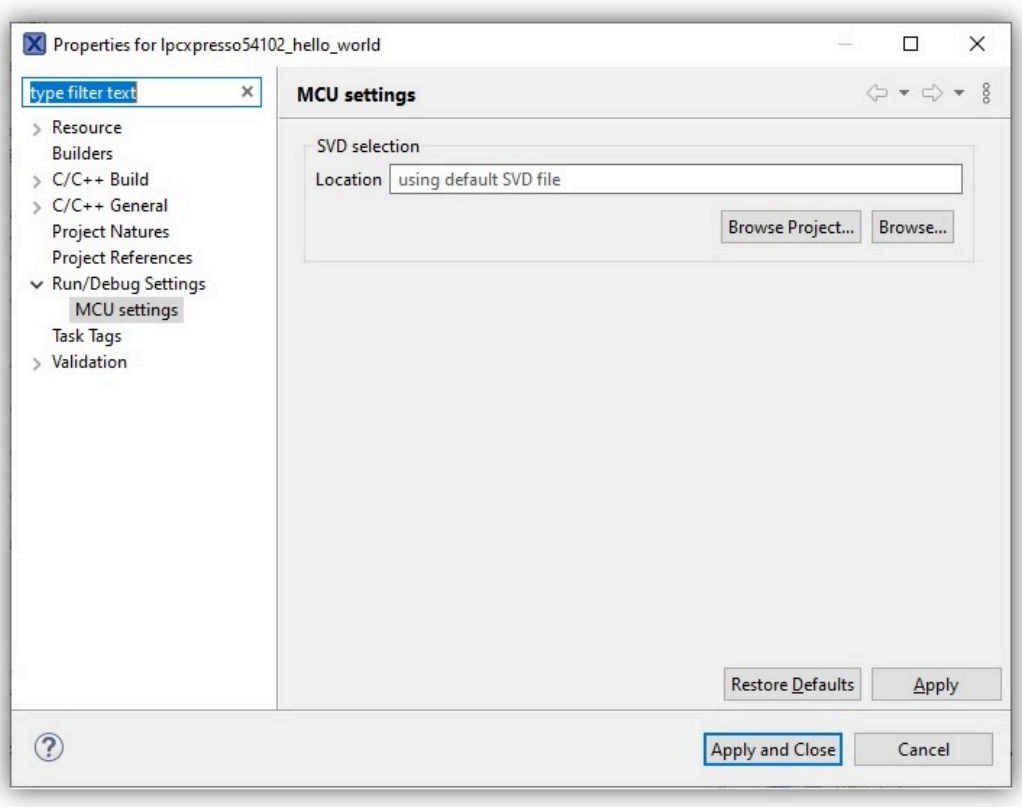


Figure 13.29. SVD Custom Selection

The SVD file can be imported from:

- the selected project
- a custom file location

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

In the default case, where this file is not specified, the SVD is loaded from the associated SDK.

13.9 Offline Peripherals

MCUXpresso IDE provides a way to inspect the peripheral registers without the need for an active debug session. Registers are shown with the fields, but of course without the actual values. Instead the view shows the reset value for the registers.

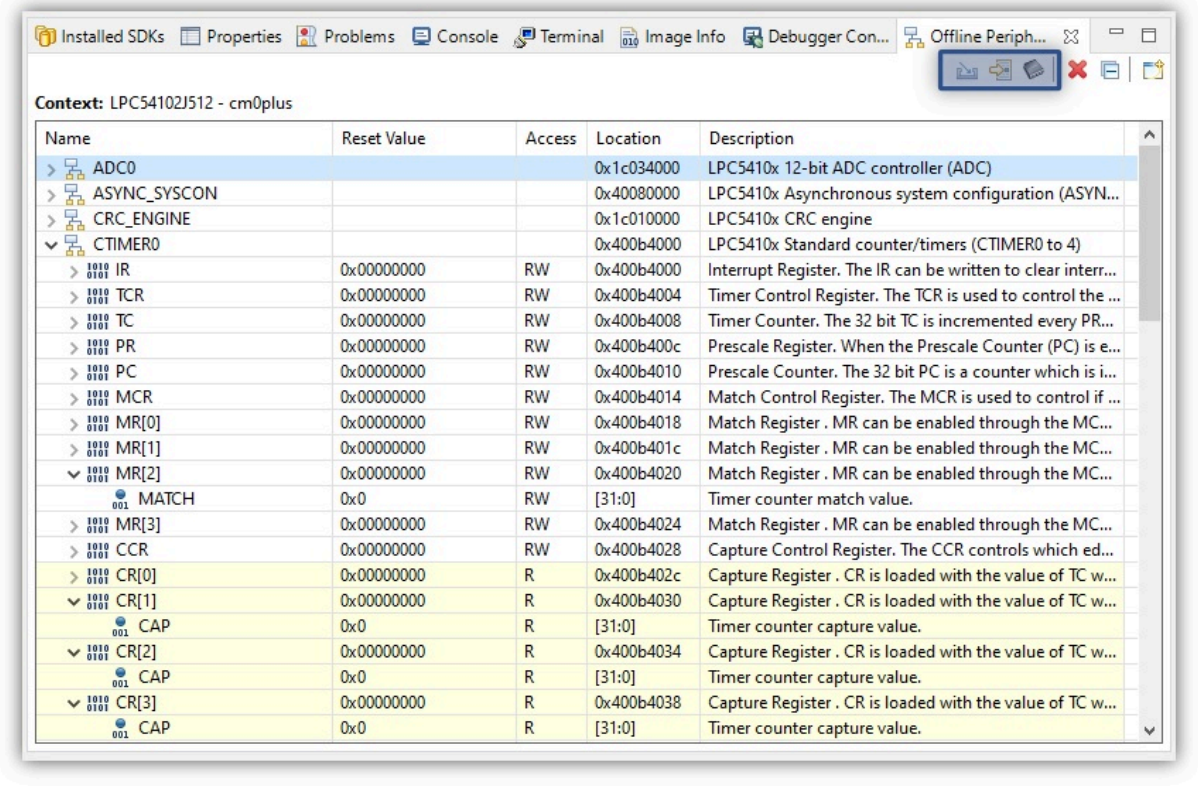


Figure 13.30. Offline Peripherals View

The view provides three ways of importing peripherals:

- from a local file
- for the device/core used by the selected project
- from a list of available devices



Tip

Holding the mouse over the bit field, it will show a tooltip that includes the detailed description for the current value and also the descriptions for the other possible values.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

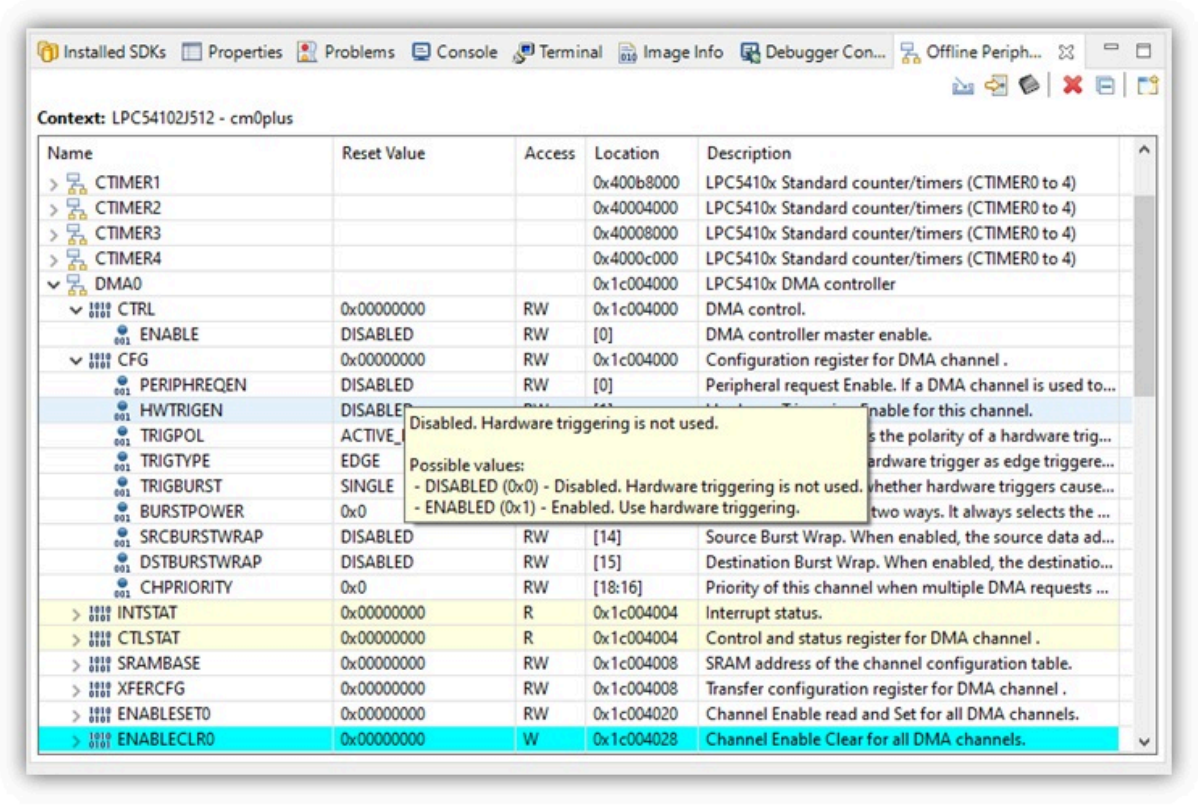


Figure 13.31. Offline Peripherals View Bit Field Information

13.9.1 Loading Custom SVD File in Offline Peripherals view

To load the custom SVD file which was set into Project Properties, just push the “Load peripherals for device used by selected project” button. The location from where it was provided can be found in Context.

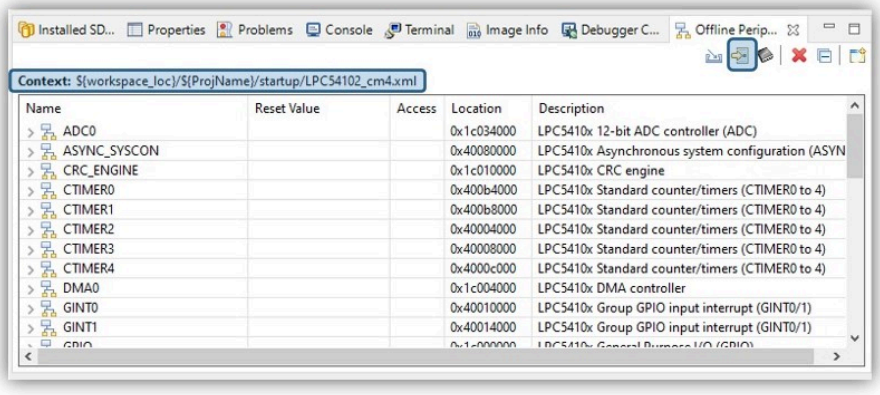


Figure 13.32. Load Custom SVD File

13.10 Global and Live Global Variables

Global and Static variables are stored within system RAM memory and can therefore be accessed by the debug chain (read and potentially written) while an application is both paused and running.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: ARM processor inside the NXP MCU utilises a load store architecture, this means that a global variable must be read (loaded) from memory and then written back by the processor (if changed). The value of the variable displayed will correspond to the value in memory and this may potentially be different from the value held by the processor. Modern MCUs execute millions of instructions every second, so any variable observed while an application is running may have been changed many times from the value displayed in the view, therefore take care that this is understood before attempting to change a variable value within the Global variable view.

This view can be populated from a selection of a project's global variables. Simply click the "Add global" button to launch a dialogue:

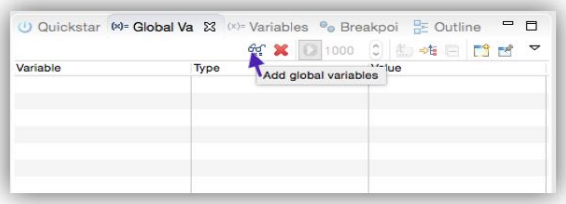


Figure 13.33. Add Global Variables

This will then display a list of the global variables available in the image being debugged. Select the ones of interest via their checkboxes and click OK :

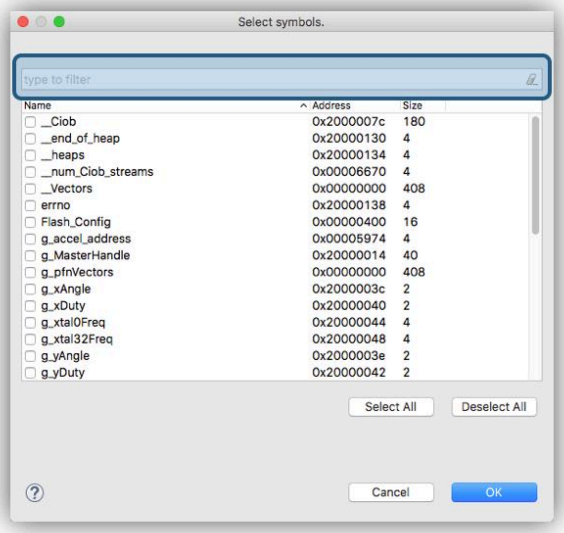


Figure 13.34. Global Variable Selector

Note: to simplify the selection of a variable, this dialogue supports the option to filter (highlighted) and sorts on each column.

Once selected, the chosen variables will be remembered for that occurrence of the dialogue.

For all supported debug chains there is now the capability to view global variable values when the debug target (MCU) is running. When this feature is used, these are known as " **Live Variables**".

For variables to be "Live":

- the target must be running

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- the enable/disable (run) button clicked (as shown highlighted below)

Once done, the display will update at the frequency selected (selectable from 500 ms to 10 s).

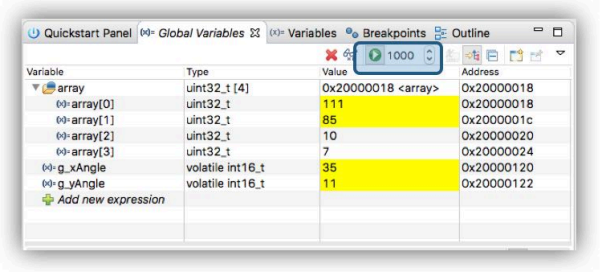


Figure 13.35. Global Variable Display

Also available is the ability to enter an expression (using standard C notation) or symbol. The expression will be evaluated and the address displayed in the Address column.

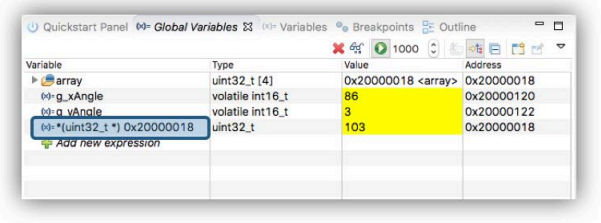


Figure 13.36. Global Variable Display Expression

Live Variables like normal Globals can also be edited in place. Simply click on the variable value and edit the contents. During the edit operation, the display will not update. This mechanism provides a powerful way of interacting with a running target without impacting on any other aspect of system performance.

Note: If you wish to have some global variables ‘Live’ and others not, then this can be achieved by spawning a second Globals display via the ‘New View’ button and populating this without enabling the ‘run’ feature for that view.

The usefulness of **Live Variables** reduces as the number of Globals monitored increases, and ultimately there will be a limit as to how many variables can be updated at the selected frequency. However, complex list of variables can be monitored if required. For example:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

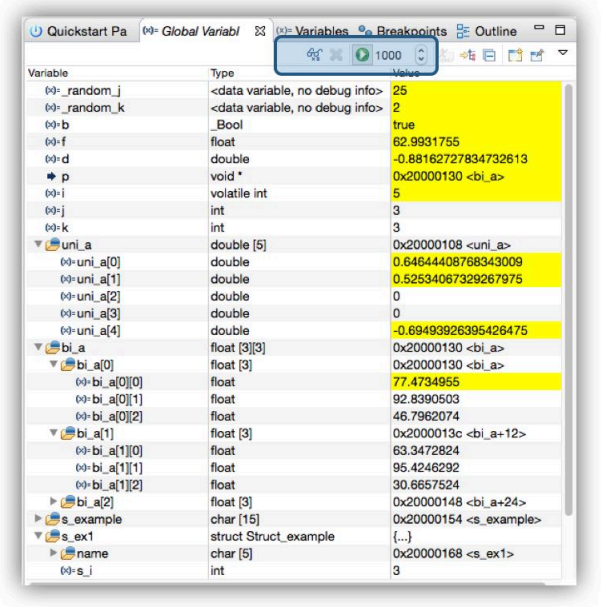
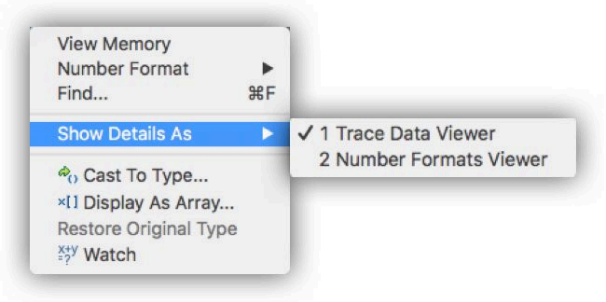


Figure 13.37. Global Variable Display Complex

13.11 Live Global Variable Plotting

In addition to displaying *Live Variables* the IDE can also trace (sample) their values for plotting as graphs, logging, or calculating statistics.

By default it is assumed that variables values may be traced but alternatively their values can be displayed in a details view via a right click menu selection.



Variables can only be traced if they have first been added to the Global Variable panel as discussed in the previous section. The selection of variables to plot is simply made by clicking to highlight the variable of interest.

Note: Once a variable has been selected, the timebase (uptime) will begin and variables values will be sampled and displayed. If additional variables are selected, their values will join the display at the current uptime. If a variable is unselected its values will no longer be sampled and displayed. If however, it is selected again within the same debug session, it will be displayed along with any previously captured values. During any period it was not selected its values will show as zero.



Tip

If the display is paused, data will still be captured but the new values will not be displayed, this can help detailed viewing of the data. Once un-paused, the captured data will be added to the display.

Note: If the target is paused, time (x-axis) will continue to advance although the display will not update until the target is resumed.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

13.11.1 Live Global Variable Graphing details

In the example below, two variables have been added to the Global variable view and both have been selected for tracing.

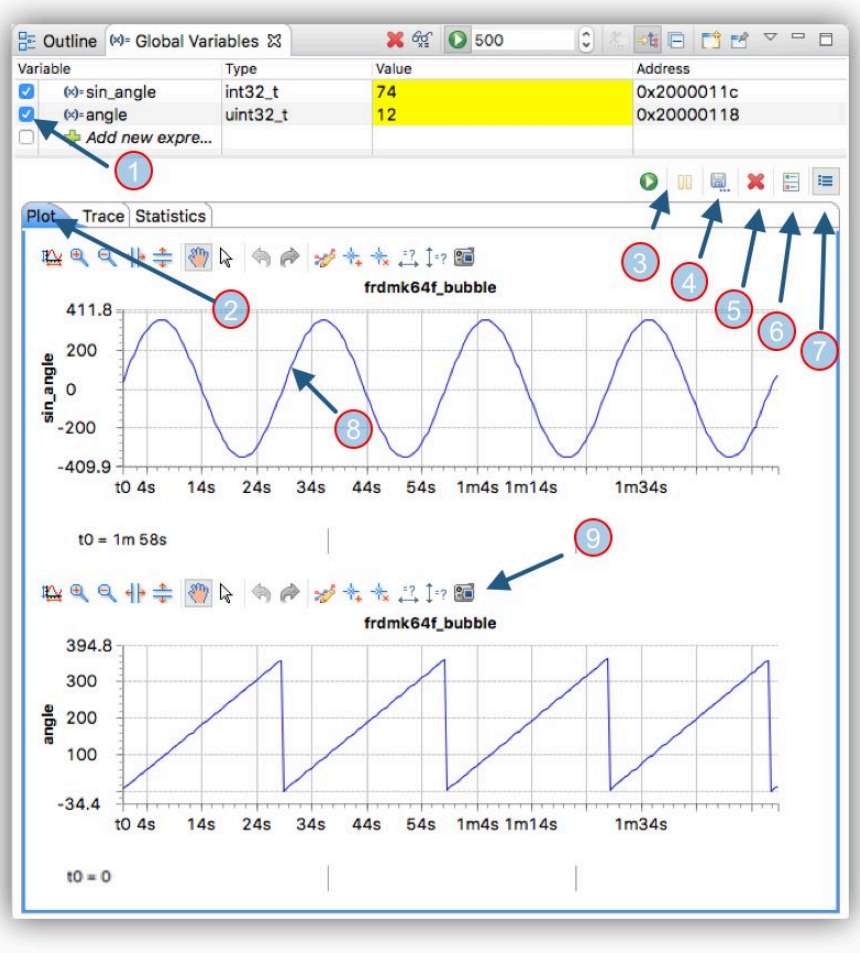


Figure 13.38. Global Variable Graphing major features

The highlighted features are explained below:

1. Selected variables for . Click the checkbox to select a variable for plotting
   - Once selected the variable will exist in the internal database of values and will remain until the debug session is terminated (even if it is later unselected)
2. Plot types: the traced data may be viewed in 3 ways:
   - Plot – display as a graph over time
   - trace – log the values
   - Statistics – calculate statistics for the traced values (max, min, average)
3. Resume and Pause: Click Resume to start plotting variables. Click to pause the graph display updated. Variables values will still be captured but the screen will not update
4. Save: Click to save the captured data.
   - the size of the png will be proportional to the size of the global view. Therefore, for more detail, increase the size of the global view before saving
   - this button will offer the option to save each of the Plot types of Plot (png), Trace (tsv), Statistics (tsv)
5. Clear Data: Display: Click to discard any traced date
6. Show or Hide the Graph Toolbar

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

7. Multiple/Single Graphs: Click to toggle the display between separate graphs for each variable and all variables plotted on a single graph
8. Click on the graph to see the X,Y co-ordinates for the selected point
9. Graph Toolbar – explained below

Clicking the button marked as (7) will combine individual graphs into a single graph view.

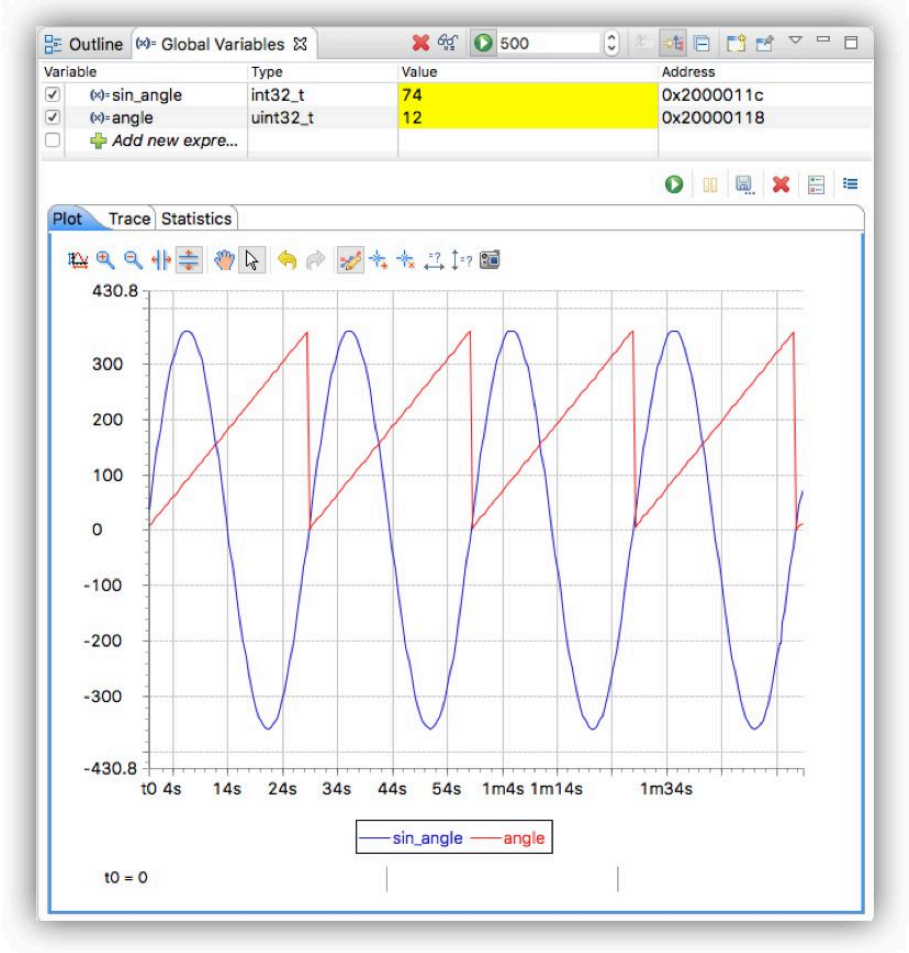


Figure 13.39. Multiple Global Variable Graph

Each graph view has an optionally visible Toolbar (6). The annotated image below shows a magnified version of the Toolbar.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

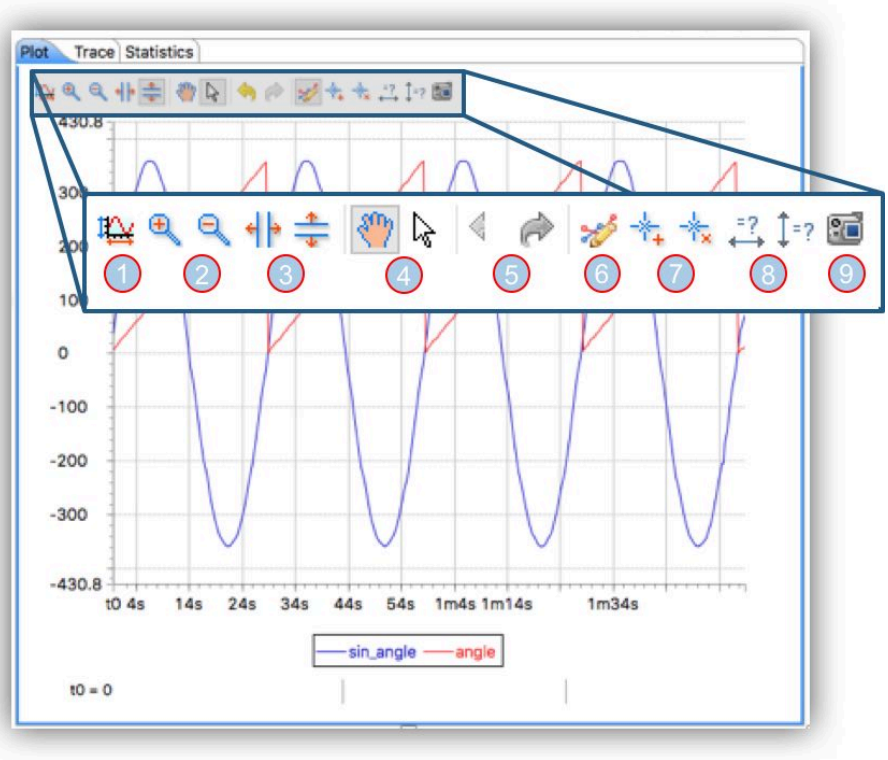


Figure 13.40. Global Variable Graph Toolbar

Where:

1. Autoscale the display to show all of the data
2. Zoom In and Out: Select the desired button and then click into the area of the graph where zoom is required
3. Zoom Horizontally and Vertically: Select the desired button and then select and drag within the graph to perform the desired zoom
4. Panning and None: Select Panning to click and drag a zoomed display. Select None to prevent interaction with the graph
5. Undo and Redo: Click these button to cycle through previous actions
6. Add a Legend (shown)
7. Add and remove Annotations. Annotations can be named and will snap to a plotted point and display its value
8. Measure Horizontally or Vertically: Click and drag to snap between plotted points to measure the value of their separation
9. Save the graph as a png file

13.12 Heap and Stack View

Located by default in the MCUXpresso IDE Develop perspective, along with the Memory view at the bottom right of the perspective.

One of the common issues within embedded system development is allocating the appropriate memory for heap and stack usage. The Heap and Stack View offers the ability to monitor heap and stack usage within their allocated regions of memory. The View allows the monitoring of heap usage in real time (while an application is running). However. since the value of the Stack is held within a processor register, Stack usage can only be updated when the application is paused.

The Heap and Stack view displays usage with respect to the configured heap and stack sizes as set within the Projects Properties at: *C/C++Build -> Settings -> Manager Linker Script -> Heap and Stack placement*

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

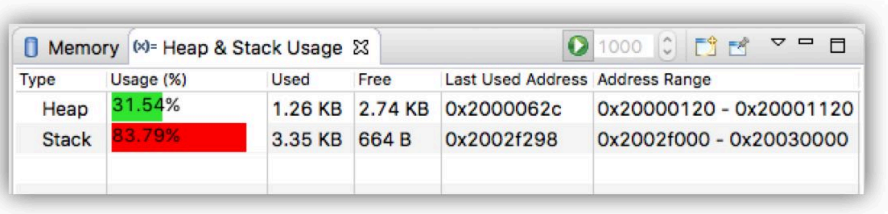


Figure 13.41. Heap and Stack View

This view will automatically update when the target is paused. To enable updating of the heap usage when the debug target is running, click the *Run* icon at the top of the view to enable or disable updates to the view. The frequency of the updates can be set between 500ms and 10 seconds.



Tip

Although real time monitoring of the stack is not possible, a watchpoint could be used to force a target halt when an access to a particular stack depth is performed. Please see further details in the section on Advanced Heap and Stack placement [154].

The symbols used to generate this view are created by the Managed Linker Script mechanism. However, other symbols can be substituted if required via the workspace preferences as show below:

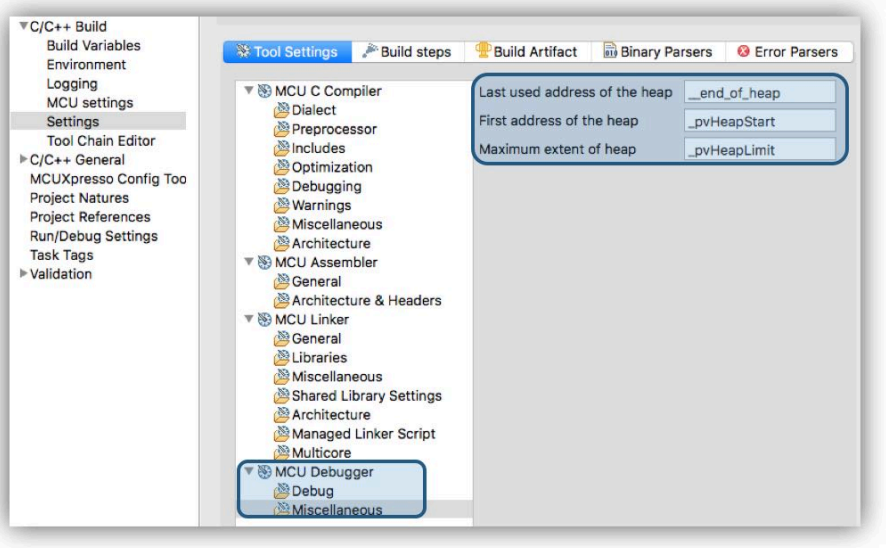


Figure 13.42. Heap and Stack View Symbols



Tip

As a guide the memory usage % display will be colored green when more than half of the available memory is free, then changing through yellow to red if more memory is used

13.13 Additional Debug Features

13.13.1 Local Variables

Situated alongside the **Quickstart** panel, the local variable view displays the local variables in scope when the target is paused. Typically, local variables are held within processor registers

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

and so are cannot be accessed when the processor is running. From this view registers can be viewed and their values edited if required.

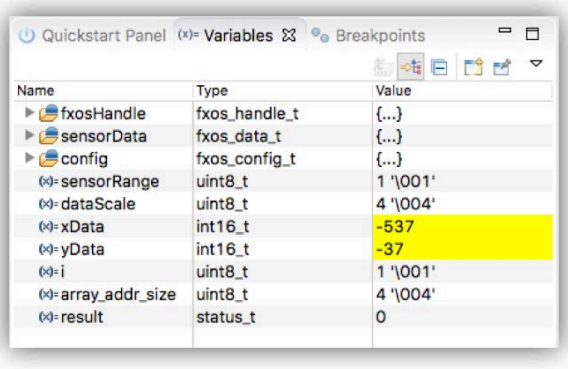


Figure 13.43. Local Variables View

13.13.2 Disassembly view

The Disassembly view allows an application's code to be viewed at the assembler level (as generated by the compiler).

The view can be enabled (if required) via the *Instruction Stepping button* within a debug stack view. This button has two functions, in that it both spawns the view and also switches stepping mode from source level to assembler level. Assembler level stepping is typically used in conjunction with the Registers view to examine the detailed behaviour of short pieces of code.

Stepping mode can be returned to source level by re-clicking this button.

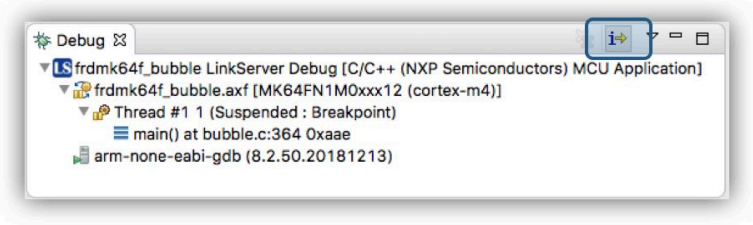


Figure 13.44. Disassembly Enable

Once enabled, the disassembly view will display the low level assembler instructions usually from the current PC.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

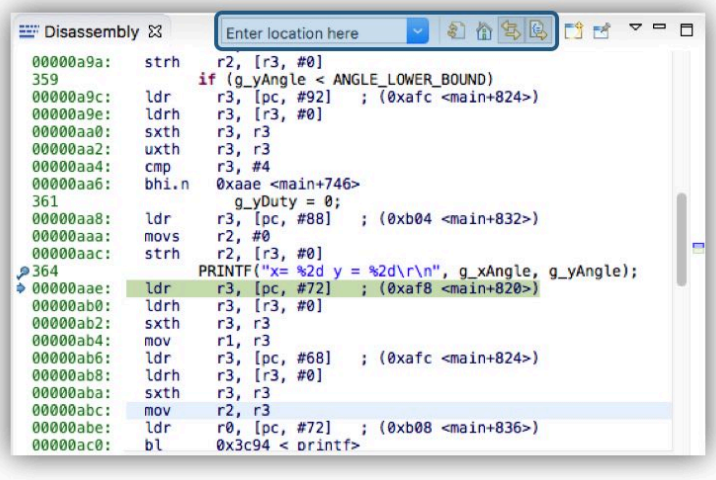


Figure 13.45. Disassembly View

The view has a number of features including:

- Setting a new address to view
- Refreshing the view contents (this might be useful if the underlying code may have changed)
- The linking and unlinking from the current debug session (PC)
- The intermixing of source code lines with their related assembler instructions
  - the usefulness of this feature decreases as compiler optimisation increases

13.13.3 Memory view

Stacked by default in the MCUXpresso IDE Develop perspective, along with the Heap and Stack view. The memory view allows debug target memory to be explored in a traditional manner. The view can be populated with target memory regions via the Peripherals [139] or by entering required address values.

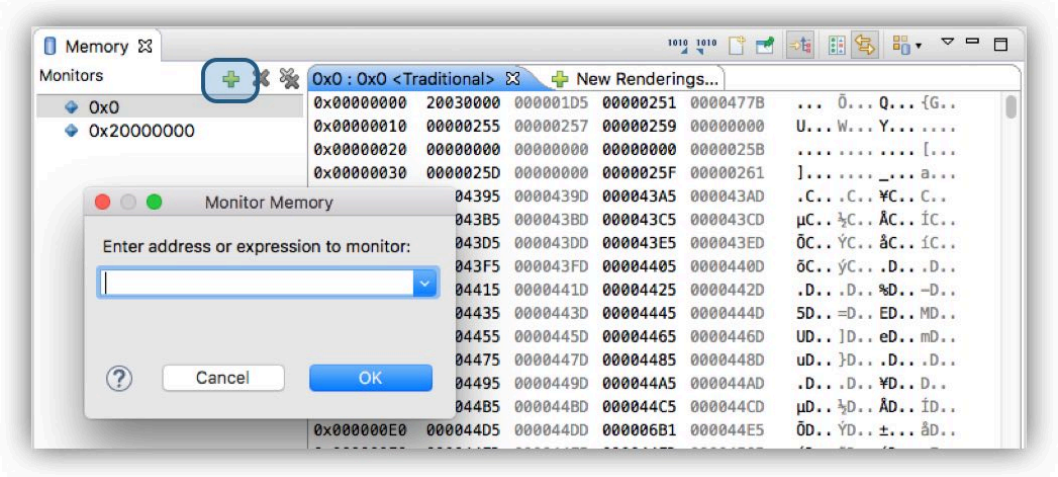


Figure 13.46. memory View

Note: Although it is technically possible to populate this view while the target is running, this mode of operation is not currently supported. Particular memory of interest can be monitored *live* via Global variable expressions if required.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

14. Configuring a Project

When a project is imported or generated using a wizard, there are many configuration options available at creation time. However, once a project has been created or if a project is shared by other means, then there still may be a requirement to make changes.

The range of possible project changes is almost infinite but below we will discuss a number of common changes that may be required and the potential ramifications that may be encountered. Note that many of these changes can be started from a project's Virtual Nodes [155].

Note: This section only discusses a few of the common changes that may be made. Please also see the sections on Memory Configuration and Linker Scripts [199], FlashDrivers [173], Library Support [188], and the additional Config Tool documentation for a more comprehensive description of the options available.

14.1 Changes available via Quickstart Quick Settings

MCUXpresso IDE provides quick access to a range of project settings via the **Quickstart** Panel as shown below:

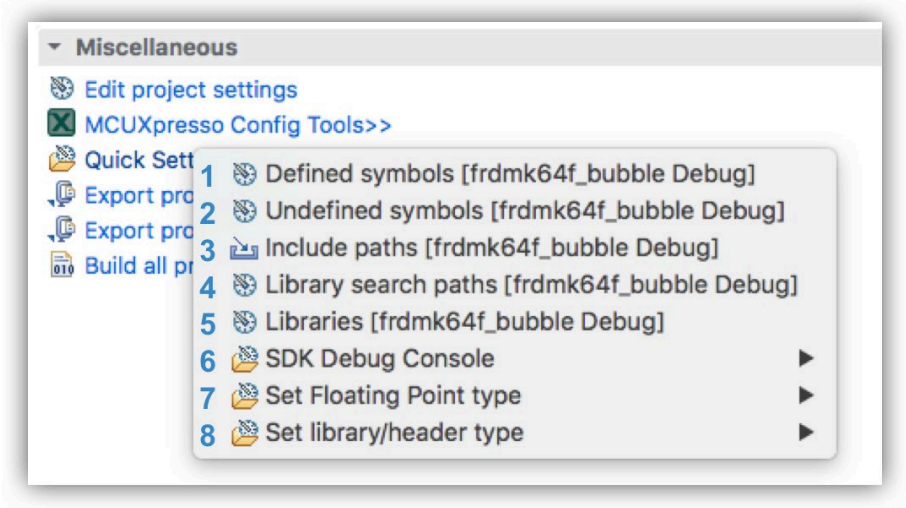


Figure 14.1. Quick Settings

Note: These settings apply to the selected project's active build configuration only and simplify access to commonly used settings normally accessed from *Properties -> C/C++ Build -> Settings* Also note Quick Settings changes may be made to multiple projects if more than one project is selected (where their settings are compatible).



Tip
The current setting for Debug Console, Floating Point and Library type is shown

1. Defined symbols – select to edit the (-D) symbols
2. Undefined symbols – select to edit the (-U) symbols
3. Include paths – select to edit the (-I) the include paths
4. Library search paths – select to edit the (-L) the library
5. Libraries – select to edit the (-l) the linker libraries search
6. SDK Debug Console – select the SDK Debug Console's PRINTF output to be via UART or to redirect via the C libraries printf function
   - selecting printf will increase the size of the project binary compared to UART output
   - for semihosted printf output to be generated, the project must be linked against a suitable library

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- for more information see the section on Semihosting and the use of printf [191]

7. Set Floating Point type – select to switch between the available Floating Point options
   - for more information see the section on Hardware Floating Point Support [265]
8. Set Library/Header type – select to switch the current C/C++ Library
   - for more information see the section on C/C++ Library Support [188]

14.2 Project Settings

Many features of a a Project can be viewed (and edited) via Virtual Nodes. Project Virtual Nodes are contained within a Project structure and provide virtual folders to display and allow the easy editing of project settings.

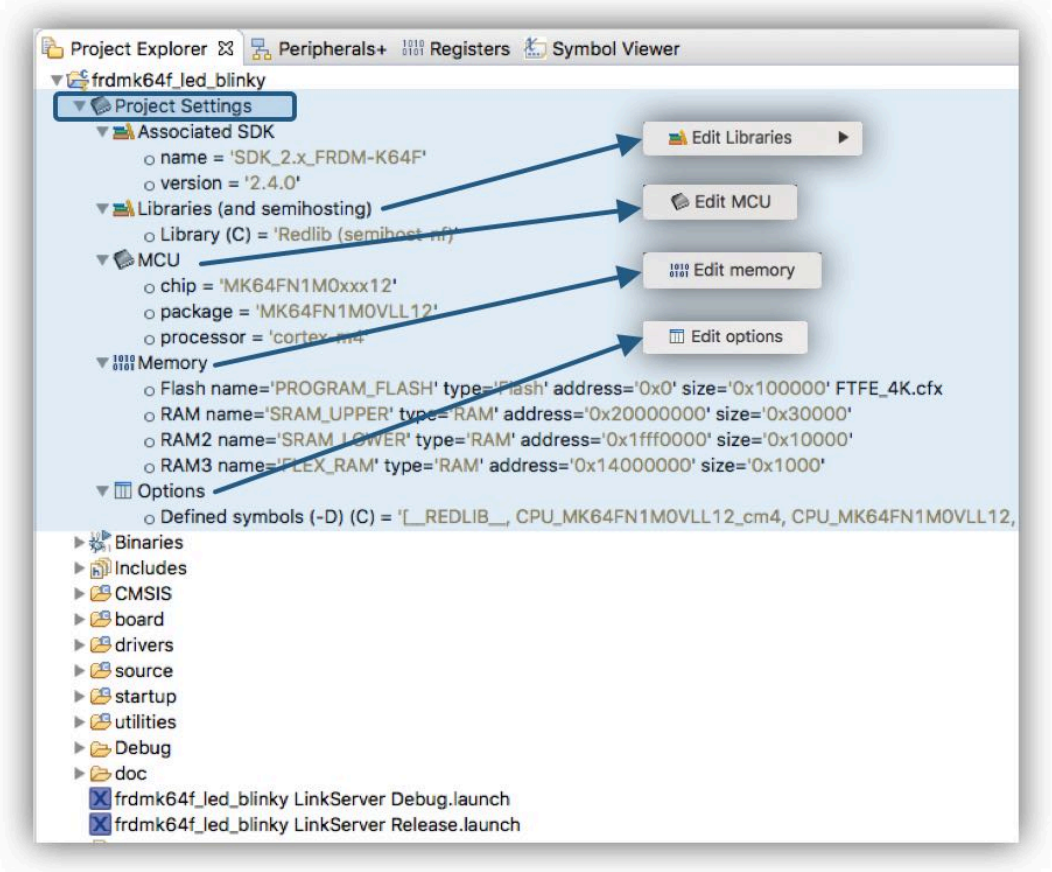


Figure 14.2. Project Settings

These are automatically generated for any project and provide a quick way to view many key project settings. In addition, a right click on these nodes provides direct options to edit the associated settings that otherwise require many more mouse clicks to reach.

14.3 Changing the MCU (and associated SDK)

All projects are associated with a particular MCU at creation time. The target MCU determines the project memory layout, startup code, LinkServer flash driver, libraries, supporting sources, launch configuration options etc. etc. so changing a project's associated MCU should not be undertaken unless you have a total grasp of the consequence of this change.

Therefore rather than changing a project's associated MCU, it is strongly recommended that instead a new project is generated for the desired MCU and this new project is edited as required.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

However, on occasion it may be expedient to reset a project's MCU (and associated SDK) and this can be achieved as follows. From the project virtual nodes, select *Edit MCU*.

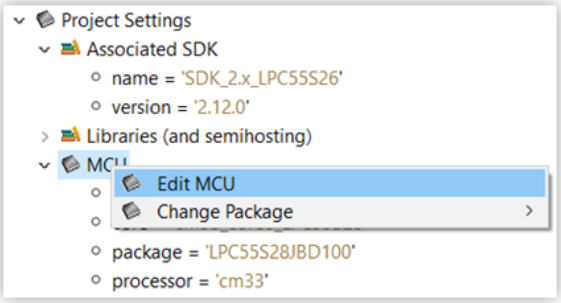


Figure 14.3. Edit MCU

You will then be presented with the MCU Setting dialogue (as below)

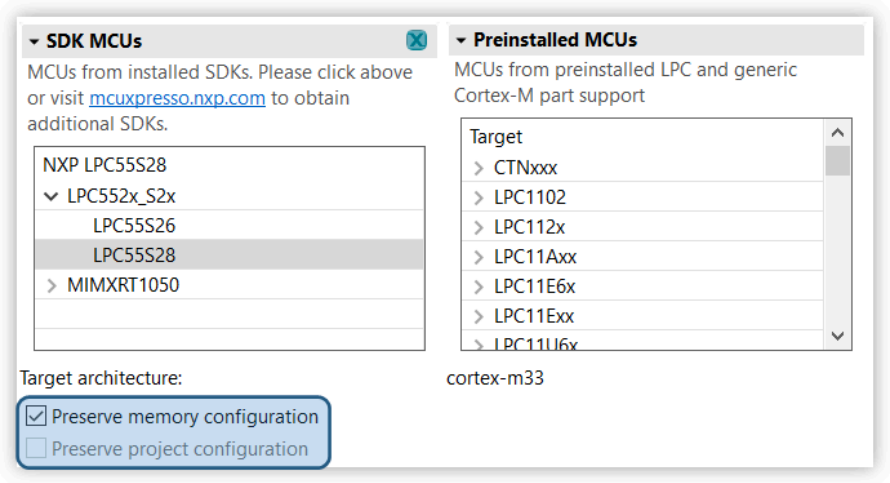


Figure 14.4. Select MCU

From here, an alternative MCU can be selected but note, there are two check boxes that must be set as required **before** this is done:

- Preserve Memory Configuration – if set (the default), the original project memory settings will be preserved, otherwise the MCU setting for the chosen MCU will replace the original settings
- Preserve Project Configuration – if not set (the default), the new MCUs configurations (such as Cortex Architecture) will replace the original settings

When the new MCU is selected, a warning dialogue as below will be generated:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

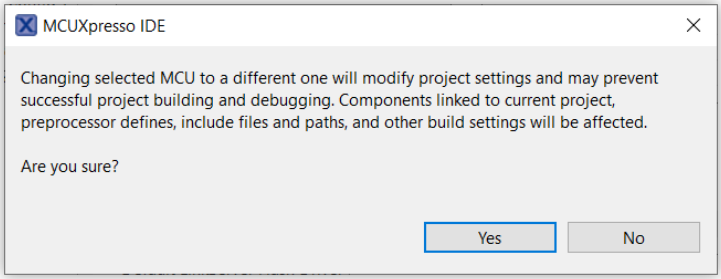


Figure 14.5. Select MCU Warning

Project changes will only be made if *Yes* is selected and *Apply* and/or *Apply and Close* are then further clicked to close the *Properties* dialog.

The actual changes that will be made inside the project depend on a few more user inputs asked before completing the entire process:

1. Confirm selection of the new board, new device package and new core associated with project.
2. Allow removal of the SDK components associated with the old MCU. Only SDK components that have an associated component to the new MCU are actually removed at this step.
3. Allow addition of SDK components associated with the new MCU. Only SDK components that had an associated component to the old MCU are actually added at this step.

Note: Backup the original project before initiating the change of device process.

14.3.1 Confirm device information

This step allows selection of the new board, new device package and new core. These selections depend on the SDK (or no SDK if dealing with pre-installed part support) that's associated with the new MCU. An MCU can be fitted on multiple boards, comes in different packages and might be a multi-core device.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

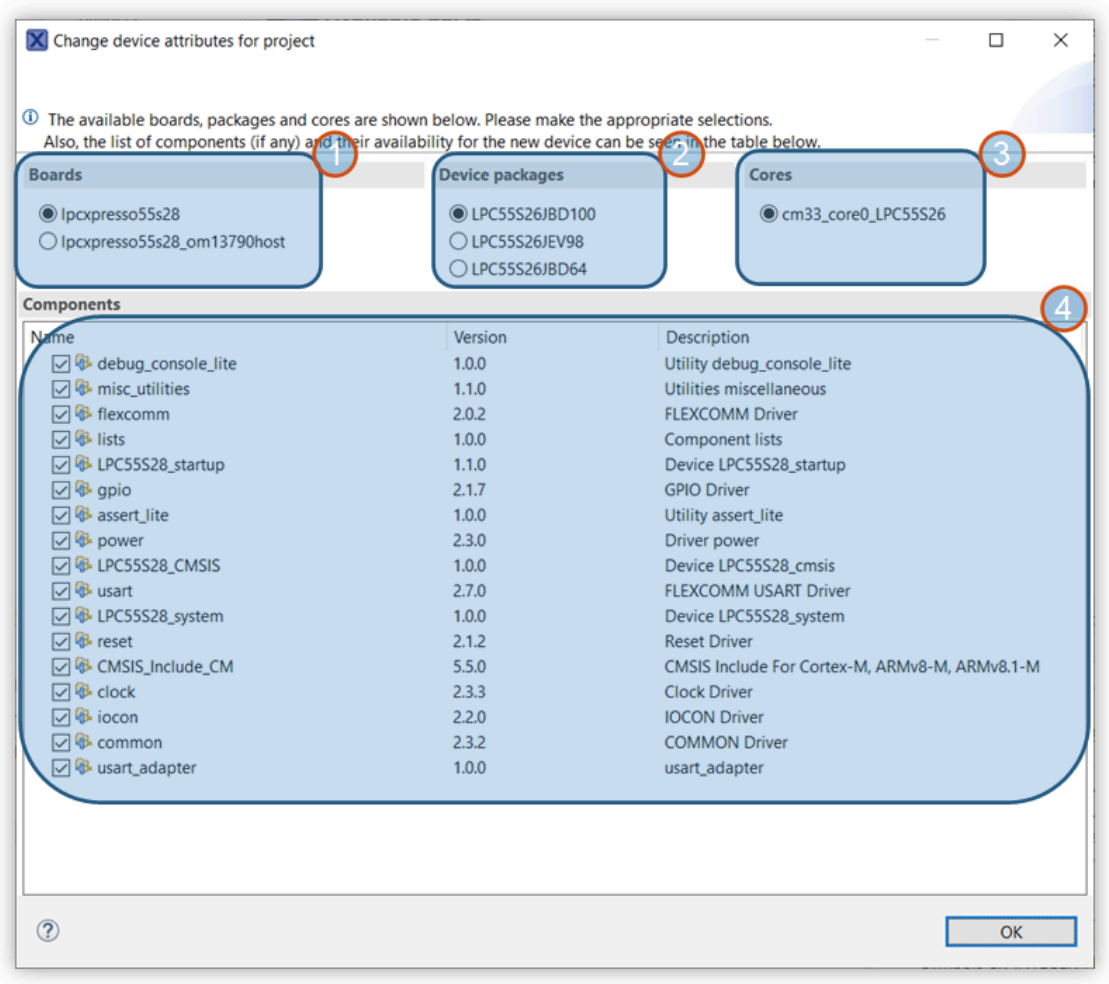


Figure 14.6. Change device attributes (same SDK)

In the above screenshot, the following sections can be highlighted:

1. Available boards
2. Available device packages
3. Available cores
4. The list of SDK components found inside the project. The table contains a read-only list of checkbox items. If an entry is ticked, it means that the SDK component associated with the old MCU has an associated SDK component for the new MCU. Tooltips offer more details about the old-to-new mapping. All SDK components that are selected inside the table, will be “migrated” (i.e. old components removed, new components added) during the process of changing the device.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

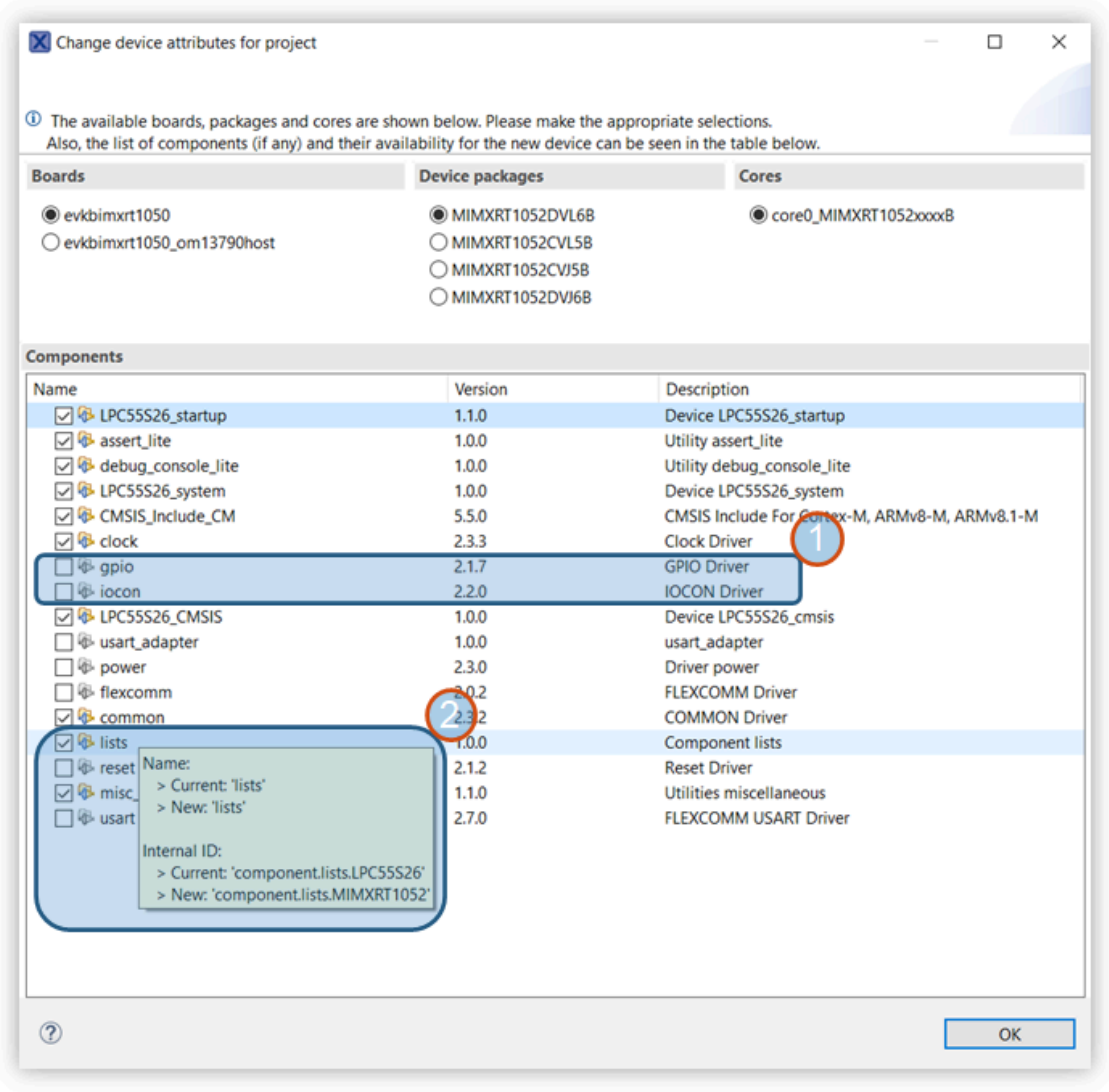


Figure 14.7. Change device attributes (different SDK)

We can see in the above screenshot some components that are not going to be migrated (1). This is because the IDE was unable to match the old internal SDK component ID to an ID associated with a new SDK component for the new MCU. The tooltips (2) offer some insights about the actual migration.

When switching from an SDK-supported part to a pre-installed part, all SDK components information will be lost from the project description. However, no source files will be removed/changed/added along the process.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

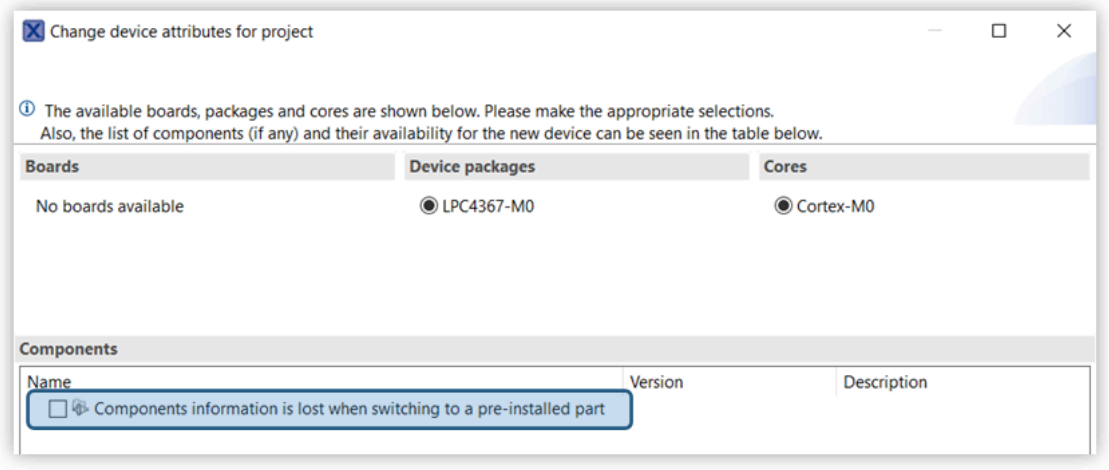


Figure 14.8. Change device attributes (pre-installed part)

14.3.2 Removal of SDK components associated with the old MCU

The following step requires confirmation on the removal of outdated SDK components (i.e. associated with the old MCU). Files listed in the dialog will be removed from project and also replaced by their counterpart associated with the new device. These are SDK-specific source files and no user changes are expected to be found inside. In this context, any change will be lost once removal of components is confirmed.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

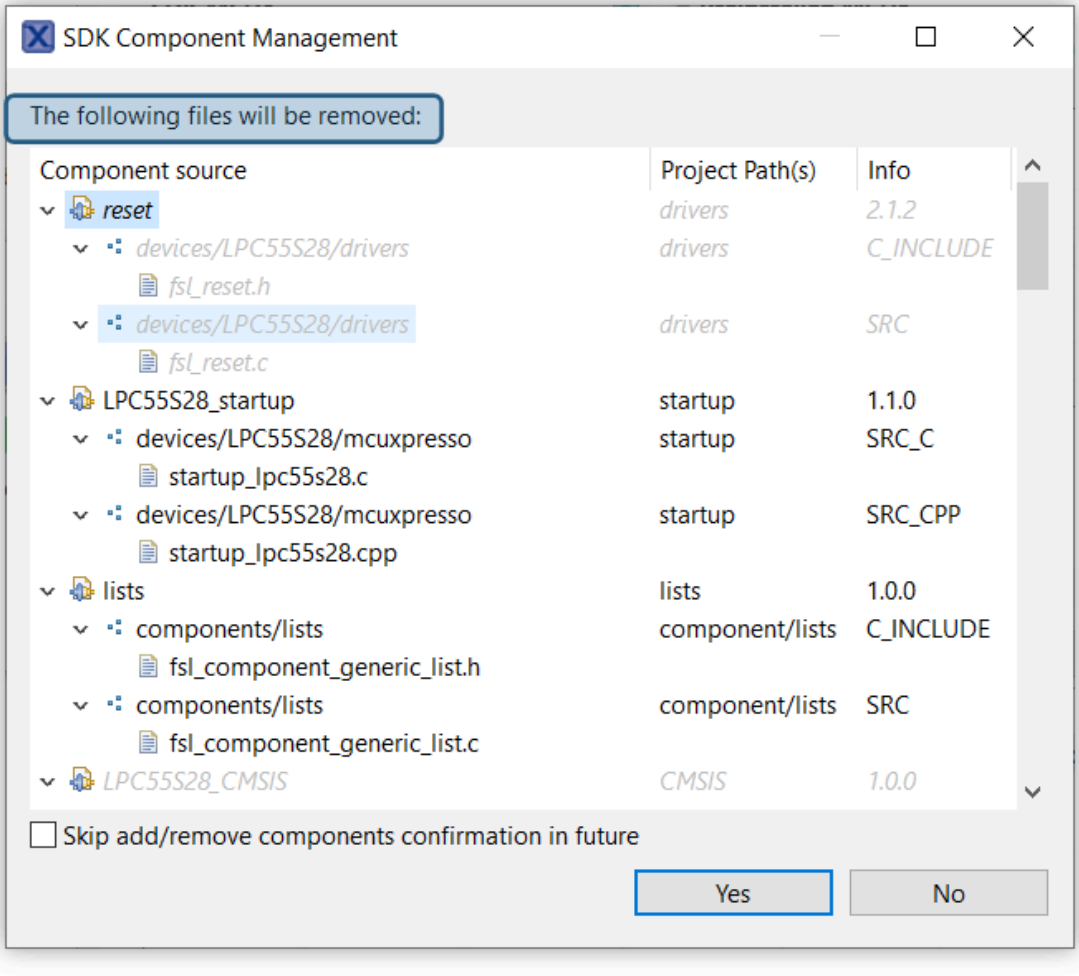


Figure 14.9. Confirm removal of outdated SDK components

14.3.3 Addition of SDK components associated with the new MCU

This is the last step of the process. At this point, components associated with the new MCU/SDK will be added to the project. Depending on the imported SDK type (zipped or unzipped), files can be copied inside the project or linked. Only unzipped SDKs allow linking of source files.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

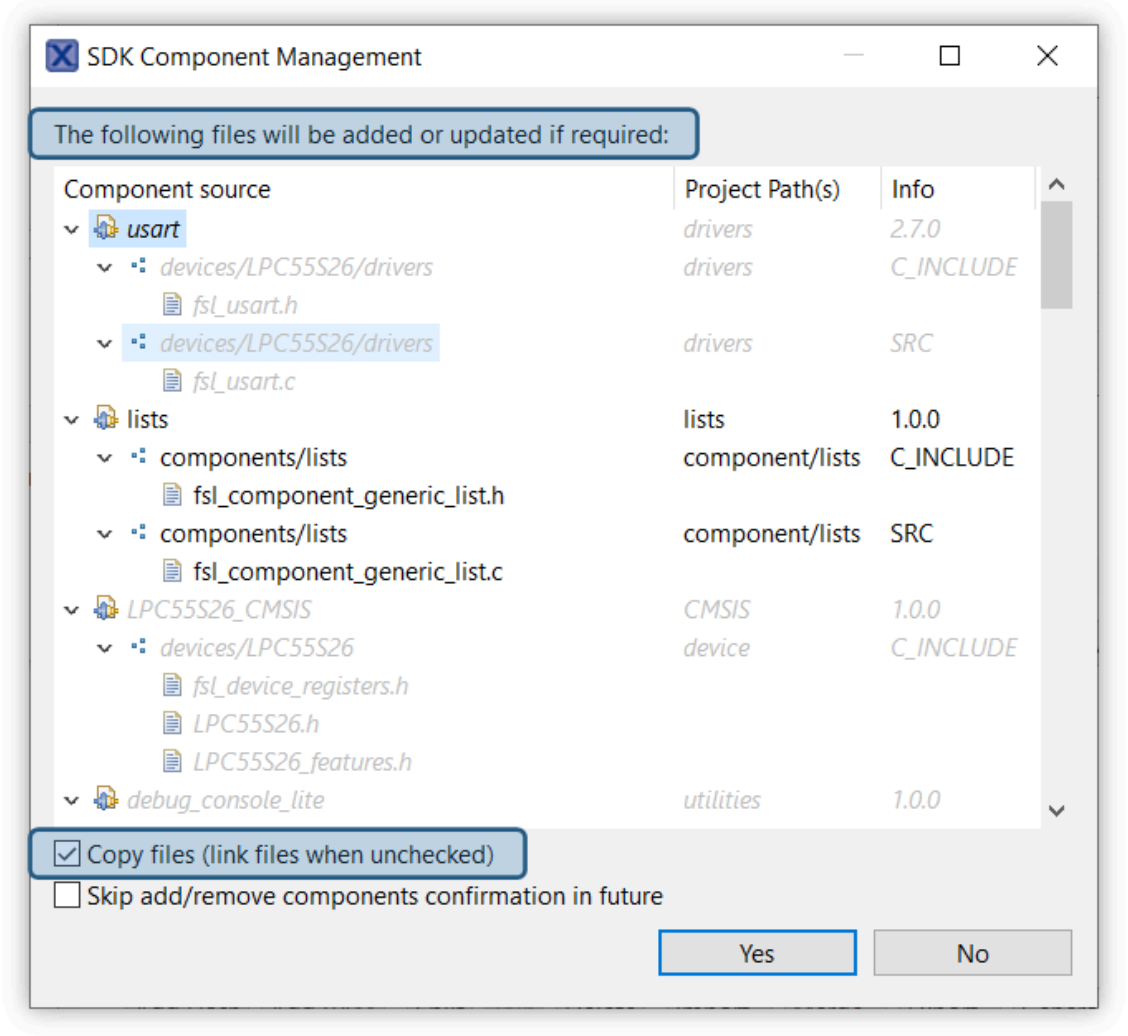


Figure 14.10. Confirm addition of new SDK components

14.4 Changing the MCU (SDK) package type

MCUs are commonly available in a range of package types. Different packages may impact the options available on the MCU external pins, for example the number of GPIO lines. MCUXpresso IDE makes no use of this package type however it is significant to the included MCUXpresso Config Tools [164] .

As shown in the previous section, from the project virtual nodes, select *Edit MCU*.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

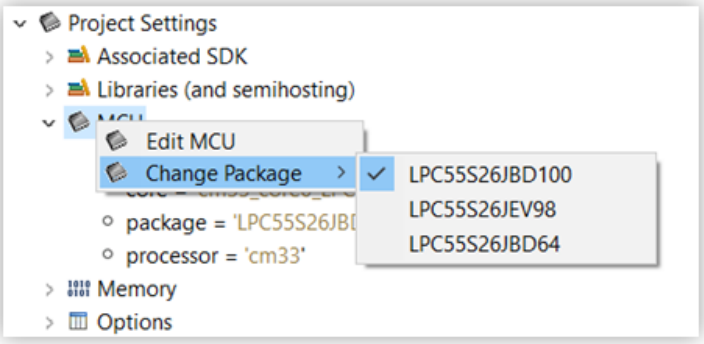


Figure 14.11. Edit Package

then select *Change Package* and choose the package required.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

15. MCUXpresso Config Tools

This chapter provides an introduction to the features of the MCUXpresso Config Tools installed by default with MCUXpresso IDE. The Config Tools present new perspectives in addition to the IDE's Develop and Debug perspectives.

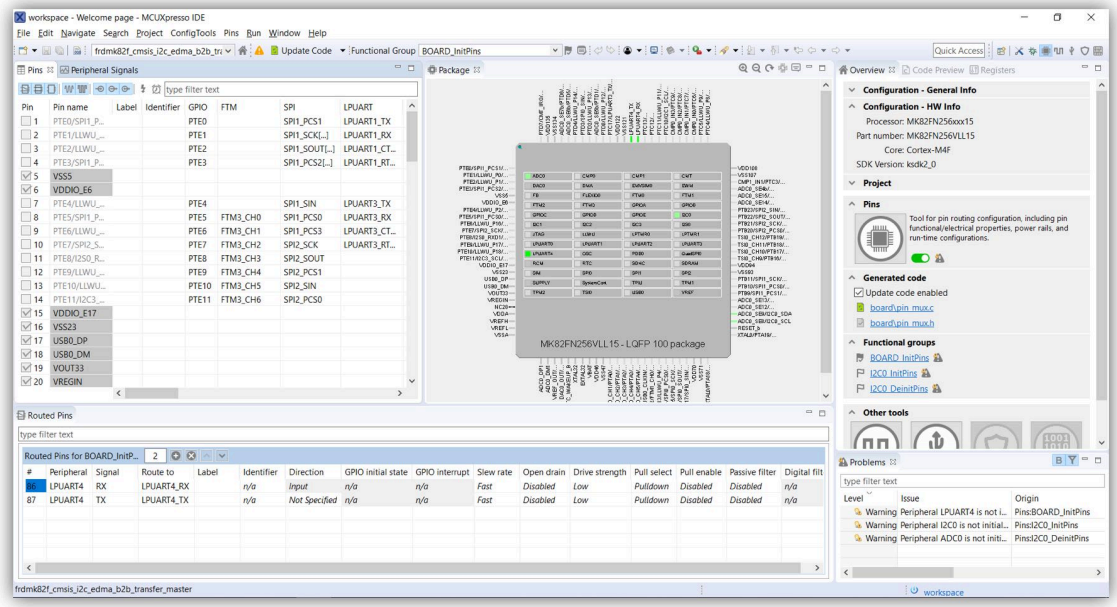


Figure 15.1. Config Tools Showing Pins Perspective

Please refer to the MCUXpresso IDE Config Tools User Manual for detailed information.

15.1 Using the Config Tools

MCUXpresso IDE includes the following Config Tools:

- Pins Tool
  - allows you to configure pin routing and generates 'pin_mux.c & .h' source files
- Clocks Tool
  - allows you to configure system clocks and generates 'clock_config.c & .h' source files
- Peripherals Tool
  - allows you to configure other peripherals and generates 'peripherals.c & .h' source files
- Device Configuration Tool
  - allows you to configure the initialization of memory interfaces of your device and generate dcd.d and dcd.h source files in C array or binary format
- TEE Tool
  - allows you to configure security policies of memory areas, bus masters, and peripherals, in order to isolate and safeguard sensitive areas of your application and generate tzm_config.c & .h source files.

MCUXpresso Config Tools can be used to review or modify the configuration of SDK example projects or new projects based on SDK 2.x. To open the tool, simply right click on the project in Project Explorer and select the appropriate Open command:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

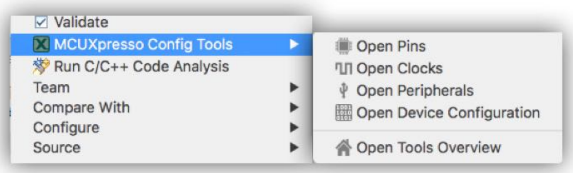


Figure 15.2. Config Tools Launch

If the project does not contains any configuration file (.mex) yet, it is automatically created by importing the existing source files (from YAML comments from pin_mux.c, clock_config.c and/or peripherals.c). If there are no source files in the project, a default configuration is created. The configuration is stored in the root of project folder with ".mex" file extension.

15.1.1 Tool Perspectives

Each tool is displayed in separate perspective. Once the configuration is opened, you can switch between perspectives to review/modify configuration of each tool – using the toolbar on the upper right part of the IDE window:



If your workspace contains multiple projects, please be aware that the MCUXpresso Config Tools only support one configuration to be opened at a time and that configuration must be opened explicitly for each project using the Open command from the popup menu. Switching perspectives does not switch the selected configuration.

15.1.2 Pins Tool

The Pins Tool allows you to display and configure the pins of the MCU. Basic configuration can be done in either of these views Pins, Peripheral Signals or Package. More advanced settings (pin electrical features) can be adjusted in Routed Pins view.

15.1.3 Clocks Tool

The Clocks Tool allows you to display and modify clock sources and outputs settings in Table view. More advanced settings can be adjusted via Diagram view and Details view. Global settings of the clocking environment such as run modes, MCG modes and SCG modes can be modified via main application toolbar.

15.1.4 Peripherals Tool

You can use the Peripherals tool to configure initialization of selected peripherals and generate code for them. In the Peripherals view, select the peripheral to configure and confirm addition of the configuration component. Then you can select the mode of the peripheral and configure the settings within the settings editor.

15.1.5 Device Configuration Tool

Device Configuration tool allows you to configure the initialization of memory interfaces of your device. Use the Device Configuration Data (DCD) view to create different types of commands and specify their sequence, define their address, values, sizes, and polls.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

15.1.6 TEE Tool

In the Trusted Execution Environment, or TEE tool, you can configure security policies of memory areas, bus masters, and peripherals, in order to isolate and safeguard sensitive areas of your application. You can set security policies of different parts of your application in the Security access configuration and its sub-views, and review these policies in the Memory map and Access overview views. Use the User Memory Regions view to create a convenient overview of memory regions and their security levels.

15.1.7 Generate Code

To update sources in the project, simply hit “Update Code” button on the toolbar. The command opens dialog with list of files that will be re-generated and allows to select which tools will generate the code.

Alternatively, it is also possible to export select source file by hitting export button in the Sources view.

15.1.8 SDK Components

Generated code uses the API of the SDK components to configure peripherals. SDK components missing in the IDE project are reported in problems view. It is possible to add component into IDE project by right click on the reported problem and selecting the proposed quick fix.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

16. The GUI Flash Tool

The GUI Flash tool provides flash programming capabilities for **all supported debug solutions**.

As well as implementing seamless programming of Flash when starting a debug session, MCUXpresso IDE enables the Flash programming capabilities of the supported debug solutions to be accessed directly, both via the GUI and from the command line (which might be useful for performing small production runs).

These flash programming capabilities can be accessed from three distinct places with the IDE.

Firstly, the most feature capable (advanced) variant is launched via the IDE button (and will be described in this section) :



Clicking this will launch a dialogue similar to:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

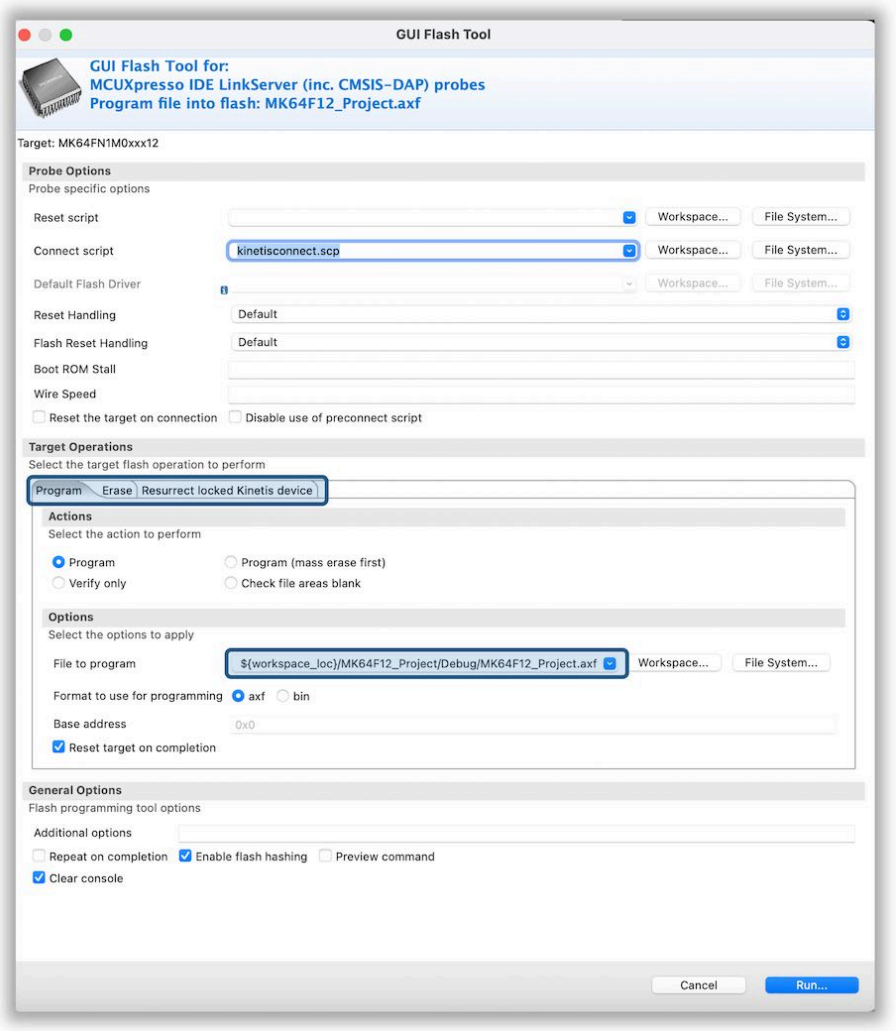


Figure 16.1. GUI Flash Tool

Note: This dialogue will vary subtly for each debug solution.

Secondly, project launch configurations now contain a GUI Flash Tool Tab providing project specific flash operations. Please see Debug Solutions Overview [94] for more information.

Finally, the **Quickstart** panel Debug Shortcuts provide easy access for simple project flash programming. Please see Debug Quickstart Shortcuts [123] for more information.



Tip

For Multicore MCUs, core selection is usually made automatically, but for GUI flash operations, it maybe necessary to take direct control of core selection, so this option is made available to the user.

16.1 The Advanced GUI Flash Tool

The operations below are supported for each debug solution.

1. Programming an .axf or .bin file into flash
2. Flash Mass Erase
3. Various debug solution specific features

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

When launched, each debug solution will present a dialogue similar to the LinkServer variant – described below:

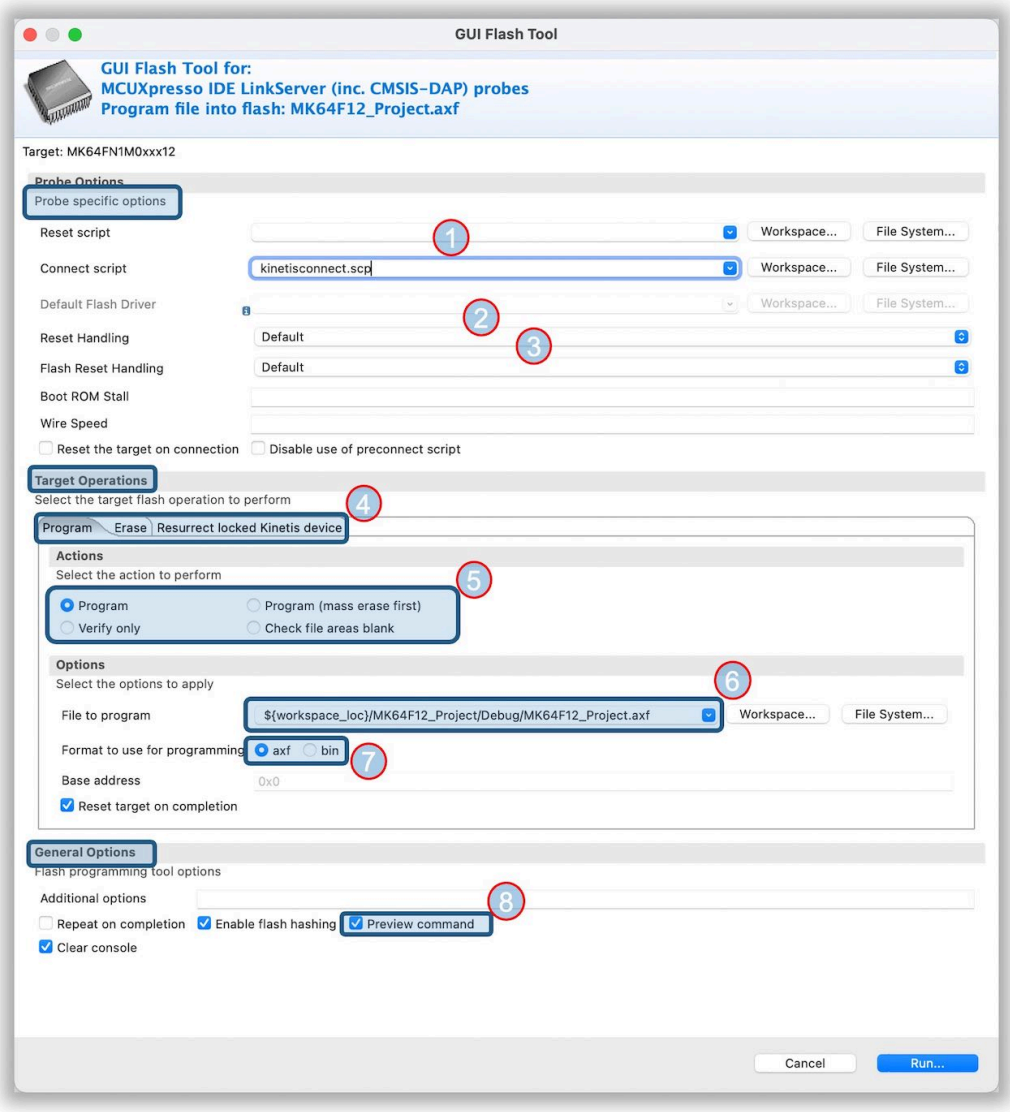


Figure 16.2. GUI Flash Tool major features

Note: Probe options (highlighted above) will be different for each debug solution, where as Target and General Options (also highlighted) will be broadly similar.



Tip

A project must first be selected before the Advanced GUI Flash Tool can be launched. The device and other project configurations (such as flash driver) will be inherited from this selected project. The advanced GUI Flash tool will not create or use information within project associated launch configurations.

1. Reset and Connect scripts: Any SDK specified Reset or Connect scripts will be automatically selected. A different script can be selected if required using the Workspace or File System shortcut buttons. If specified, a Reset script overrides the Reset Handling.
2. Reset Handling: The device default reset handling can be overridden from the selection: Default, SYSRESETREQ, VECTRESET, SOFT
3. Flash Reset Handling: The flash drivers default reset handling can be overridden from the selection: Default, SYSRESETREQ, VECTRESET, SOFT

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

4. Program/Erase/Resurrect locked Kinetis Device
   - **Program view** (displayed) should be selected to program an application of binary into flash. Only the Program options will be described below.
   - **Erase view** should be selected for options to erase a flash device to its blank state
     - offers options to Mass erase, Erase by sector, Check blank (to verify a blank flash).
     - generally flashes do not need to be erased, since program operations automatically erase sections of the flash as required. However on occasion it can be useful to erase a flash most often because the image in flash is causing problems.
     - Erase by sector is not recommended for Kinetis parts since this will leave the device fully erased and therefore in a locked state – should this occur, use the option below ...
   - **Resurrect locked Kinetis device** view should be selected to recover a *locked* device.
5. Programming actions:
   - **Program:** the default action will program the selected application or binary erasing only the required sections of the flash device.
   - **Program (mass erase first):** will erase the whole device before program the selected application or binary. This will ensure that any previous flash contents are erased.
   - **Verify only:** this option will compare the contents of flash with the selected application or binary. **Note**: most flash programming operations are verified at the programming stage. Flash contents are not changed.
   - **Check file area blank:** this can be used to verify that a program operation will not overwrite any data already programmed into flash. Flash contents are not changed.
6. File selection: if the selected project contains a built .axf file, then this will automatically be selected. Alternatively a different file can be selected using the Workspace and File System shortcut buttons.
7. Format: these radio buttons will be preset by the *File to Program* type. However, if an .axf file is selected, clicking bin will automatically generate a .bin from the selected .axf.
   - for file types containing no base address information, such as .bin, a base address must be specified.
8. Preview command: select this option to be presented with a preview programming command to be issued and a script that can perform this action independently of the IDE (see below)
   - the previewed command can be edited if required, changes will be reflected within the script. Various shell script *flavours* can be selected, and finally the script can be copied to the clipboard with a single click

Finally, click *Run* to execute the flash programming operation, a dialogue displaying the success of the operation will be displayed once the program operation has completed.

16.1.1 Advanced GUI Flash Tool command Preview

As discussed in point 8 above, the GUI Flash Tool can optionally display the command to be issued – allowing the opportunity of editing the command before execution.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

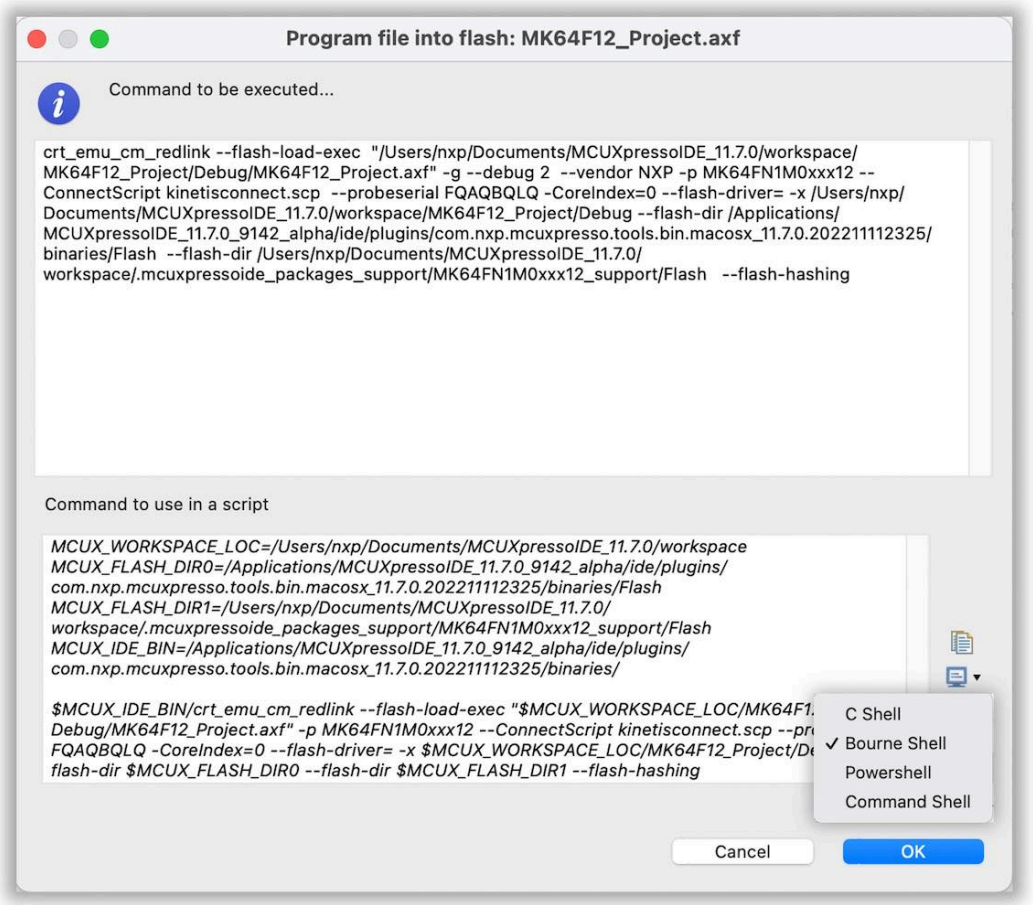


Figure 16.3. GUI Flash Tool Command Preview

In addition to displaying the command to be issued, the dialogue also contains a script that can be issued independently of the IDE to perform the flash programming operation. Changes the command to be executed will also be reflected within the script.

Notes

- The script will setup the local environment to be independent of your local shells configuration. However components of MCUXpresso IDE are of course referenced so the script can only be used if MCUXpresso IDE is installed and any referenced workspace files are present.
- Debug probes may install drivers when **first** seen by a host, this driver installation may take some time to complete.
- MCUXpresso IDE is able to maintain connection to multiple debug probes, while the IDE can dynamically maintain knowledge of connected probes, any generated command line will be a snapshot of a given instance. Therefore it is essential that only a single debug probe is connected if the command script is to be captured for re-use.
- Typically, LPC-Link2 or LPCXpresso V2 and V3 boards have debug probe firmware soft loaded automatically by the IDE when a debug operation is first performed. Therefore to use these debug probes from the command line they must either have their firmware softloaded or have probe firmware programmed into the Flash. Probe firmware can be soft-loaded from the command line by use of scripts *boot_link1* for LPC-Link and *boot_link2* for LPC-Link2, these are located at *mcuxpresso_install_dir/ide/binaries*. To program debug probe firmware into the Flash memory of an LPC-Link2 debug probe, please see: https://www.nxp.com/LPCSCRYPT

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

16.1.2 Advanced GUI Flash Tool logged Output

When a GUI Flash Tool operation is performed, the low level output will be logged into the debug log. A snippet of a LinkServer successful program operation is shown below:

```
...
Loading 'MK64FN1M0xxx12_Project.axf' ELF 0x00000000 len 0x3CF8
Opening flash driver FTFE_4K.cfx (already resident)
Sending VECTRESET to run flash driver
Writing 15608 bytes to address 0x00000000 in Flash
1 of 1 (  0) Writing pages 0-3 at 0x00000000 with 15608 bytes
(  0) at 00000000: 0 bytes - 0/15608
( 26) at 00000000: 4096 bytes - 4096/15608
( 52) at 00001000: 4096 bytes - 8192/15608
( 78) at 00002000: 4096 bytes - 12288/15608
(100) at 00003000: 4096 bytes - 16384/15608
Erased/Wrote page  0-3 with 15608 bytes in 693msec
Closing flash driver FTFE_4K.cfx
(100) Finished writing Flash successfully.
Flash Write Done
Loaded 0x3CF8 bytes in 1081ms (about 14kB/s)
Reset target (system)
Starting execution using system reset
```

16.1.3 Advanced GUI Flash Tool Programming an arbitrary Binary

The GUI Flash tool will usually be used to program a binary generated from a Project's .axf file. However on occasion, it might be required to program a binary (or .axf) file generated elsewhere. This can be achieved by generating a project with the required memory/chip combination and simply dropping the .bin file into this project. When the GUI Flash tool is invoked, the user can browse for the required binary file and program this in the usual way.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

17. LinkServer Flash Support

LinkServer (CMSIS-DAP) Flash drivers are used by LinkServer debug connections only. Please refer to the section on LinkServer Debug [98] for details of the LinkServer debug solution.

MCUXpresso IDE's LinkServer based debug connections makes use of a RAM loadable Flash driver mechanism. Such a Flash driver contains the knowledge required to program the **internal Flash** on a particular MCU (or potentially, family of MCUs). This knowledge may be either hardwired into the driver, or some of it may be determined by the driver as it starts up (typically known as a 'generic' Flash driver).

At the time a debug connection is started by MCUXpresso IDE, a LinkServer debug session running on the host will typically download a Flash driver into RAM on the target MCU. It will then communicate with the downloaded Flash driver via the debug probe in order to program the required code and data into the Flash memory.

In addition, the loadable Flash driver mechanism also provides the ability to support Flash drivers which can be used to program external Flash memory (for instance via the SPIFI Flash memory interface on LPC18x, LPC40xx, LPC43xx, LPC5460x and iMXRT families). The sources for some of these drivers is provided in the *Examples/Flashdrivers* subdirectory within the MCUXpresso IDE installation directory.

Note: Quad SPI (QSPI) and SPIFI are used interchangeably within this section. The term SPIFI (SPI Flash Interface) is commonly used to reference LPC use of QSPI.

LinkServer Flash drivers have a .cfx file extension. For Preinstalled MCUs, the Flash driver used for each part/family will be located in the */binaries/Flash* subdirectory of the MCUXpresso IDE installation. For SDK installed MCUs, the Flash driver will generally be supplied within the SDK, although copies may also provided in the */binaries/Flash* subdirectory.

Important Note: LinkServer flash drivers are fully integrated into the MCUXpresso IDE Managed Linkerscript build mechanism and specified within SDK metadata. Other debug solutions invoke MCU specific flash programming strategies based on their debug implementation's knowledge of the MCU being debugged.

17.1 Default vs Per-Region Flash Drivers

By default, for legacy reasons, Preinstalled MCUs are configured to use what is called a 'Default' Flash driver. This means that this Flash driver will be used for all Flash memory blocks that are defined for that MCU (i.e. as displayed in the Memory Configuration Editor).

For most users, there is never any need to change the automatically selected Flash driver for the MCU being programmed.

However, MCUXpresso IDE also supports the creation and programming of projects that span multiple Flash devices. In order to allow this to work, Flash drivers can also be specified per memory region.

For example, this allows a project based on an LPC43xx device with internal Flash to also make use of an external SPIFI Flash device. This is achieved by removing the default Flash driver from the memory configuration and instead explicitly specifying the Flash driver to use for each Flash memory block (per-region Flash drivers). A typical use case could be to create an application to run from the MCU's internal Flash that makes use of static constant data (e.g. for graphics) stored in external SPIFI device. An example memory configuration is shown below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

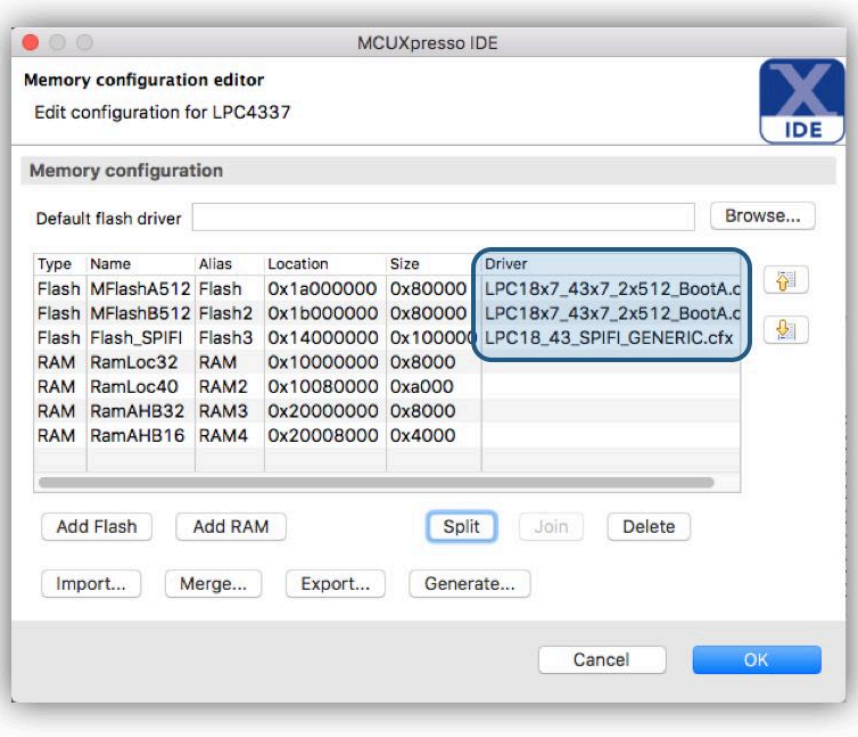


Figure 17.1. Per Region Drivers

Note: SDK installed MCU support always use Per-Region Flash drivers.

17.2 Advanced Flash Drivers

Most wizard generated projects or projects imported from SDKs (or LPCOpen) will be pre-configured with an appropriate LinkServer flash driver for the target flash device. As a result, in many cases users need to pay little attention to the actual flash driver being used. However, for MCUs supporting complex flash strategies or external flash devices, the situation is more complex. This section discusses these situations but note, even in these cases, the flash driver may be automatically selected and so require no user attention.

17.2.1 LPC18xx / LPC43xx Internal Flash Drivers

A number of LPC18/43 parts provide dual banks of internal Flash, with bank A starting at address 0x1A000000, and bank B starting at address 0x1B000000.

```
* LPC18x3/LPC43x3 : Flash = 2x 256KB (512 KB total)
* LPC18x5/LPC43x5 : Flash = 2x 384KB (768 KB total)
* LPC18x7/LPC43x7 : Flash = 2x 512KB ( 1 MB total)
```

When you create a new project using the New Project Wizard for one of these parts, an appropriate default Flash driver (from *LPC18x3_43x3_2x256_BootA.cfx* / *LPC18x5_43x5_2x384_BootA.cfx* /*LPC18x7_43x7_2x512_BootA.cfx*) will be selected which after programming the part will also configure it to boot from Bank A Flash.

If you wish to boot from Bank B Flash instead, then you will need to manually configure the project to use the corresponding “BootB” Flash driver (*LPC18x3_43x3_2x256_BootB.cfx* / *LPC18x5_43x5_2x384_BootB.cfx* / *LPC18x7_43x7_2x512_BootB.cfx*). This can be done by selecting the appropriate driver file in the “Flash driver” field of the Memory Configuration Editor. **Note**: you will also need to delete Flash Bank A from the list of available memories (or at least reorder so that Flash Bank B is first).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

17.2.2 LPC SPIFI QSPI Flash Drivers

A number of parts provide support for external SPIFI Flash, sometimes in addition to internal Flash. Programming these Flash memories provides a number of challenges because the size of memory (if present) is unknown, and the actual memory device is also unknown. These issues are handled using *Generic Drivers* which can interrogate the memory device to find its size and programming requirements.

At the time of writing, these LPC devices comprise:

Table 17.1. SPIFI details

LPC Part	SPIFI Address	Bootable	Flash Driver
LPC18xx/LPC43xx	0x14000000	Yes	LPC18_43_SPIFI_GENERIC.cfx
LPC40xx	0x28000000	No	LPC40xx_SPIFI_GENERIC.cfx
LPC5460x	0x10000000	No	LPC5460x_SPIFI_GENERIC.cfx
LPC540xx	0x10000000	Yes	LPC540xx_SPIFI_GENERIC.cfx

During a programming operation, the Flash driver will interrogate the SPIFI Flash device to identify its configuration. If the device is recognised, its size and name will be reported in the MCUXpresso IDE Debug log - as below:

```
...
Inspected v.2 External Flash Device on SPI using SPIFI lib LPC18_43_SPIFI_GENERIC.cfx
Image 'LPC18/43 Generic SPIFI Mar  7 2017 13:14:25'
Opening flash driver LPC18_43_SPIFI_GENERIC.cfx
flash variant 'MX25L8035E' detected (1MB = 16*64K at 0x14000000)
...
```

Note: Although the Flash driver reports the size and location of the SPIFI device, the IDE's view of the world is determined by the project memory configuration settings. It remains the users responsibility to ensure these setting match the actual device in use.

Flash devices supported by our LPC SPIFI Flash Drivers

The paragraph below contains information that is largely deprecated – please see the section Flash Drivers using SFDP [177]

Below is a list of SPIFI Flash devices supported by our supplied Generic SPIFI Flash drivers. **Note**: additional devices which identify as one of the devices below are also expected to work. However if a device is not supported by our supplied Flash Drivers, sources to generate these drivers are supplied in the *Examples/Flashdrivers* subdirectory within the MCUXpresso IDE installation directory. Users may thus add support for new SPIFI devices if needed.

```
GD25Q32C
MT25QL128AB
MT25Q512A
MT25Q256A
N25Q256
N25Q128
N25Q64
N25Q32
PM25LQ032C
MX25L1606E
MX25L1635E
MX25L3235E
MX25R6435F
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
MX25L6435E
MX25L12835E
MX25V8035F
MX25L8035E
S25FL016K
S25FL032P
S25FL064P
S25FL129P 64kSec
S25FL129P 256kSec
S25FL164K
S25FL256S 64kSec
S25FL256S 256kSec
S25FL512S
W25Q40CV
W25Q32FV
W25Q64FV
W25Q128FV
W25Q256FV_Untested
W25Q80BV
```

17.2.3 i.MX RT QSPI and Hyper Flash Drivers

I.MX.RT MCUs support external flash via a QSPI/Hyperbus interface, a range of LinkServer flash drivers supporting devices fitted to EVK development boards are included with MCUXpresso IDE (as described below).

Note: these drivers are also supplied in source project form so they may be used as a base for development of drivers for other external flash parts. These driver projects can be found at *Examples/Flashdrivers/NXP/iMXRT*

Table 17.2. Flash details

iMX RT Part	Base Address	Bootable	Flash Driver
i.MX RT 1050	0x60000000	Yes	MIMXRT1050-EVK_S26KS512S.cfx
i.MX RT 1050	0x60000000	Yes	MIMXRT1050-EVK_IS25WP064A.cfx
i.MX RT 1050	0x60000000	Yes	MIMXRT1050-EcoXiP_ATXP032.cfx
i.MX RT 1020	0x60000000	Yes	MIMXRT1020-EVK_IS25LP064.cfx

When used with the approriate SDK for your development board, the correct driver will be automatically selected

Important Note: For an application to Boot and execute in place (XIP) from these flash devices (post reset), a correct header for the specific device **MUST be programmed into the flash (as part of the Project)**. SDK examples will build to include an appropriate header automatically however, MCUXpresso IDE will not prevent users programming projects without headers into these devices. If this occurs the application will not boot and subsequent flash programming operations may fail.

Should this occur, the recommended recovery procedure is to change the boards boot strategy (via DIP switches) to prevent booting from QSPI or hyperflash. Power cycle the board and then perform a Mass Erase of the flash. Next, reprogram with an image that has appropriate header, restore the boot strategy and power cycle again.



Tip

In addition, these drivers are complemented by a range of self configuring drivers supporting all current iMX RT EVK boards , please see Flash Drivers using SFDP protocol [177] for more information on the drivers and this methodology.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

17.2.4 Flash Drivers using SFDP (LPC and iMX RT)

As discussed above, *the programming these Flash memories provides a number of challenges because the size of memory (if present) is unknown, and the actual memory device is also unknown*

LinkServer **Generic** flash drivers attempted to solve this problem by recognising specific devices (via their JEDEC ID) and then setting their sizes and programming parameters accordingly. However, this mechanism only works if the device is recognised by the flash driver, and in consequence will fail if any device is not recognised.

This issue, combined with the sheer volume of devices available has forced a different approach to be taken. Fortunately, modern flash devices typically contain a data block describing their properties including device size, low level structure and programming details etc. These data blocks and their use are collectively known as *Serial Flash Discovery Protocol* or SFDP. The standard for these blocks are described by JEDEC JESD216 standard(s).

Introduced in MCUXpresso IDE version 10.2.0 are a range of Generic flash drivers built to self configure via SFDP data and these have been extended for later MCUXpresso IDE versions. The current list of supported SFDP drivers is shown below:

Table 17.3. SFDP Flash details

Part	Base Address	Bootable	Flash Driver
LPC18xx/LPC43xx	0x14000000	Yes	LPC18_43_SPIFI_SFDP.cfx
LPC546xx	0x10000000	No	LPC546xx_SPIFI_SFDP.cfx
LPC540xx	0x10000000	Yes	LPC540xx_SPIFI_SFDP.cfx
LPC55S36	0x10000000	Yes	LPC553x_FlexSPI_A_MXIC_OPI.cfx
i.MX RT 1170	0x30000000	Yes	MIMXRT1170_SFDP_MXIC_OPI.cfx
i.MX RT 1170	0x30000000	Yes	MIMXRT1170_SFDP_QSPI.cfx
i.MX RT 1160	0x30000000	Yes	MIMXRT1160_SFDP_MXIC_OPI.cfx
i.MX RT 1160	0x30000000	Yes	MIMXRT1160_SFDP_QSPI.cfx
i.MX RT 1064	0x70000000	Yes	MIMXRT1064.cfx
i.MX RT 1060	0x60000000	Yes	MIMXRT1060_SFDP_HYPERFLASH.cfx
i.MX RT 1060	0x60000000	Yes	MIMXRT1060_SFDP_QSPI.cfx
i.MX RT 1050	0x60000000	Yes	MIMXRT1050_SFDP_HYPERFLASH.cfx
i.MX RT 1050	0x60000000	Yes	MIMXRT1050_SFDP_QSPI.cfx
i.MX RT 1024	0x60000000	Yes	MIMXRT1024.cfx
i.MX RT 1020	0x60000000	Yes	MIMXRT1020_SFDP_QSPI.cfx
i.MX RT 1015	0x60000000	Yes	MIMXRT1015_SFDP_QSPI.cfx
i.MX RT 1010	0x60000000	Yes	MIMXRT1010_SFDP_QSPI.cfx
i.MX RT 600	0x8000000	Yes	MIMXRT600_FlexSPI_A_MXIC_OPI.cfx
i.MX RT 600	0x8000000	Yes	MIMXRT600_FlexSPI_A_SFDP_QSPI.cfx
i.MX RT 600	0x8000000	Yes	MIMXRT600_FlexSPI_B_MXIC_OPI.cfx
i.MX RT 600	0x8000000	Yes	MIMXRT600_FlexSPI_B_SFDP_QSPI.cfx
i.MX RT 500	0x8000000	Yes	MIMXRT500_SFDP_MXIC_OSPI.cfx
i.MX RT 500	0x8000000	Yes	MIMXRT500_SFDP_QSPI.cfx
PN7640	0x218000	Yes	PN76xx.cfx

Important Note: for some iMX RT parts, the current SDKs reference the device specific flash driver rather than the SFDP version. However you can modify your project to use the SFDP version if required. Flashdrivers cannot detect whether QSPI or Hyperflash is fitted on a board, therefore it is the responsibility of the user to ensure the correct driver is used.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: The iMX RT 1024 and 1064 MCUs incorporate a flash device within the MCU package itself however, the flash driver still uses the SFDP mechanism to detect the device and hence is listed in the table above.

QSPI SFDP issues and Limitations

Some (usually older) QSPI parts do not support the SFDP mechanism and therefore will not be programmable via this protocol. However since some of these QSPI devices are fitted to NXP (LPC) manufactured development boards, some basic assumptions are made by these drivers if SFDP data is not found. In such a case, the device and its size will be assumed to be 1MB and some standard programming mechanisms will be used. This scheme should ensure that NXP LPC development boards with QSPI can be used with this driver type.

Note: this information is correct at the time of writing and only applies to LPC Drivers – future development of these drivers may change their capabilities.

Flash programming log

When programming code or data into flash, a portion of the debug log will display the flash programming operations (as below):

```
Inspected v.2 External Flash Device on SPI using SFDP JEDEC ID LPC18_43_SPIFI_SFDP.cfx –(1)
Image 'LPC1843_JEDEC_SFDP May  1 2018 15:32:05'
Opening flash driver LPC18_43_SPIFI_SFDP.cfx   ----------------------------------------(2)
Sending VECTRESET to run flash driver
flash variant 'JEDEC_SFDP_EF4014' detected (1MB = 16*64K at 0x14000000) ---------------(3)
Closing flash driver LPC18_43_SPIFI_SFDP.cfx
NXP: LPC43S37
Connected: was_reset=true. was_stopped=false
Awaiting telnet connection to port 3330 ...
GDB nonstop mode enabled
Opening flash driver LPC18_43_SPIFI_SFDP.cfx (already resident) ----------------------(4)
Sending VECTRESET to run flash driver
Writing 1046900 bytes to address 0x14000000 in Flash ---------------------------------(5)
Erased/Wrote page  0-15 with 1046900 bytes in 7548msec -------------------------------(6)
Closing flash driver LPC18_43_SPIFI_SFDP.cfx
Flash Write Done
Flash Program Summary: 1046900 bytes in 7.55 seconds (135.45 KB/sec) ------------------(7)
Stopped: Breakpoint #1
```

Note: when accessing unknown flash devices, the driver will be called twice. First to identify the device and secondly to perform the required programming. In a situation where multiple devices are being programmed, the flash driver(s) may be (re)loaded for each use.

Where:

1. *SFDP JEDEC ID* is the method used to access the flash and *LPC18_43_SPIFI_SFDP.cfx* is the flash driver used
2. the driver named above is loaded and initialised (this step will setup clocks, pin muxing, and perform some investigation of the connected device)
3. the driver returns a string *JEDEC_SFDP* indicating that SFDP data was found and successfully read
   - the devices JEDEC ID was read as *EF4014*, in this case corresponding to a Winbond 25Q80DVSIG (as fitted to the LPC-Link2 board used in Target mode)
   - the devices size was read as 1MB divided up into 16 64KB Sectors/Blocks – these blocks are the erase size that will be used for programming and so any operation to program this flash must start on an address aligned to this 64KB size
4. the driver is opened a second time (without reloading since it remains from the previous call)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

5. the project that referenced this driver requested that 1046900 bytes of data were written to the address starting 0x14000000, as set within the project's memory configuration
6. the write operation is performed via 16 page writes
   - **Note**: this flash driver (like many LinkServer drivers) uses a virtual page size that is much larger than the actual flash device page size to optimise driver operation
7. finally, a summary of the operation is printed showing the flash programming performance

Note: If the driver fails to find SFDP data, it will attempt to program the device with standard routines. If this occurs, the size will be assumed to be 1MB and the flash variant will be reported as *ID* rather than *SFDP* as shown below:

```
flash variant 'JEDEC_ID_EF4014' detected (1MB = 16*64K at 0x14000000)
```

On occasion, some devices that report the same JEDEC ID will actually be different, in this particular case the device is a very similar Winbond 25Q80BVSIG i.e. **..BV** rather than **..DV**

QSPI Programming and Booting

When dealing with external flash, it is important to understand the difference between the flash programming operation performed by the flash driver and the subsequent use of the flash for executing code and/or providing data. Essentially the flash drivers responsibility ends with a successful program operation, after this point, correct operation of the MCU/SPI flash combination lies elsewhere.

Thus, once the MCU is reset (or power cycled), the responsibility for the devices configuration and operation lie entirely outside of MCUXpresso IDE and instead lie with one or all of the following:

- development board/MCU boot settings
  - these may be DIP switches or Jumpers providing inputs to the MCU boot flow, alternatively these could be OTP bits programmed within the MCU
- MCU's BootROMs ability to understand and setup the device
  - BootROMs on devices such as the LPC1800 and LPC4300 have inbuilt understanding of certain QSPI devices allowing them to be configured for boot. However, this boot process may fail with some QSPI flash despite the fact that it has been correctly programmed
  - BootROMs on devices such as the LPC540xx and RT10xx rely on correct header (XIP) information being programmed (as part of the Application) into the QSPI flash itself. If this data is incorrect (or not present), the boot/reset will fail.
- Devices that incorporate both internal boot flash and external SPIFI/QSPI flash such as the LPC546xx typically place the responsibilities for QSPI configuration to the users application, where this might include
  - Setup of pinmuxing
  - QSPI/SPIFI clock setup
  - Flash interface initialisation
  - QSPI initialisation (this may be QSPI device specific)
    - including setup of appropriate waitstates for QSPI operation at the selected QSPI clock frequency

FlexSPI Flash reset

A number of IMX RT MCUs that support external flash via the FlexSPI interface implement a flash device reset sequence.

During FlexSPI boot the boot process requires the FlexSPI Flash device to be in a certain mode, for example, 1-bit SPI compatible mode. The Flash device will naturally be in this mode after a POR reset because the power-up sequence will reset it with the RT MCU device together. However, the Flash device will not be in 1-bit SPI compatible mode if the flash device is configured to DPI mode or QPI mode or Octal mode when any non-POR resets happen. In such case, special processing is required by the boot process to restore the Flash device to 1-bit SPI compatible mode before continuing access to the Flash device. In general, this can be

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

achieved by using a GPIO to assert a reset pin on the Flash device. The bootloader can perform the reset process and reset the Flash device to 1-bit SPI compatible mode based on fuses configuration, using the GPIO specified by the combination of FLEXSPI_RESET_PIN_PORT and FLEXSPI_RESET_PIN_GPIO.

When starting a flash-resident debug session in MCUXpresso IDE this reset sequence may need to be performed by the flash driver as well. Flash drivers for IMX RT500 and RT600 MCUs implement this functionality.

Note: Custom boards may not be wired identical to EVK development boards in regards to the actual pin dedicated to flash device reset. In such cases the pre-connect script needs to be modified in order to pass to the flash drivers the relevant information about the GPIO pin used for flash reset.

17.3 Kinetis Flash Drivers

Kinetis MCUs make use of a range of generic drivers, which are supplied as part of the SDK part support package. When a project is created or imported, the appropriate Flash driver is automatically selected and associated with the project.

Kinetis Flash drivers generally follow a simple naming convention i.e. **FTFx_nK_xx** where:

- FTFx is the Flash module name of the MCU, where x can take the value E, A or L
- nK represents the Flash sector size the Flash device supports, where n can take the value 1, 2, 4, 8
  - a sector size is the smallest amount of Flash that can be erased on that device
- xx represents an optional additional characters for special case drivers e.g. __Tiny for use on parts with a small quantity of RAM
  - a further optional _D suffix is used to show the driver is written to target Data Flash rather than the more common Program Flash

So for example a K64F MCU's Flash driver will be called *FTFE_4K*, because the K64F MCU uses the FTFE Flash module type and support a 4KB Flash sector size.

When a debug session is started that programs data into Flash memory, the IDE's debug log file will report the Flash driver used and parameters it has read from the MCU. Below we can see the driver identified a K64 part and the size of the internal Flash available. It also reports the programming speed achieved when programming this device. These logs can be useful when problems are encountered.

Note: when the Flash driver starts up, it will interrogate the MCU and report a number of data items. However, due to the nature of internal registers with the MCU, these may not exactly match the MCU being debugged.

```
Inspected v.2 On chip Kinetis Flash memory module FTFE_4K.cfx
Image 'Kinetis SemiGeneric Feb 17 2017 17:24:02'
Opening flash driver FTFE_4K.cfx
Sending VECTRESET to run flash driver
Flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)
Closing flash driver FTFE_4K.cfx
Connected: was_reset=true. was_stopped=true
Awaiting telnet connection to port 3330 ...
GDB nonstop mode enabled
Opening flash driver FTFE_4K.cfx (already resident)
Sending VECTRESET to run flash driver
Flash variant 'K 64 FTFE Generic 4K' detected (1MB = 256*4K at 0x0)
Writing 25856 bytes to address 0x00000000 in Flash
00001000 done  15% (4096 out of 25856)
00002000 done  31% (8192 out of 25856)
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
00003000 done  47% (12288 out of 25856)
00004000 done  63% (16384 out of 25856)
00005000 done  79% (20480 out of 25856)
00006000 done  95% (24576 out of 25856)
00007000 done 100% (28672 out of 25856)
Erased/Wrote sector  0-6 with 25856 bytes in 301msec
Closing flash driver FTFE_4K.cfx
Flash Write Done
Flash Program Summary: 25856 bytes in 0.30 seconds (83.89 KB/sec)
```

Flash drivers for a number of Kinetis MCUs are listed below:

```
K64F  FTFE_4K (1MB)
K22F  FTFA_2K (512KB)
KL43  FTFA_1K (256KB)
KL27  FTFA_1K (64KB)
K40   FTFL_2K (256KB)
```

17.4 Configuring projects to span multiple Flash Devices

https://community.nxp.com/thread/388979

17.5 The LinkServer GUI Flash Programmer

The LinkServer GUI Flash Programmer has been replaced by the debug solution independent GUI Flash Tool [167] .

17.6 The LinkServer Command Line Flash Programmer

While the information below is still current, for most users this functionality has been replaced by features within the The GUI Flash Tool [167] .

17.6.1 Command Line Programming

Flash programming is usually invoked automatically when a debug session is launched from within MCUXpresso IDE, but flash programming operations can also be accessed directly using a command line utility (also known as the LinkServer debug stub). This can be useful for things like programming the Flash for devices with limited production runs.

The MCUXpresso IDE Flash programming utility is located at:

```
<install_dir>/ide/binaries/
```

To run a Flash programming operation from the command line, the correct Flash utility stub for your part should be called with appropriate options. For boards containing Cortex-M MCUs the utility is called *crt_emu_cm_redlink*.

For example:

```
crt_emu_cm_redlink -p LPC11U68 --flash-load "LPC11U68_App.axf"
```

will load the AXF file LPC11U68_App.axf into Flash on an LPC11U68.

Note: typically, LPC-Link2 or LPCXpresso V2 and V3 boards have debug probe firmware soft loaded automatically by the IDE when a debug operation is first performed. Therefore to use these debug probes from the command line they must either have their firmware softloaded or

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

have probe firmware programmed into the Flash. Probe firmware can be soft-loaded from the command line by use of scripts *boot_link1* for LPC-Link and *boot_link2* for LPC-Link2, these are located at *mcuxpresso_install_dir/ide/binaries*. To program debug probe firmware into the Flash memory of an LPC-Link2 debug probe, please see: https://www.nxp.com/LPCSCRYPT

Programming an image into Flash

In the simplest case the Flash programming utility takes the following options if the file to be flashed is an AXF (or ELF) file:

```
crt_emu_cm_redlink -p target --flash-load "filename" [--flash-driver "flashdriver"]
```

it is also possible to flash binary files using:

```
crt_emu_cm_redlink -p target --flash-load "filename" --load-base base_address [--flash-driver /
  "flashdriver"]
```

Where:

- crt_emu_cm_redlink is the name of the Flash utility
- target is the target chip name. For example LPC1343, LPC1114/301, LPC1768 etc. (see 'Finding Correct Parameters...' below)
- --flash-load can actually be one of a few different options. Use:
  - --flash-load to write the file to Flash,
  - --flash-load-exec to write it to Flash and then cause it to start running,
  - --flash-mass-load to erase the Flash and then write the file to the Flash, and
  - --flash-mass-load-exec to erase the Flash, write the file to Flash and then cause it to start running.
- filename is the file to Flash program. It may be an executable (axf) or a binary (bin) file. If using a binary file, the base_address also must be specified. Using enclosing quotes is optional unless the name includes unusual characters or spaces.
- base_address is the address where the binary file will be written. It can be specified as a hex value with a leading 0x.

If you are using Flash memory that is external to the main chip you will need to specify an appropriate Flash driver that supports the device. This usually takes the name of a .cfx file held in a default location. In unusual circumstances it is possible to specify an absolute file system name of a file. Using enclosing quotes is optional unless the name includes unusual characters or spaces (see 'Finding Correct Parameters...' below).

WARNING: When crt_emu_cm_redlink Flash drivers program data that they believe will form the start of an execute-in-place image they determine where the image's vector table is and automatically inserts a checksum of the initial few vectors, as required in many LPC parts. This may not be the value held in that location by the file from which the Flash was programmed. This means that if the content of the Flash were to be compared against the file a difference at that specific location may be found.

WARNING: Flash is programmed in sectors. The sizes and distributions of Flash sectors is determined by the Flash device used. Data is programmed in separate contiguous blocks – there may be many contiguous blocks of data specified in an EFL (.AXF) file but there is only one in a binary file. When a contiguous data block is programmed into Flash data preceding the block start in its Flash sector is preserved. Data following data in the block in the final sector, however is erased.

Programming Flash with SDK Part Support

The above method works for parts supported with preinstalled part support. If SDK part support is required, then additional options must be passed to the utility.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- *sdk_parts_directory* - the place where the utility can find SDK part information; and
- *sdk_flash_directory* - the place where the utility can find Flash drivers provided by the SDK.

These are supplied to the utility by adding the following two options

```
-x "sdk_parts_directory" --flash-dir "sdk_flash_directory"
```

on to the command line already described. For example:

```
crt_emu_cm_redlink -p LPC54018 --flash-load "LPC54018_app.axf" \
  -x ~/mcuxpresso/01/.mcuxpressoide_packages_support/LPC54018_support \
  --flash-dir ~/mcuxpresso/01/.mcuxpressoide_packages_support/LPC54018_support/Flash
```

Since this is quite a lot to type you might wish to put the location of your SDK support directory into an environment variable as follows:

Windows:

```
set DIR_SDK ...\mcuxpresso\01\.mcuxpressoide_packages_support\LPC54018_support
crt_emu_cm_redlink -p LPC54018 --flash-load "LPC54018_app.axf" -x %DIR_SDK% \
  --flash-dir %DIR_SDK%\Flash
```

MacOS or Linux:

```
export DIR_SDK="~/.mcuxpresso/01/.mcuxpressoide_packages_support/LPC54018_support"
crt_emu_cm_redlink -p LPC54018 --flash-load "LPC54018_app.axf" -x $DIR_SDK \
  --flash-dir $DIR_SDK/Flash
```

Use “Finding Correct Parameters from MCUXpresso IDE”, below, to determine what values you require for these options.

Programming Flash taking MCUXpresso IDE project Memory edits into Account

MCUXpresso IDE allows the user to modify the default definition of the memory areas (including the specification of different named Flash regions) used in a hardware using the Edit... button found in the project’s properties at *C/C++Build -> MCU Settings* under the heading “Memory details”. The editor can create multiple named Flash regions.

In order to use these updates to the project’s part information the utility must use the directory where MCUXpresso IDE stores the project’s products for whatever configuration has been modified (typically the configuration will be called ‘Debug’) as the source of its part information.

To find the location of this directory in MCUXpresso expand the project in the Project Editor view, select the directory with the required configuration name (e.g. ‘Debug’), right click on it to bring up its properties and see the ‘Resource’ heading.

Supply this directory name as the sdk_parts_directory to the utility by adding the options:

```
-x "sdk_parts_directory"
```

Even if the part is supported by an SDK this will be the correct option to use for -x.

Programming Flash for complex debug connections

Some boards or chips occasionally need additional steps to occur before a stable debug connection can be established. Such debug connections are set up by small BASIC like programs called Connect Scripts. A good indication as to whether your chip or board normally requires a

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

connect script can be discovered when “Finding Correct Parameters from MCUXpresso IDE” (see below).

Connect scripts are distributed within the product and do not normally need to be written from scratch.

If a connect script is required it can be supplied by adding the following option to the command line already described:

--connectscript "connectscript"

In addtion to connect scripts, some chips also require a preconnect script that prepares the target MCU for the initial debug connection. A preconnect script can be supplied by adding the following option to the command line already described:

--preconnectscript "preconnectscript"

If you are using --flash-load-exec rather than --flash-load you may also find that the part that you are using requires its own “reset script” to replace the standard means of starting the execution of the flashed image. Again you may discover whether one is necessary as below. When required it can be supplied by adding the following option to the command line:

--resetscript "resetscript"

(As usual the quotes are required only if the script file name contains a space or other unprintable character.)

Finding the correct parameters from MCUXpresso IDE

Note: A simple way of finding the correct command and options is to use the GUI Flash Programmer described above, the completion dialog shows the exact command line invoked by the GUI. On this line the IDE will have chosen the correct

- target name
- a default Flash driver, *flashdriver*
- a connect script to be run, if needed
- a preconnect script to be run, if needed
- a reset script to be run, if needed with --flash-load-exec
- an sdk_parts_directory where XML information about the part being used (if it is provided via an SDK) can be found
- an sdk_flash_directory where flash drivers supporting the part being used (if it is provided via an SDK) can be found

Note: that the details will only appear and be relevant only if a project supporting the relevant chip or board is selected in the project explorer view.

For example the command line produced might be:

```
crt_emu_cm_redlink  "/Workspace/frdmk64f_driver_examples_blinky.axf" -g --debug 2 --vendor NXP \
  -p MK64FN1M0xxx12 -ProbeHandle=1 -CoreIndex=0 --ConnectScript kinetisconnect.scp   -x \
  /Users/nxp/mcuxpresso/01/.mcuxpressoide_packages_support/MK64FN1M0xxx12_support  --flash-dir \
  /Users/nxp/mcuxpresso/01/.mcuxpressoide_packages_support/MK64FN1M0xxx12_support/Flash
```

Looking at this the *target name* follows *-p*; the *flashdriver* follows *--flash-driver*; a *connectscript* follows *--connectscript*; a *resetscript* follows *--resetscript*; any *sdk_flash_directory* is provided following *--flash-dir* and any *sdk_parts_directory* is provided following *-x*.

If the target does not require a connect script or reset script the relevant options will not appear. If the project is not based on an SDK -x and --flash-dir do not appear.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Dealing with Errors during Flash operations

If your board requires a connect script to be run in order to provide a stable environment for Flash drivers you may see errors when you undertake a Flash operation without using it. You can use 'Finding Correct Parameters from MCUXpresso IDE', above, to check whether a connect script is required.

On some boards it is possible to run an image which is incompatible with the Flash driver (which crt_emu_cm_redlink runs on the target to help it manipulate a Flash device). This incompatibility is likely to show in the form of programming errors signalled as the operation progresses. Often they are due to unmaskable exceptions (such as watchdog timers) being used by the previous image that interfere with a Flash driver's operation.

There are a number of ways to address this situation:

- Does your board support In System Processing (ISP) Reset? Using it will usually reset the hardware and stop in the Boot ROM, thus ensuring a stable environment for Flash drivers. If present it can usually be activated with one or more on-board switches. You may have to refer to the board's documentation.
- Use the --vc option with crt_emu_cm_redlink. This option causes a reset when the utility's connection to the board's debug port is established. Most chips will be left having executed part of the Boot ROM and usually the resulting state is suitable for running a Flash driver. (There are exceptions however.)
- Erase the contents of Flash (see below) or program a (e.g. small) image that ensures no non-maskable exceptions are involved. Naturally these solutions have the problem that they are as likely to fail (and for the same reason) as the programming operation. It is sometimes the case that an incompatible image will allow the Flash drivers to operate for a short period in which there is a chance that one of these 'solutions' can be used.

Validating the Content of Flash

The Flash programming utility can validate the content of Flash programmed as an AXF (or ELF) file:

```
crt_emu_cm_redlink -p target --flash-verify "filename" [--flash-driver "flashdriver"]
```

it is also possible to verify binary files using:

```
crt_emu_cm_redlink -p target --flash-verify "filename" --load-base base_address \
  [--flash-driver "flashdriver"]
```

Where target and Flash driver have the same meaning as above.

For example:

```
crt_emu_cm_redlink -p LPC11U68 --flash-verify "LPC11U68_App.axf"
```

Note: the issues described in 'Dealing with Errors During Flash Operation' still apply when executing this command.

Erasing the Flash

The Flash programming utility can also delete the content of Flash. To do so it takes the following options:

```
crt_emu_cm_redlink -p target --flash-mass-erase [--flash-driver "flashdriver"]
```

Where target and Flash driver have the same meaning as above.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

For example:

```
crt_emu_cm_redlink -p LPC11U68 --flash-mass-erase
```

Note: the issues described in ‘Dealing with Errors During Flash Operation’ still apply when executing this command.)

Validating that Flash has been Erased

The Flash programming utility can validate that the content of Flash has been erased:

```
crt_emu_cm_redlink -p target --flash-check --area flash " [--flash-driver "flashdriver"]
```

For example:

```
crt_emu_cm_redlink -p LPC11U68 --flash-check --area flash
```

It is also possible to check that just the specific areas that would have been programmed by a given AXF or binary file are blank.

```
crt_emu_cm_redlink -p target --flash-check-file "filename" [--flash-driver "flashdriver"]
```

it is also possible to verify binary files using:

```
crt_emu_cm_redlink -p target --flash-check-file "filename" --load-base base_address \
  [--flash-driver "flashdriver"]
```

Where target and Flash driver have the same meaning as above.

For example:

```
crt_emu_cm_redlink -p LPC11U68 --flash-check-file "LPC11U68_App.axf"
```

Note: the issues described in ‘Dealing with Errors During Flash Operation’ still apply when executing this command.)

Examples

To load the binary executable file app.bin at location 0 on an LPC54113J128 target using LPC-Link2, use the following command line:

```
crt_emu_cm_redlink -p LPC54113J128 --load-base 0 --flash-load-exec app.bin
```

To load the executable file app.axf and start it executing on an LPC1768 target using LPC-Link2, use:

```
crt_emu_cm_redlink -p LPC1768 --flash-load-exec "app.axf"
```

To erase Flash, program the executable app.axf into an LPC18S37 board, which has no internal Flash but supports external Flash on the board, and then run it:

```
crt_emu_cm_redlink -p LPC18S37 --flash-mass-load-exec "app.axf" --flash-driver \
  LPC18x7_43x7_2x512_BootA.cfx
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

To erase then program app.axf into a Kinetis MK64FN1M0xxx12, which is supported through an SDK, and requiring a connect script (on MacOS/Linux):

```
crt_emu_cm_redlink -p MK64FN1M0xxx12 --flash-mass-load "app.axf" \
   --connectscript kinetisconnect.scp \
   -x ~/mcuxpresso/01/.mcuxpressoide_packages_support/MK64FN1M0xxx12_support \
   --flash-dir ~/mcuxpresso/01/.mcuxpressoide_packages_support/MK64FN1M0xxx12_support/Flash
```

To delete the Flash on an LPC1343:

```
crt_emu_cm_redlink -p LPC1343 --flash-mass-erase
```

To delete the Flash on an LPC54113J128 using vector catch to ensure that the currently booted code does not interfere with the Flash driver:

```
crt_emu_cm_redlink -p LPC54113J128 --flash-erase --vc
```

To check that the Flash is blank on an LPC54018 which is supported by an SDK and which has modified its memory layout stored in the MCUXpresso SDK example project held at ~/ws/lpcxpresso54018_driver_examples_gpio_gpio_led_output:

```
crt_emu_cm_redlink -p LPC54018 --flash-check -x \
  ~/ws/lpcxpresso54018_driver_examples_gpio_gpio_led_output/Debug \
  --flash-dir ~/mcuxpresso/01/.mcuxpressoide_packages_support/LPC54018_support/Flash
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

18. C/C++ Library Support

MCUXpresso IDE ships with three different C/C++ library families. This provides the maximum possible flexibility in balancing code size and library functionality.

18.1 Overview of Redlib, Newlib and NewlibNano

- **Redlib** Our own (non-GNU) ISO C90 standard C library, with some C99 extensions.
- **Newlib** GNU C/C++ library
- **NewlibNano** a version of the GNU C/C++ library optimized for embedded.

By default, MCUXpresso IDE will use Redlib for C projects, NewlibNano for SDK C++ projects, and Newlib for C++ projects for preinstalled MCUs.

Newlib provides complete C99 and C++ library support at the expense of a larger (in some cases, much larger) code size in your application.

NewlibNano was produced as part of ARM's "GNU Tools for ARM Embedded Processors" initiative in order to provide a version of Newlib focused on code size. Using NewlibNano can help dramatically reduce the size of your application compared to using the standard version of Newlib – for both C and C++ projects.

If you need a smaller application size and don't need the additional functionality of the C99 or C++ libraries, we recommend the use of Redlib, which can often produce much smaller applications.

18.1.1 Redlib extensions to C90

Although Redlib is basically a C90 standard C library, it does implement a number of extensions, including some from the C99 specification. These include:

- Single precision math functions
  - Single precision implementations of some of the math.h functions such as sinf() and cosf() are provided.
- stdbool.h
  - An implementation of the C99 stdbool.h header is provided.
- inttypes.h
  - An implementation of the C99 inttypes.h header is provided.
- itoa
  - itoa() is non-standard library function which is provided in many other toolchains to convert an integer to a string. To ease porting, an implementation of this function is provided, accessible via stdlib.h. More details can be found later in this chapter.

18.1.2 Newlib vs NewlibNano

Differences between Newlib and NewlibNano include:

- NewlibNano is optimized for size.
- The printf and scanf family of routines have been re-implemented in NewlibNano to remove a direct dependency on the floating-point input/output handling code. Projects that need to handle floating-point values using these functions must now explicitly request the feature during linking.
- The printf and scanf family of routines in NewlibNano support only conversion specifiers defined in C89 standard. This provides a good balance between small memory footprint and full feature formatted input/output.
- NewlibNano removes the now redundant integer-only implementations of the printf/scanf family of routines (iprintf/iscanf, etc). These functions now alias the standard routines.
- In NewlibNano, only unwritten buffered data is flushed on exit. Open streams are not closed.
- In NewlibNano, the dynamic memory allocator has been re-implemented

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

18.2 Library Variants

Each C library family is provided in a number of different variants : None, Nohost and Nohost-nf, Semihost and Semihost-nf (Redlib only). These variants each provide a different set of ‘stubs’ that form the very bottom of the C library and include certain low-level functions used by other functions in the library.

Each variant has a differing set of these stubs, and hence provides differing levels of functionality:

- **Semihost(-mb)**
  - This library variant provides implementation of all functions, including file I/O. The file I/O will be directed through the debugger and will be performed on the host system (semihosting). For example, printf/scanf will use the debugger console window and fread/fwrite will operate on files on the host system. **Note**: emulated I/O is relatively slow and can only be used when debugging.
- **Semihost(-mb)-nf (no files)**
  - Redlib only. Similar to Semihost, but only provides support for the 3 standard built-in streams – stdin, stdout, stderr. This reduces the memory overhead required for the data structures used by streams, but means that the user application cannot open and use files, though generally this is not a problem for embedded applications.
- **Nohost and Nohost-nf**
  - This library variant provides the string and memory handling functions and some file-based I/O functions. However, it assumes that you have no debugging host system, thus any file I/O will do nothing. However, it is possible for the user to provide their own implementations of some of these I/O functions, for example to redirect output to the UART.
- **None**
  - This has literally no stub and has the smallest memory footprint. It excludes low-level functions for all file-based I/O and some string and memory handling functions.

Note: -mb library variants are not selected by default during any wizard project creation however they may optionally be selected for enhanced semihost performance with the penalty of slightly larger RAM usage. Please see Semihosted Printf [192] for additional information.

In many embedded microcontroller applications it is possible to use the None variant by careful use of the C library, for instance avoiding calls to printf().

If you are using the wrong library variant, then you will see build errors of the form:

- Linker error "Undefined reference to ‘xxx’ "

For example for a project linking against Redlib(None) but using printf() :

```
… libcr_c.a(fpprintf.o): In function `printf':
fpprintf.c:(.text.printf+0x38): undefined reference to `__sys_write'
fpprintf.c:(.text.printf+0x4c): undefined reference to `__Ciob'
… libcr_c.a(_deferredlazyseek.o): In function `__flsbuf':
_deferredlazyseek.c:(.text.__flsbuf+0x88): undefined reference to `__sys_istty'
… libcr_c.a(_writebuf.o): In function `_Cwritebuf':
_writebuf.c:(.text._Cwritebuf+0x16): undefined reference to `__sys_flen'
_writebuf.c:(.text._Cwritebuf+0x26): undefined reference to `__sys_seek'
_writebuf.c:(.text._Cwritebuf+0x3c): undefined reference to `__sys_write'
… libcr_c.a(alloc.o): In function `_Csys_alloc':
alloc.c:(.text._Csys_alloc+0xe): undefined reference to `__sys_write'
alloc.c:(.text._Csys_alloc+0x12): undefined reference to `__sys_appexit'
… libcr_c.a(fseek.o): In function `fseek':
fseek.c:(.text.fseek+0x16): undefined reference to `__sys_istty'
fseek.c:(.text.fseek+0x3a): undefined reference to `__sys_flen'
```

Or if linking against NewlibNano(None):

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
… libc_nano.a(lib_a-writer.o): In function `_write_r':
writer.c:(.text._write_r+0x10): undefined reference to `_write'
… libc_nano.a(lib_a-closer.o): In function `_close_r':
closer.c:(.text._close_r+0xc): undefined reference to `_close'
… libc_nano.a(lib_a-lseekr.o): In function `_lseek_r':
lseekr.c:(.text._lseek_r+0x10): undefined reference to `_lseek'
… libc_nano.a(lib_a-readr.o): In function `_read_r':
readr.c:(.text._read_r+0x10): undefined reference to `_read'
… libc_nano.a(lib_a-fstatr.o): In function `_fstat_r':
fstatr.c:(.text._fstat_r+0xe): undefined reference to `_fstat'
… libc_nano.a(lib_a-isattyr.o): In function `_isatty_r':
isattyr.c:(.text._isatty_r+0xc): undefined reference to `_isatty'
```

In such cases, simply change the library hosting being used (as described below), or remove the call to the triggering C library function.

18.3 Switching the selected C library

Normally the library variant used by a project is set up when the project is first created by the New Project Wizard. However it is quite simple to switch the selected C library between Redlib, Newlib and NewlibNano, as well as switching the library variant in use.

To switch, highlight the project in the Project Explorer view and go to:

Quickstart -> Quick Settings -> Set library/header type

and select the required library and variant.

18.3.1 Manually Switching

Alternatively, you can make the required changes to your project properties manually as follows...

When switching between Newlib(Nano) and Redlib libraries you must also switch the headers (since the 2 libraries use different header files). To do this:

1. Select the project in Project Explorer
2. Right-click and select Properties
3. Expand C/C++ Build and select Settings
4. In the Tools settings tab, select Miscellaneous under MCU C Compiler. **Note**: Redlib is not available for C++ projects
5. In Library headers, select Newlib or Redlib
6. In the Tools setting tab, select Architecture & Headers under MCU Assembler
7. In Library headers, select Newlib or Redlib

Repeat the above sequence for all Build Configurations (typically Debug and Release).

To then change the libraries actually being linked with (assuming you are using Managed linker scripts):

1. Select the project in Project Explorer
2. Right-click and select Properties
3. Expand C/C++ Build and select Settings
4. In the Tools settings tab, select Managed Linker Script under MCU Linker
5. In the Library drop-down, select the Newlib, NewlibNano or Redlib library variant that you require (None, Nohost, Semihost, Semihost-nf).

Again repeat the above sequence for all Build Configurations (typically Debug and Release). **Note**: Redlib is not available for C++ projects.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

18.4 What is Semihosting?

Semihosting is a term to describe application IO via the debug probe. For this to operate, library code and debug support are required.

18.4.1 Background to Semihosting

When creating a new embedded application, it can sometimes be useful during the early stages of development to be able to output debug status messages to indicate what is happening as your application executes.

Traditionally, this might be done by piping the messages over a serial cable connected to a Terminal program running on your PC. [276] MCUXpresso IDE offers an alternative to this scheme, called semihosting. Semihosting provides a mechanism for code running on the target board to use the facilities of the PC running the IDE. The most common example of this is for the strings passed to a printf being displayed in the IDE's console view.

The term "semihosting" was originally termed by ARM in the early 1990s, and basically indicates that part of the functionality is carried out by the host (the PC with the debug tools running on it), and partly by the target (your board). The original intention was to provide I/O in a target environment where no real peripheral-based I/O was available at all.

18.4.2 Semihosting Implementation

The way it is actually implemented by the tools depends upon which target CPU you are running on. With Cortex-M based MCUs, the bottom level of the C library contains a special BKPT instruction. The execution of this is trapped by the debug tools which determine what operation is being requested – in the case of a printf, for example, this will effectively be a "write character to stdout". The debug tools will then read the character from the memory of the target board – and display it in the console window within the IDE.

Semihosting also provides support for a number of other I/O operations (though this relies upon your debug probe also supporting them)... For example it provides the ability for scanf to read its input from the IDE console. It also allows file operations, such that fopen can open a file on your PC's hard drive, and fscanf can then be used to read from that file.

18.4.3 Semihosting Performance

It is fair to say that the semihosting mechanism does not provide a high performance I/O system. Each time a semihosting operation takes place, the processor is basically stopped whilst the data transfer takes place. The time this takes depends somewhat on the target CPU, the debug probe being used, the PC hardware and the PC operating system. But it takes a definite period of time, which may make your code appear to run more slowly.

In MCUXpresso IDE version 10.2.0 semihosting performance has been enhanced to deliver roughly double the speed when compared with the previous IDE release. Furthermore, a new **MB** library variant is been supplied that delivers a significant further improvement in performance when combined with LinkServer debug connections. This library along new LinkServer debug support provides the added benefit of no impact on code execution performance.

18.4.4 Important notes about using Semihosting

When you have linked with the semihosting library, your application will no longer work standalone – it will only work when connected to the debugger.

Semihosting operations cause the CPU to drop into "debug state", which means that for the duration of the data transfer between the target and the host PC no code (including interrupts) will

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

get executed on the target. Thus if your application uses interrupts, then it is normally advisable to avoid the use of semihosting whilst interrupts are active – and certainly within interrupt handlers themselves. If you still need to use printf, then you can retarget the bottom level of the C library to use an alternative communication channel, such as a UART or the Cortex-M CPU's ITM channel.

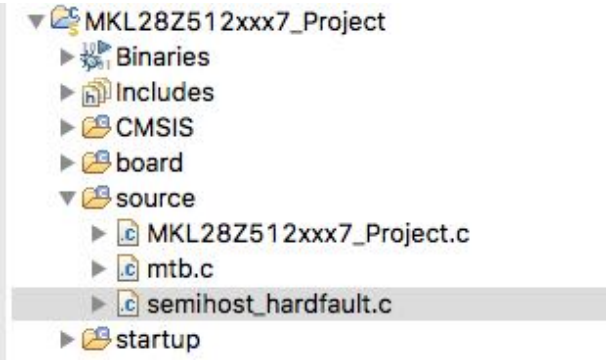
18.4.5 Semihosted printf and Debugging

Semihosting is common to all supported debug solutions so the implications of this mechanism should be understood:

Projects linked against semihosting libraries that perform semihosted operations e.g. printf, can not execute without a debugger connected. This is because semihosted operations make use of a BreakPoint instruction that is intercepted by the debug tools to trigger the desired behaviour (typically the printf string appearing within the IDE console). Without a debug connection, these BreakPoint instructions will not be trapped and a Hard Fault exception will occur. By default, the supplied Hard Fault handler implementation will be an infinite loop. Therefore if an 'attach' is performed to such a target, the user will observe the code running within the hard fault handler. To avoid this occurring, ensure that the project makes no use of semihosted operations via sending output to a UART, using the ITM feature, commenting out semihosted operations etc.

In consequence, if for example a user had created an LED blinky application that also performed semihosted printf operations, then without a debug connection the blinky would stop when the first printf was executed.

Introduced in MCUXpresso IDE version 10.1.0: New projects and newly imported SDK example projects will automatically include a semihost hardfault handler (as can be seen in the image below). The purpose of this handler is to prevent the problem described above. Now, if a semihosted operation is performed without debug tools attached, the new semihost hardfault handler will be entered. The handler will check to see if a semihosted operation cause it to be entered and if so, simply return.



In consequence, if the user creates an LED blinky application that also performs semihosted printf operations, then without a debug connection the blinky will continue regardless of any printf operation that may occur.

This functionality can be disabled if required by either simply deleting the handler file, or by defining a symbol:

```
__SEMIHOST_HARDFAULT_DISABLE
```

Note: Previously created projects imported into MCUXpresso IDE (such as LPCOpen projects), will not inherit this feature.

Introduced in MCUXpresso IDE version 10.2.0: The inclusion of the hardfault handler can be controlled via a preference *preferences -> MCUXpresso IDE -> SDK Options -> Include semihost hardfault handler ...*, where the default is to include.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Redlib Semihost MB

Introduced in MCUXpresso IDE version 10.2.0: is the optional Redlib Semihost **MB** library variant. This library provides enhanced semihosting performance from LinkServer debug connections (other debug solutions will perform as before) with the added benefit of no impact on code execution performance. There is a small penalty of slightly larger code and data sizes compared to other Redlib Semihost libraries. This optional library is recommended for users needing high semihosting performance and/or have slow debug probe performance.

18.4.6 Semihosting Specification

The semihosting mechanism used within MCUXpresso IDE is based on the specification contained in the following document available from ARM's website... => ARM Developer Suite (ADS) v1.2 Debug Target Guide, Chapter 5. Semihosting

18.5 Use of printf

By default, the output from printf() (and puts()) will be displayed in the debugger console via the semihosting mechanism. This provides a very easy way of getting basic status information out from your application running on your target.

For printf() to work like this, you must ensure that you are linking with a "semihost" or "semihost-nf" library variant.

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

18.5.1 Redlib printf Variants

Redlib provides the following two variants of printf. Many of the MCUXpresso New project wizards provide options to select which of these to use when you create a new project.

Character vs String output

By default printf() and puts() functions will output the generated string at once, so that a single semihosted operation can output the string to the console of the debugger. **Note**: these versions of printf() /puts() make use of malloc() to provide a temporary buffer on the heap in order to generate the string to be displayed.

It is possible to switch to using "character-by-character" versions of these functions (which do not require heap space) by specifying the build define "CR_PRINTF_CHAR" (which should be set at the project level). This can be useful, for example, if you are retargeting printf() to write out over a UART (as detailed below)- as in this case it is pointless creating a temporary buffer to store the whole string, only to then print it out over the UART one character at a time

Integer only vs full printf (including floating point)

The printf() routine incorporated into Redlib is much smaller than that in Newlib. Thus if code size is an issue, then always try to use Redlib if possible. In addition, if your application does not pass floating point numbers to printf, you can also select an "integer only" (non-floating point compatible) variant of printf. This will reduce code size further.

To enable the "integer only" printf from Redlib, define the symbol "CR_INTEGER_PRINTF" (at the project level). This is done by default for projects created from the SDK new project wizard.

18.5.2 NewlibNano printf Variants

By default, NewlibNano uses non-floating point variants of the printf and scanf family of functions, which can help to dramatically reduce the size of your image if only integer values are used by such functions.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

If your codebase does require floating point variants of printf/scanf, then these can be enabled by going to:

Project -> Properties -> C/C++ Build -> Settings -> MCU Linker -> Managed Linker Script and selecting the " *Enable printf/scanf float*" tick box.

18.5.3 Newlib printf variants

Newlib provides an “iprintf” function which implements integer only printf.

18.5.4 Printf when using LPCOpen

If you are building your application against LPCOpen, you may find that printf output does not get displayed in MCUXpresso IDE’s debug console by default. This is due to many LPCOpen board library projects by default redirecting printf to a UART output.

If you want to direct printf output to the debug console instead, then you will need to modify your projects so that:

1. Your main application project is linked against the “semihost” variant of the C library, and
2. You disable the LPCOpen board library’s redirection of printf output by either:
   - locating the source file board.c within the LPCOpen board library and comment out the line: #include retarget.h, or
   - locating the file board.h and enable the line: #define DEBUG_SEMIHOSTING

18.5.5 Printf when using SDK

The MCUXpresso SDK codebase provides its own printf style functionality through the macro PRINTF. This is set up in the header file fsl_debug_console.h such that it can either point to the printf function provided by the C library itself, or can be directly to the SDK function pseudo-printf function : DbgConsole_Printf() . This will typically cause the output to be sent out via a UART (which may be connected to an on-board debug probe which will sent it back to the host over a USB VCOM channel). This is controlled by the macro **SDK_DEBUGCONSOLE** thus:

- If SDK_DEBUGCONSOLE == 0
  - PRINTF is directed to C library printf()
- If SDK_DEBUGCONSOLE == 1
  - PRINTF is directed to SDK DbgConsole_Printf()

The Advanced page of the SDK new project wizard and Import SDK examples wizard offer the option to configure a project so that PRINTF is directed to C library printf() by setting **SDK_DEBUGCONSOLE** appropriately.

In addition, if PRINTF is being directed to the C library printf(), then if **SDK_DEBUGCONSOLE_UART** is also defined, then printf output will still be directed to the UART. Again the Advanced page of the SDK new project wizard and Import SDK examples wizard offer an option to control this.

18.5.6 Retargeting printf/scanf

By default, the printf function outputs text to the debug console using the “semihosting” mechanism.

In some circumstances, this output mechanism may not be suitable for your application. Instead, you may want printf to output via an alternative communication channel such as a UART or – on Cortex-M3/M4 – the ITM channel of SWO Trace. In such cases you can retarget the appropriate portion of the bottom level of the library.

The section “How to use ITM Printf” below provides an example of how this can be done.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: when retargeting these functions, you can typically link against the “nohost” variant of the C Library, rather than the “semihost” one.

Redlib

To retarget Redlib’s printf(), you need to provide your own implementations of the function __sys_write():

```
int __sys_write(int iFileHandle, char *pcBuffer, int iLength)
```

Function returns number of unwritten bytes if error, otherwise 0 for success

Similarly if you want to retarget scanf(), you need to provide your own implementations of the function __sys_readc():

```
int __sys_readc(void)
```

Function returns character read

Note: these two functions effectively map directly onto the underlying “semihosting” operations.

Newlib / NewlibNano

To retarget printf(), you will need to provide your own implementation of the Newlib system function _write():

```
int _write(int iFileHandle, char *pcBuffer, int iLength)
```

Function returns number of unwritten bytes if error, otherwise 0 for success

To retarget scanf, you will need to provide your own implementation of the Newlib system function _read():

```
int _read(int iFileHandle, char *pcBuffer, int iLength)
```

Function returns number of characters read, stored in pcBuffer

More information on the Newlib system calls can be found at: https://sourceware.org/newlib/libc.html#Syscalls

18.5.7 How to use ITM Printf

ITM Printf is a scheme to achieve application IO via a debug probe without the usual semihosting penalties.

ITM Overview

As part of the Cortex-M3/M4 SWO Trace functionality available when using an LPC-Link2 (with NXP’s CMSIS-DAP firmware), MCUXpresso IDE provides the ability to make use of the ITM : The Instrumentation Trace Macrocell (ITM) block provides a mechanism for sending data from your target to the debugger via the SWO trade stream. This communication is achieved through a memory-mapped register interface. Data written to any of 32 stimulus registers is forwarded to the SWO stream. Unlike other SWO functionality, using the ITM stimulus ports requires changes to your code and so should not be considered non-intrusive.

Printf operations can be carried out directly by writing to the ITM stimulus port. However the stimulus port is output only. And therefore scanf functionality is achieved via a special global

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

variable, which allows the debugger to send characters from the console to the target (using the trace interface). The debugger writes data to the global variable named ITM_RxBuffer to be picked up by scanf.

Note: MCUXpresso IDE currently only supports ITM via stimulus port 0.

Note: For more information on SWO Trace, please see the MCUXpresso IDE LinkServer SWO Trace Guide.

ITM printf with SDK

The Advanced page of the SDK new project wizard and Import SDK examples wizard offer the option to configure a project so as to redirect printf/scanf to ITM. Selecting this option will cause the file retarget_itm.c to be generated in your project to carry out the redirection.

ITM printf with LPCOpen

To use this functionality with an LPCOpen project you need to: Include the file retarget_itm.c in your project – available from the Examples subdirectory of your IDE installation Ensure you are using a semihost, semihost-nf, or nohost C library variant. Then simply add calls to printf and scanf to your code.

If you just linking against the LPCOpen Chip library, then this is all you need to do. However if you are also linking against an LPCOpen board library then you will likely see build errors of the form:

```
../src/retarget.h:224: multiple definition of `__sys_write'
../src/retarget.h:240: multiple definition of `__sys_readc'
```

locating the file board.h and enable the line: *#define DEBUG_SEMIHOSTING*, or locating the source file board.c within the LPCOpen board library and comment out the line: *#include "retarget.h"*

18.6 itoa() and uitoa()

itoa() is non-standard library function which is provided in many other toolchain to convert an integer to a string.

18.6.1 Redlib

To ease porting, MCUXpresso IDE provides two variants of this function in the Redlib C library....

```
char * itoa(int value, char *vstring, unsigned int base);
char * uitoa(unsigned int value, char *vstring, unsigned int base);
```

which can be accessed via the system header....

```
#include <stdlib.h>
```

itoa() converts an integer value to a null-terminated string using the specified base and stores the result in the array pointed to by the vstring parameter. Base can take any value between 2 and 16; where 2 = binary, 8 = octal, 10 = decimal and 16 = hexadecimal.

If base is 10 and the value is negative, then the resulting string is preceded with a minus sign (-). With any other base, value is always considered unsigned. The return value to the function is a pointer to the resulting null-terminated string, the same as parameter vstring.

uitoa() is similar but treats the input value as unsigned in all cases.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: the caller is responsible for reserving space for the output character array – the recommended length is 33, which is long enough to contain any possible value regardless of the base used.

Example invocations

```
char vstring [33];
itoa (value,vstring,10); // convert to decimal
itoa (value,vstring,16); // convert to hexadecimal
itoa (value,vstring,8);; // convert to octal
```

Standards compliance

As noted above, itoa() / uitoa() are not standard C library functions. A standard-compliant alternative for some cases may be to use sprintf() - though this is likely to cause an increase in the size of your application image:

```
sprintf(vstring,"%d",value); // convert to decimal
sprintf(vstring,"%x",value); // convert to hexadecimal
sprintf(vstring,"%o",value); // convert to octal
```

18.6.2 Newlib/NewlibNano

Newlib and NewlibNano now also provide similar functionality though with slightly different naming - **itoa()** and **utoa()**.

18.7 Libraries and linker scripts

When using the managed linker script mechanism, as described in the chapter “Memory configuration and Linker Script Generation”, then the appropriate settings to link against the required library family and variant will be handled automatically.

However if you are not using the managed linker script mechanism, then you will need to define which library files to use in your linker script. To do this, add one of the following entries before the SECTION line in your linker script:

- Redlib (None), add
  - [C project only]: GROUP (libcr_c.a libcr_eabihelpers.a)
- Redlib (Nohost), add
  - [C projects only]: GROUP (libcr_nohost.a libcr_c.a libcr_eabihelpers.a)
- Redlib (Semihost-nf), add
  - [C projects only]: GROUP (libcr_semihost_nf.a libcr_c.a libcr_eabihelpers.a)
- Redlib (Semihost), add
  - [C projects only]: GROUP (libcr_semihost.a libcr_c.a libcr_eabihelpers.a)
- NewlibNano (None), add
  - [C projects]: GROUP (libgcc.a libc_nano.a libm.a libcr_newlib_none.a)
  - [C++ projects]: GROUP (libgcc.a libc_nano.a libstdc++_nano.a libm.a libcr_newlib_none.a)
- NewlibNano (Nohost), add
  - [C projects]: GROUP (libgcc.a libc_nano.a libm.a libcr_newlib_nohost.a)
  - [C++ projects]: GROUP (libgcc.a libc_nano.a libstdc++_nano.a libm.a libcr_newlib_nohost.a)
- NewlibNano (Semihost), add
  - [C projects]: GROUP (libgcc.a libc_nano.a libm.a libcr_newlib_semihost.a)
  - [C++ projects]: GROUP (libgcc.a libc_nano.a libstdc++_nano.a libm.a libcr_newlib_semihost.a)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Newlib (None), add
  - [C projects]: GROUP (libgcc.a libc.a libm.a libcr_newlib_none.a)
  - [C++ projects]: GROUP (libgcc.a libc.a libstdc++.a libm.a libcr_newlib_none.a)
- Newlib (Nohost), add
  - [C projects]: GROUP (libgcc.a libc.a libm.a libcr_newlib_nohost.a)
  - [C++ projects]: GROUP (libgcc.a libc.a libstdc++.a libm.a libcr_newlib_nohost.a)
- Newlib (Semihost), add
  - [C projects]: GROUP (libgcc.a libc.a libm.a libcr_newlib_semihost.a)
  - [C++ projects]: GROUP (libgcc.a libc.a libstdc++.a libm.a libcr_newlib_semihost.a)

In addition, if using NewlibNano, then tick box method of enabling printf/scanf floating point support in the Linker pages of Project Properties will also not be available. In such cases, you can enabling floating point support manually by going to:

Project Properties -> C/C++ Build -> Settings -> MCU Linker -> Miscellaneous

and entering *-u _printf_float* and/or *-u _scanf_float* into the “Linker flags” box.

A further alternative is to put an explicit reference to the required support function into your project codebase itself. One way to do this is to add a statement such as:

asm (“.global _printf_float”);

to one (or more) of the C source files in your project.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

19. Memory Configuration and Linker Scripts

19.1 Introduction

A key part of the core technology within MCUXpresso IDE is the principle of a default defined memory map for each MCU. For devices with internal Flash, this will also specify a Flash driver to be used to program that Flash memory (for use with LinkServer “native” debug probes).

For preinstalled MCUs, the definition of the memory map is contained within the MCU part knowledge that is built into the product. For MCUs installed into MCUXpresso IDE from an SDK, the definition of the memory map is loaded from the manifest file within the SDK structure.

But in both cases, the defined memory map is used by MCUXpresso IDE to drive the “managed linker script” mechanism. This auto-generates a linker script to place the code and data from your project appropriately in memory, as well as being made available to the debugger.

A project’s memory map can be viewed and modified by the user to add, remove (split/join) or reorder blocks using the in place Memory Configuration Editor. For example, if a project targets an MCU that supports external Flash (e.g. SPIFI), then it’s memory map can be easily extended to define the SPIFI memory region (base and size). In addition, an appropriate Flash driver can be associated with the newly defined region.

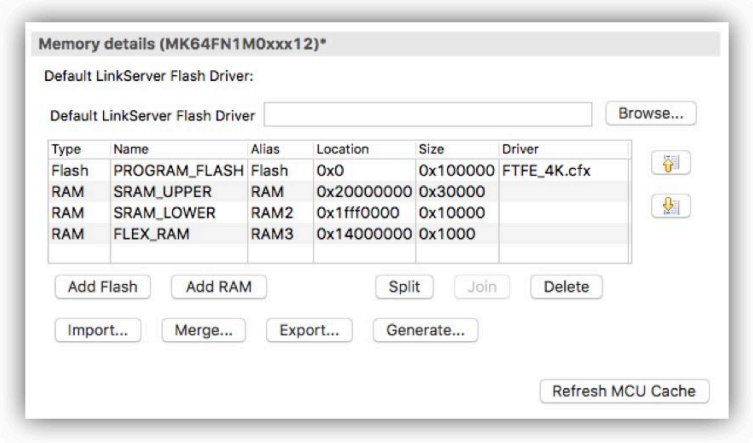


Figure 19.1. Memory Configuration

Introduced in MCUXpresso IDE version 10.3.0 Memory configurations can be edited directly in place rather than requiring a separate *Edit* to launch a separate dialogue. In place editing of memory configurations is incorporated within all project wizards and project properties views.

19.2 Managed Linker Script Overview

By default, the use of “managed linker scripts” is enabled for projects. This mechanism allows MCUXpresso IDE to automatically create a script for each build configuration that is suitable for the MCU selected for the project and the C libraries being used. It will create (and at times modify) three linker script files for each build configuration of your project:

```
<projname>_<buildconfig>_lib.ld
<projname>_<buildconfig>_mem.ld
<projname>_<buildconfig>.ld
```

This set of hierarchical files are used to define the C libraries being used, the memory map of the system and the way your code and data is placed into the memory map. These files will be located in the build configuration subdirectories of your project (typically – Debug and Release).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

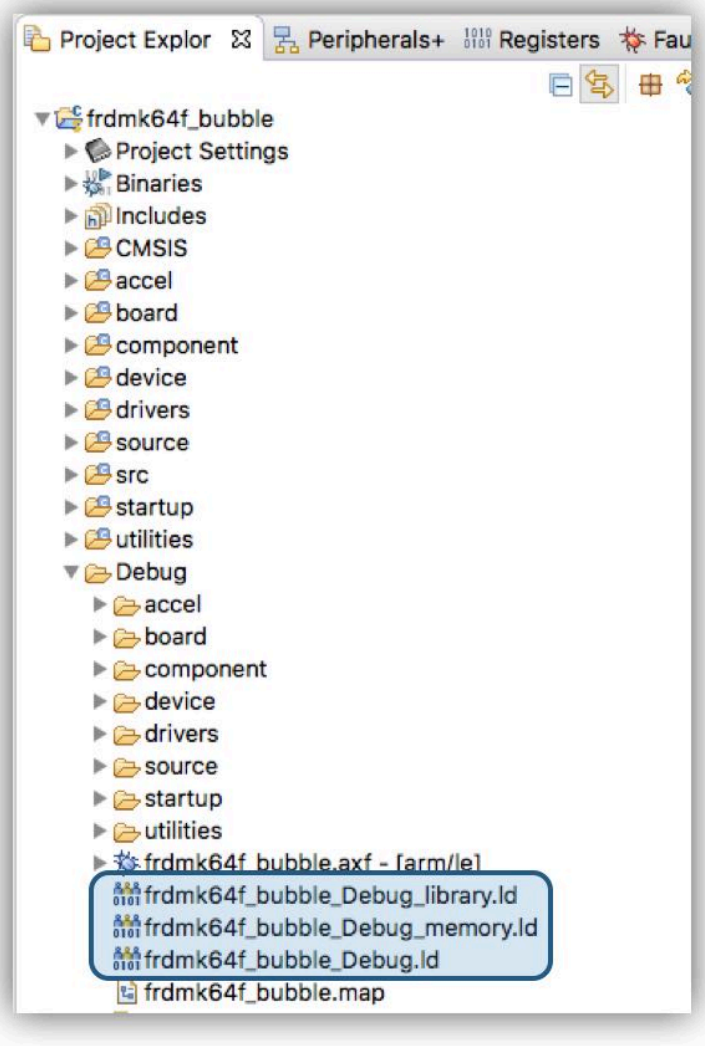


Figure 19.2. Project Explorer Debug folder Linker Scripts

The managed linker script mechanism also automatically takes into account memory map changes made in the Memory Configuration Editor as well as other configuration changes, such as C/C++ library settings.

See also the section on Heap and Stack View [150].

19.3 How are Managed Linker Scripts Generated?

MCUXpresso IDE passes a set of parameters into the linker script generator (based on the “FreeMarker” scripting engine) to create an appropriate linker script for your project. This generator uses a set of conditionally parsed template files, each of which control different aspects of the generated linker script.

It is possible to modify certain aspects of the generated linker script by providing one or more modified template files locally within *linkscripts* folder of project directory structure. Any such templates that you provide locally will then override the default ones built into MCUXpresso IDE. A full set of the default linker templates (.ldt) files are provided inside *Wizards**linker* subdirectory of your IDE install.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

19.4 Default Image Layout

Code and initial values of initialised data items are placed into first bank of Flash (as show in memory configuration editor). During startup, MCUXpresso IDE startup code copies the data into the first bank of RAM (as show in memory configuration editor), and zero initializes the BSS data directly after this in memory. This process uses a global section table generated into the image from the linker script.

Other RAM blocks can also have data items placed into them under user control and the startup code will also initialise these automatically. See later in this chapter for more details.

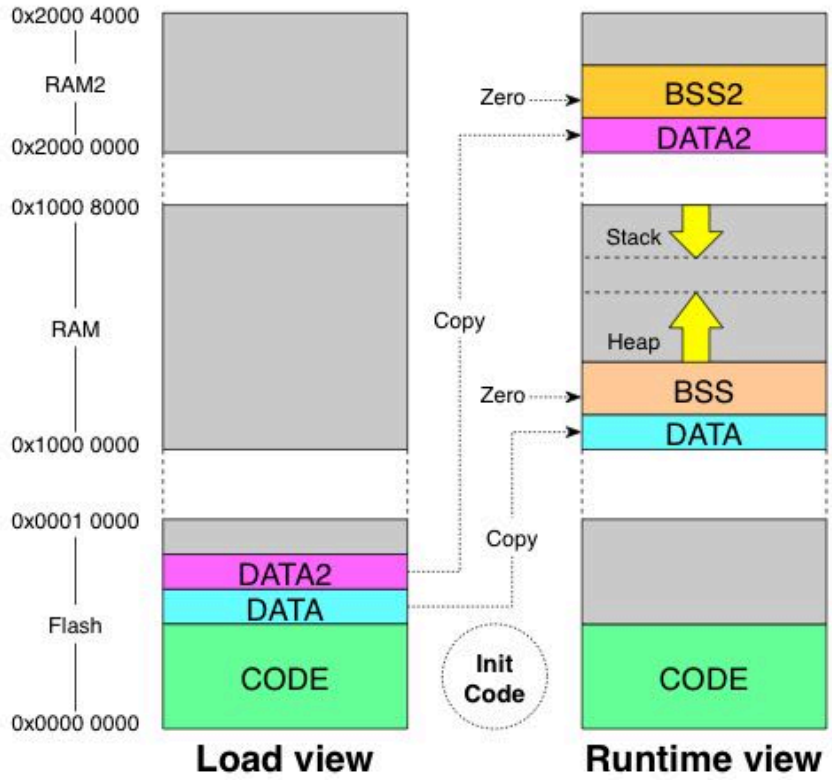


Figure 19.3. Default Memory Layout

Note: The above memory layout is simply the default used by the IDE's managed linker script mechanism. There are a number of mechanisms that can be used to modify the layout according to the requirements of your actual project – such as simply editing the order of the RAM banks in the Memory Configuration Editor. These various methods are described later in this chapter.

The default memory layout will also locate the heap and stack in the first RAM bank, such that:

- the heap is located directly after the BSS data, growing upwards through memory
- the stack located at the end of the first RAM bank, growing down towards the heap

Again this heap and stack placement is a default and it is very easy to modify the locations for a particular project, as will be described later in this chapter.

Note: When you import a project, you may find that the defaults have already been modified. Check the Project Properties to confirm the exact details.

19.5 Examining the layout of the generated image

Looking at the size of the AXF file generated by building your project on disk does not provide any information as to how much Flash/RAM space your application will occupy when downloaded

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

to your MCU. The AXF file contains a lot more information than just the binary code of your application, for example the debug data used to provide source level information when debugging, that is never downloaded to your MCU.

19.5.1 Linker --print-memory-usage

MCUXpresso IDE projects use the --print-memory-usage option on the link step of a build to display memory usage information in the build console of the following form:

```
Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       25960 B         1 MB      2.48%
      SRAM_UPPER:        8472 B       192 KB      4.31%
      SRAM_LOWER:          0 GB        64 KB      0.00%
        FLEX_RAM:          0 GB         4 KB      0.00%
Finished building target: frdmk64f_bubble.axf
```

The memory regions displayed here will match up to the memory banks displayed in the memory configuration editor when the managed linker script mechanism is being used.

By default, the application will build and link against the first Flash memory found within the MCU's memory configuration. For most MCUs there will only be one Flash device available. In this case our project requires 25960 bytes of Flash memory storage, 2.48% of the available Flash storage.

RAM will be used for global variable, the heap and the stack. MCUXpresso IDE provides a flexible scheme to reserve memory for Stack and Heap. This build has reserved 4KB each for the stack and the heap contributing 8KB to the overall 8472 bytes reported.

If using the *'LPCXpresso style'* of heap and stack placement (described later in this chapter), the RAM consumption provided by this feature is only that of your global data. It will not include any memory consumed by your stack and heap when your application is actually executing.

Note: project imported into MCUXpresso IDE may not have been created with this option. To add this, right click on the project and select *C/C++ Build ->Settings -> MCU Linker -> Miscellaneous then click '+' and add --print-memory-usage*

Comparing Code Size

This summary provides a quick method to *see* usage of the memory regions and also changes in efficiency. Below are examples of Memory Usage for the same project compiled on an older version of MCUXpresso IDE vs the current version.

Code size with MCUXpresso IDE version 11.0.x:

```
Memory region         Used Size  Region Size  %age Used
     BOARD_FLASH:       40244 B        64 MB      0.06%
        SRAM_DTC:        8580 B       128 KB      6.55%
        SRAM_ITC:          0 GB       128 KB      0.00%
         SRAM_OC:          0 GB       256 KB      0.00%
     BOARD_SDRAM:          0 GB        32 MB      0.00%
Finished building target: evkbimxrt1050_bubble_peripheral.axf
```

Code size with MCUXpresso IDE version 11.1.x:

```
Memory region         Used Size  Region Size  %age Used
     BOARD_FLASH:       36192 B        64 MB      0.05%
        SRAM_DTC:        8580 B       128 KB      6.55%
        SRAM_ITC:          0 GB       128 KB      0.00%
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
         SRAM_OC:          0 GB       256 KB      0.00%
     BOARD_SDRAM:          0 GB        32 MB      0.00%
Finished building target: evkbimxrt1050_bubble_peripheral.axf
```

See the section on the Image Information [203] view for details on further image exploration.

19.5.2 arm-none-eabi-size

In addition, a post-build step will normally invoke the arm-none-eabi-size utility to provide this information in a slightly different form....

```
  text    data     bss     dec     hex filename
  2624     524      32    3180     c6c  LPCXpresso1768_systick_twinkle.axf
```

- **text** - shows the code and read-only data in your application (in decimal)
- **data** - shows the read-write data in your application (in decimal)
- **bss** - show the zero initialized (‘bss’ and ‘common’) data in your application (in decimal)
- **dec** - total of ‘text’ + ‘data’ + ‘bss’ (in decimal)
- **hex** - hexadecimal equivalent of 'dec'

Typically:

- the Flash consumption of your application will then be text + data
- the RAM consumption of your application will then be data + bss

Again if using the *'LPCXpresso style'* of heap and stack placement (described later in this chapter), the RAM consumption will not include any memory allocated for your stack and heap when your application is actually executing.

You can also manually run the arm-none-eabi-size utility on both your final application image, or on individual object files within your build directory by right clicking on the file in Project Explorer and selecting the *Binary Utilities -> Size* option.

19.5.3 Linker Map Files

The linker option “-map” option, which is enabled by default by the project wizard when a new project is created, allows you to analyse in more detail the contents of your application image. When you do a build, this will cause a file called *projectname.map* to be created in the Debug (or Release) subdirectory, which can be loaded into the editor view. This contains a large amount of information, including:

- A list of archive members (library objects) included with details
- A list of discarded input sections (because they are unused and the linker option --gc-sections is enabled).
- The location, size and type of all code, data and bss items that have been placed in the image

19.6 Image Information (Info)

The *Image Info* View provides tools for detailed analysis of an image structure and memory footprint.

The Image Info view is stacked by default in the MCUXpresso IDE Develop perspective, along with Problems and/or Console views.

The tool bar icons for this view are shown and detailed below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

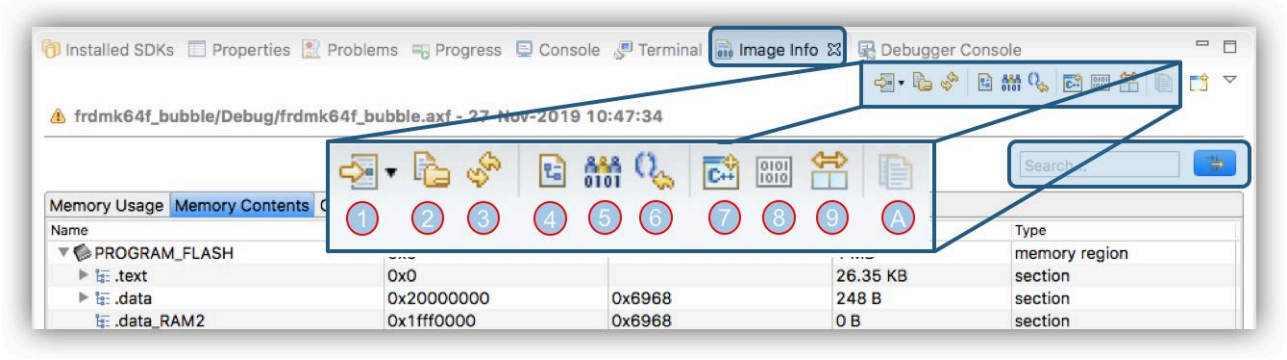


Figure 19.4. Image Info Toolbar

Where:

1. Loads the build artifact (.axf) associated with the currently selected project's active build configuration for analysis. This is the simplest option to follow to populate this view.
   - alternatively, an image, object or static library can be dragged onto this view
   - once loaded, the selected artifact name and build information (plus warnings if any) is displayed as a title to the view
   - if more than one project (or file) is selected and more than one Image Info view is open within the IDE, then the additional views will also be populated from the selection
   - also, if more than one build configuration is available, the drop down option allows *All build configurations* to be opened
   - this icon will be greyed out if the selected project's active build configuration has not been built
2. Browse to a build artifact containing symbolic information
3. Reload information from the currently loaded build artifact
   - this may be required when a project is rebuilt from changed sources
4. Open the Map file associated with the currently selected build artifact
   - this file will open up within the editor view where enhanced syntax highlighting [209] helps navigation
5. Open the Linker Script (.ld file) associated with the currently selected build artifact
   - this file will open up with the editor view where enhanced syntax highlighting [209] helps navigation and understanding
6. Locate the main symbol if present in the current tab
7. Enable/Disable C++ name mangling
   - this uses the c++filt binutils application to demangle C++ symbols from the view
     - all (mangled) items from the view will be affected – not only the current selection
8. Toggle between sizes in bytes and larger units (KB, MB etc.)
9. Click to compare with contents from another (new) Image Info view using the standard Eclipse compare utility
   - to use this feature, create a second Image Info view and load with another image, object etc. click compare in both views

10(A) Copy highlighted information to the clip board
   - copied information is held in .tsv format with the table headers added to the selection



Tip

These options are also available from a right click menu within the Image Info view

Also highlighted is the search/filter button, this can be used to switch between the highlighting of lines containing an entered search item and only displaying matching lines. This feature can be useful to remove clutter from large groups of items.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: information from highlighted lines is shown in the Properties view

The Image Information view (usually) consists of 3 sub views offering – Memory Usage, Memory Contents and (static) Call Graph information.

19.6.1 Memory Usage

The Memory Usage view shows how much memory (Flash and RAM) is used by the associated build artifact.

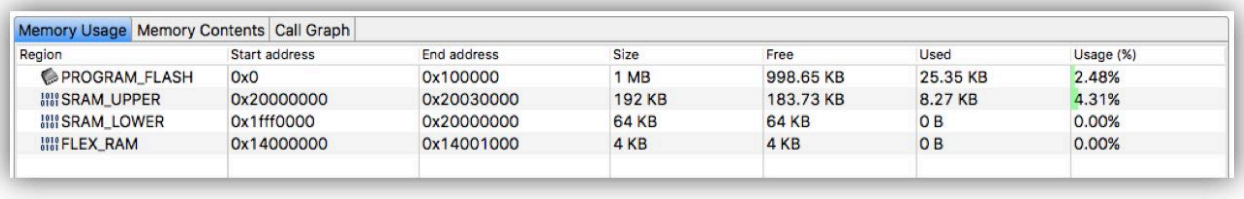


Figure 19.5. Image Info Mem Usage

The memory regions displayed will be the same as the selected project's build artifact (typically a project's generated elf (.axf) file. The detailed information is broadly the same as that provided by the Linker --print-memory-usage switch however, this view can be used to easily compare memory usage from one build to another following code changes, improvements, different build configurations etc.



Tip

As a guide the memory usage % display will be colored green when more than half of the available memory is free, then changing through yellow to red if more memory is used

Note: The Memory Usage tab will not be displayed in the following situations:

- A not-yet-linked file (*.o) was processed
- A static library (*.a) was processed
- A build artifact from outside the current workspace was processed – memory regions cannot be obtained in this case

Double click a Memory region to jump its *Contents*.

19.6.2 Memory Contents

The Memory Contents view provides a detailed view of the contents of each memory region. The image below shows various linker sections distributed within the memory regions.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

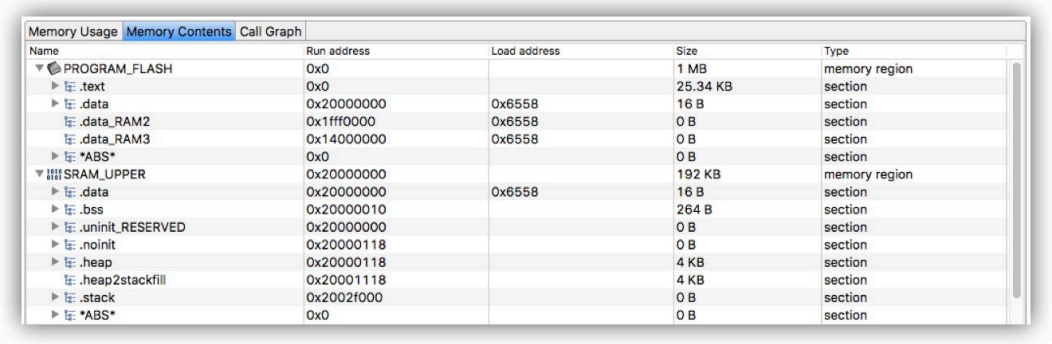


Figure 19.6. Image Info Mem Contents

Double clicking or pressing the *Enter* key on any selected symbol will open its definition.

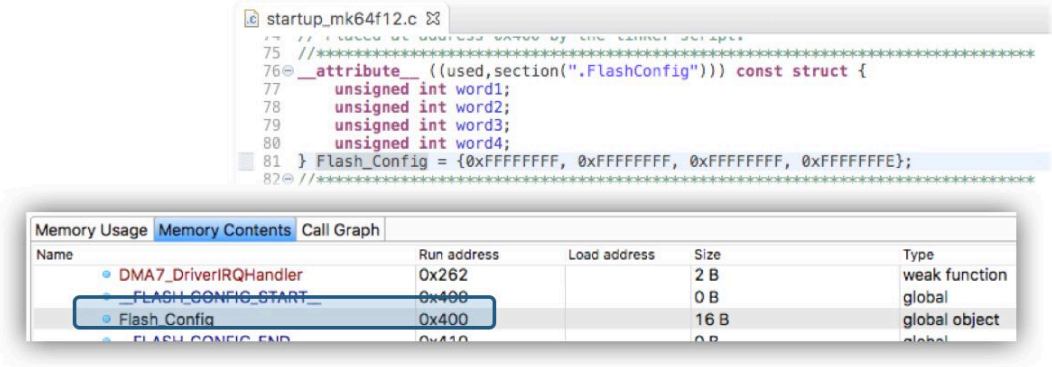


Figure 19.7. Image Info Mem Symbol linkage

Note: If a symbol cannot be found within the sources, for example the symbol is within a C library function, a message will be displayed in the Eclipse's status bar.

Selecting multiple lines within this view will total their memory usage.

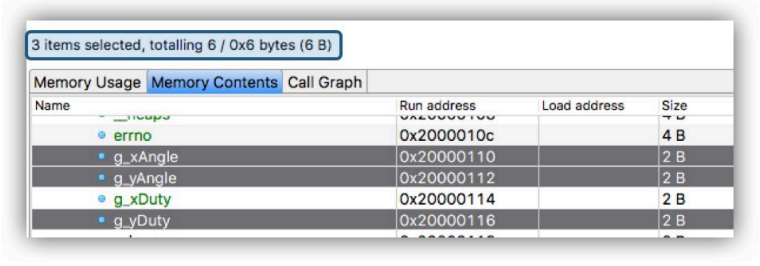


Figure 19.8. Image Info Mem Size

Note: When selecting multiple symbols, the sum of their individual sizes is computed without taking into account any space that was used for padding or alignment within the section. As a result, the actual section size might differ compared to the size of a multiple symbols selection. Note that the real section size within the application is the size displayed next to the section name, in the appropriate table column.

19.6.3 Call Graph

The Call Graph tab shows the static stack cost for the selected build artifact as generated via the -fstack-usage compiler option. The generation of Stack Usage information is now a default

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

option within MCUXpresso IDE version 11.0.0 but can be controlled via the Workspace project property shown below:

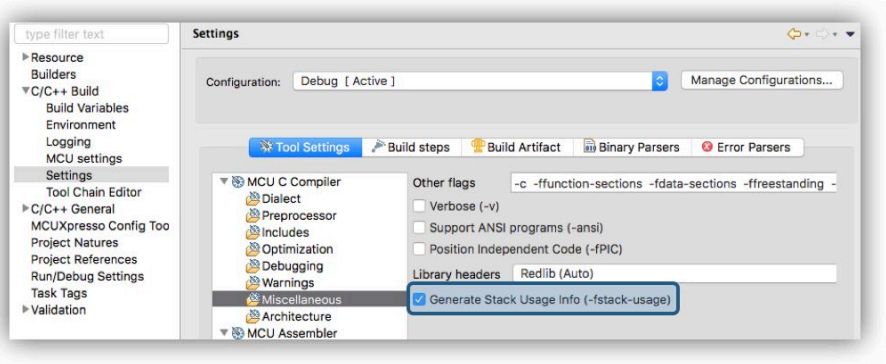


Figure 19.9. Image Info Call Graph Enable

This option enables the generation of .su (stack usage) files by the compiler and these are consumed (along with other information) to populate the Call Graph view. **Note**: the generation of these additional files has minimal impact on project build times.

If a project has been built and loaded, the call graph information for the selected build configuration will be available. Below is a truncated view of a call graph display, expanded and highlighted to display the main() function.

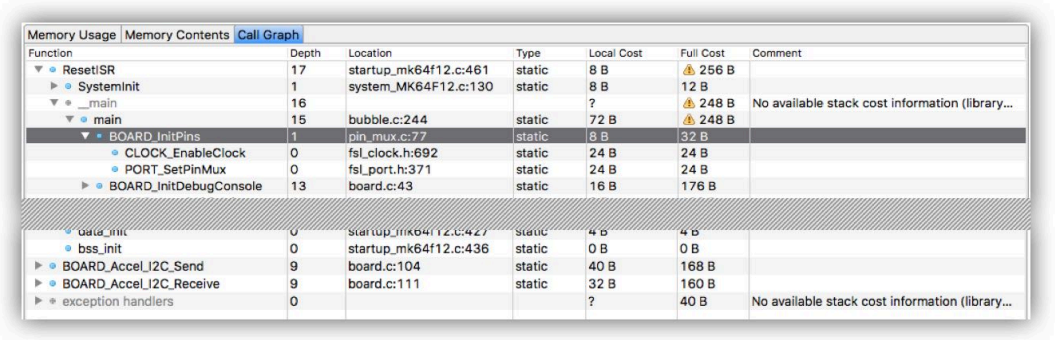


Figure 19.10. Image Info Call Graph

In this view, the columns have the following meaning:

- Function: displays the function name
- Depth: displays the maximum call depth
  - where N means the function has at least 1 child with a depth of N-1
  - and 0 means there are no child functions
- Location: function location within the source (file:line)
  - this will be empty if no source is found
- Type: show static or dynamic allocation type
- Local Cost: shows the number of bytes allocation by the function itself
- Full Cost: shows the number of bytes allocation by the function itself plus that of the deepest child function
- Comment: shows additional information such as recursive calls

Within the view, symbols are coloured to indicated meaning:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

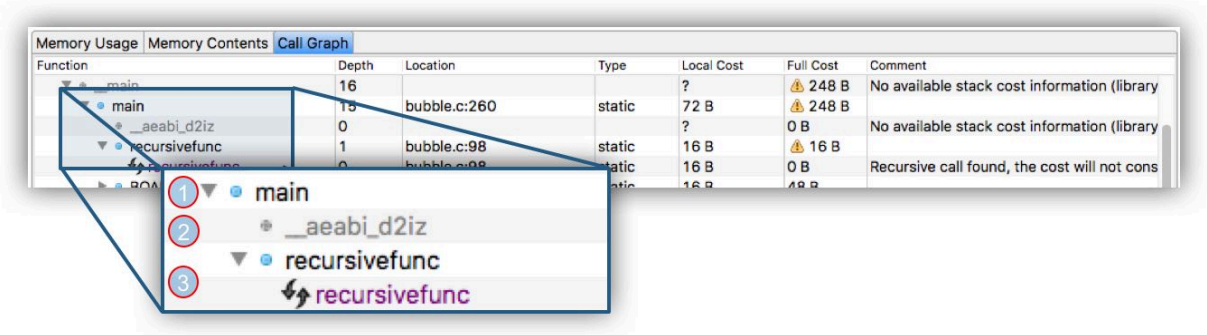


Figure 19.11. Image Info Call Graph Display Types

1. A symbol in black can be double clicked to open the associated source code
2. A symbol in grey has no associated source information
   - this might indicate an assembly or library symbol
3. A symbol with circular arrows indicates it has a recursive call and so its stack costs cannot be added to the full cost
4. exception handlers in grey (not show) will group any root symbol with a *Handler* suffix

Finally, if for any reason Call Graph information may be limited or stale, clear self describing warnings will be displayed.

19.6.4 Use of Filters

The search filter now supports both simple and regular expression search.

Below a filter for the symbol *main* will locate '__main' and 'main'.

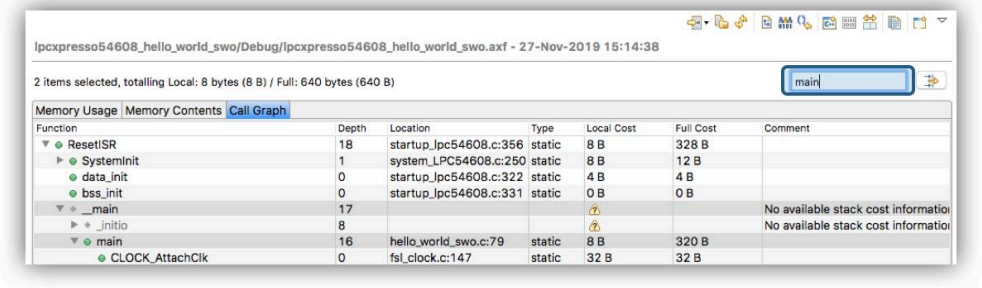


Figure 19.12. Call Graph Simple Filter

Regular expression filter supports standard regex searching...

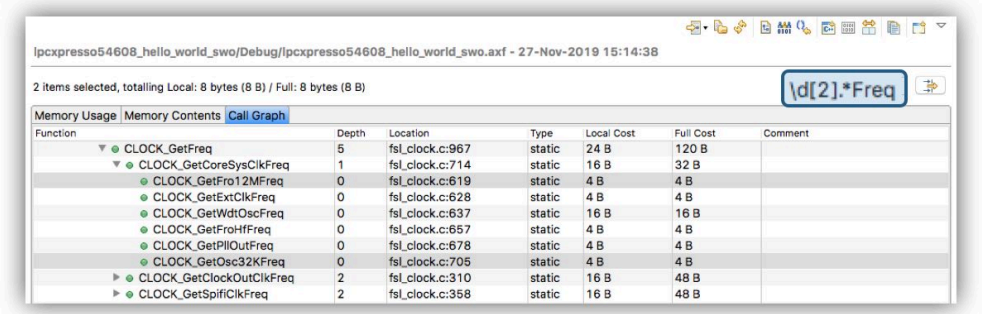


Figure 19.13. Call Graph RegEx Filter1

Use of NOT searching – search for CLOCK but not containing Xtal ...

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

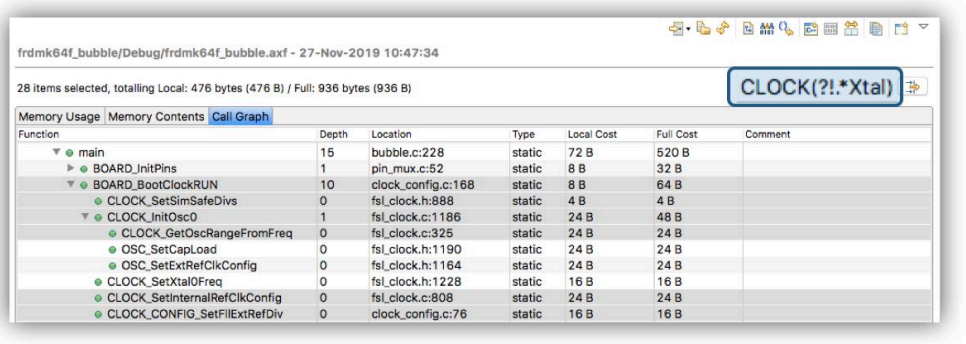


Figure 19.14. Call Graph RegEx Filter2

Note: If an error occurs when entering a regular expression, the message becomes red as you type and the tooltip will indicate the expression error.



Tip

Typically, to search for a string within a regular expression, you would write (.*)string(.*) ... In order to remove this requirement from users, strings are guarded by default at the beginning and end resulting in a search for *anything containing* the string. A side effect of this guard is that you can't search for something starting or ending with '.'.

19.7 Enhanced Syntax Highlighting

Introduced in MCUXpresso IDE version 11.0.0, additional editor capability delivering Enhanced Syntax Highlighting for GNU Linker Script .ld files (also Linker Script template and .map files). The primary goal of these enhancements is to simplify the exploration of these files and also ease the manual creation of Linker Script files for situations where MCUXpresso IDE's auto generated linker script mechanism cannot support the required configuration.

The new editor will be invoked automatically by double clicking on the .ld, .ldt or .map file within the project explorer view. If needed, this functionality can be disabled via *Preferences -> MCUXpresso IDE -> Editor Awareness*, the changes taking effect after restarting MCUXpresso IDE.

Note: To ensure that enabling and disabling the editors work as expected, MCUXpresso IDE should be launched in clean mode. This can be done either by calling the IDE executable from the command line with the *-clean* argument or by adding *-clean* on the first line of the *.ini* configuration file (which can be found in the same folder as the MCUXpresso IDE executable).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

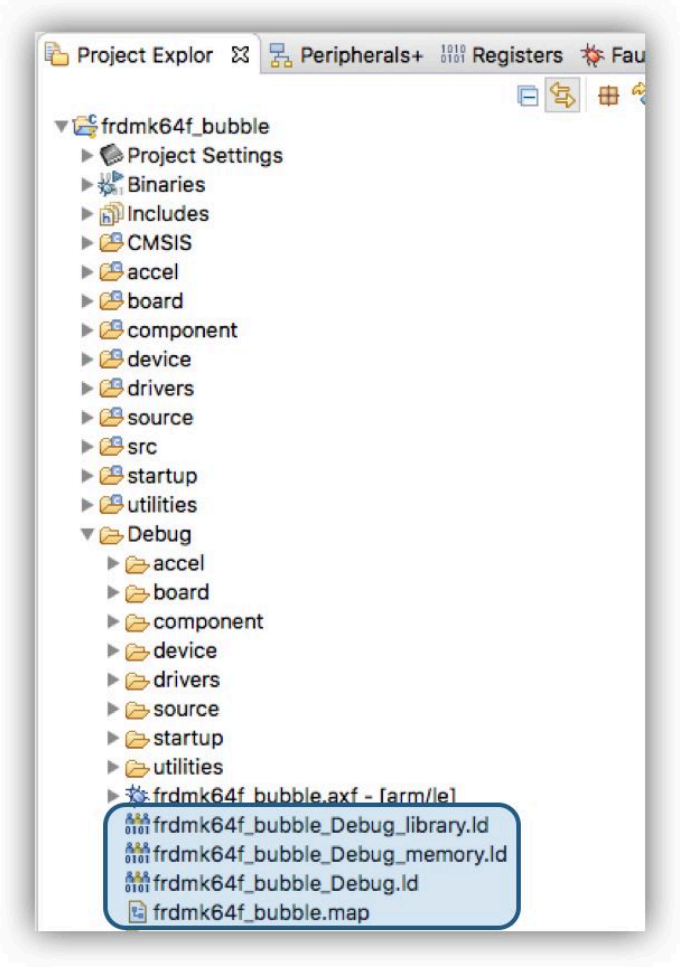


Figure 19.15. Project Build Configuration Files

Note: these files are automatically generated by the Managed Linker script mechanism [199] for the selected build configuration when a project is built

Once a file is opened as below, a number of features are available.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

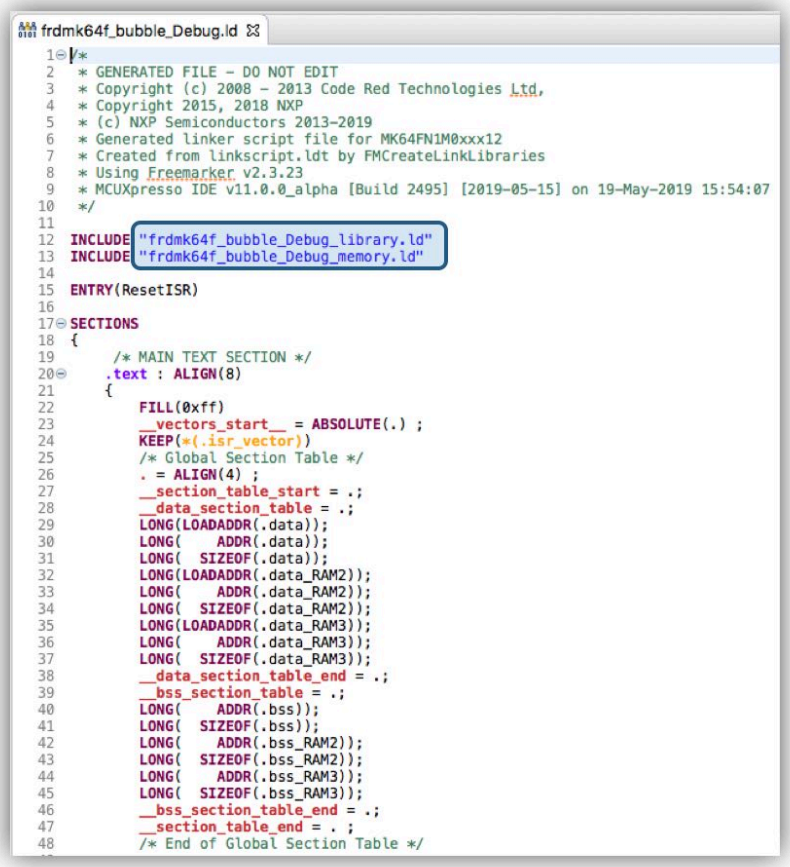


Figure 19.16. Linker Description file

Include files and Symbols source (as highlighted) can be opened in a new editor view via CTRL + Click (CMD + Click for Mac) on their filename.

The Editor also provides context aware code completion accessible by pressing CTRL + SPACE.

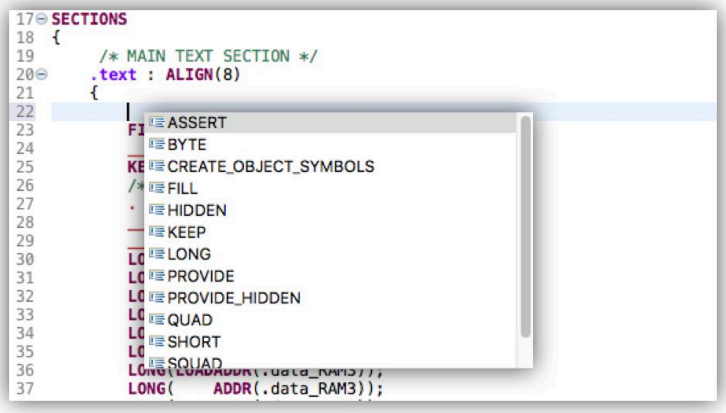


Figure 19.17. Auto Completion

The editor also provides error checking – validating that any changes are in accordance with the linker script syntax.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

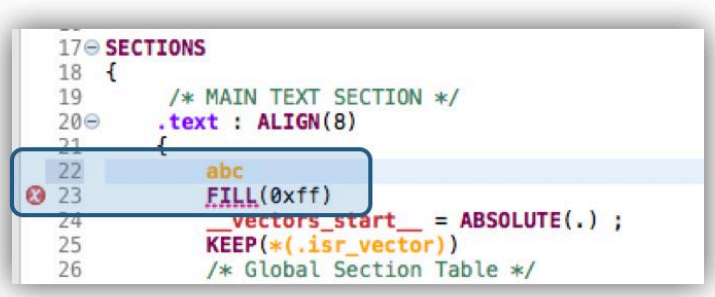


Figure 19.18. Error Checking Syntax

Furthermore, INCLUDE paths are verified and any error shown as below.

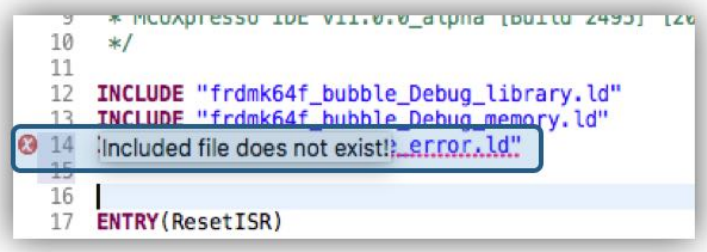


Figure 19.19. Error Checking Files

Error markers will be shown on the navigation bar and in the title of the editor window.

The Outline view displays an outline of the file that is currently open in the editor area.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

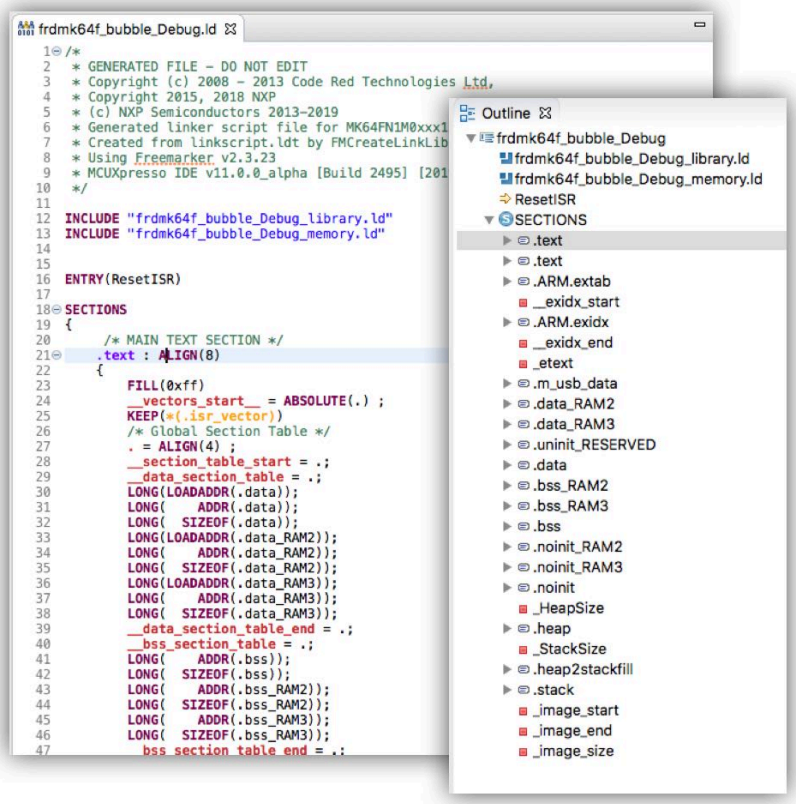


Figure 19.20. Linker Description Outline association ld

This is particularly useful for navigations through complex auto generated .map files

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

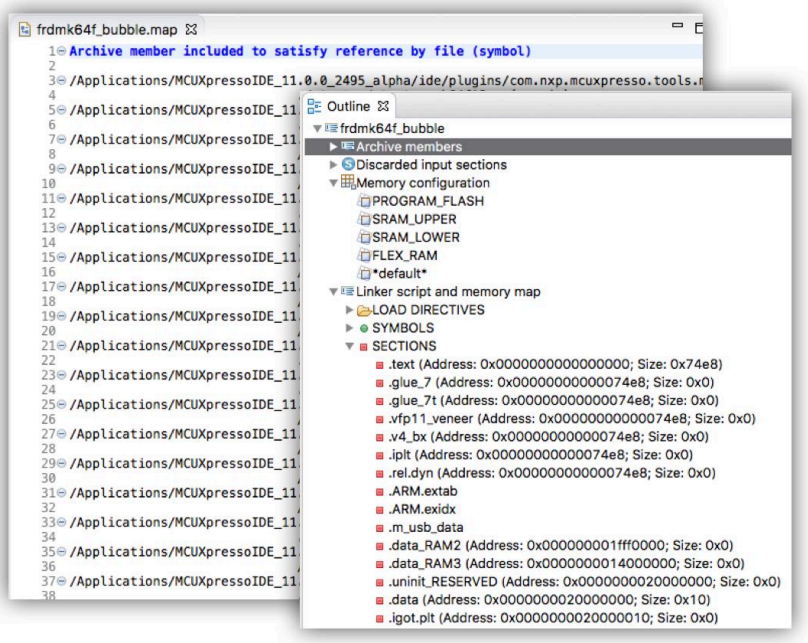


Figure 19.21. Map file Outline association map



Tip

Right clicking within the outline view will allow the opening of related source files.

Finally, if required, colours used for syntax highlighting can be configured via *Preferences -> MCUXpresso IDE -> Editor Awareness* as below.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

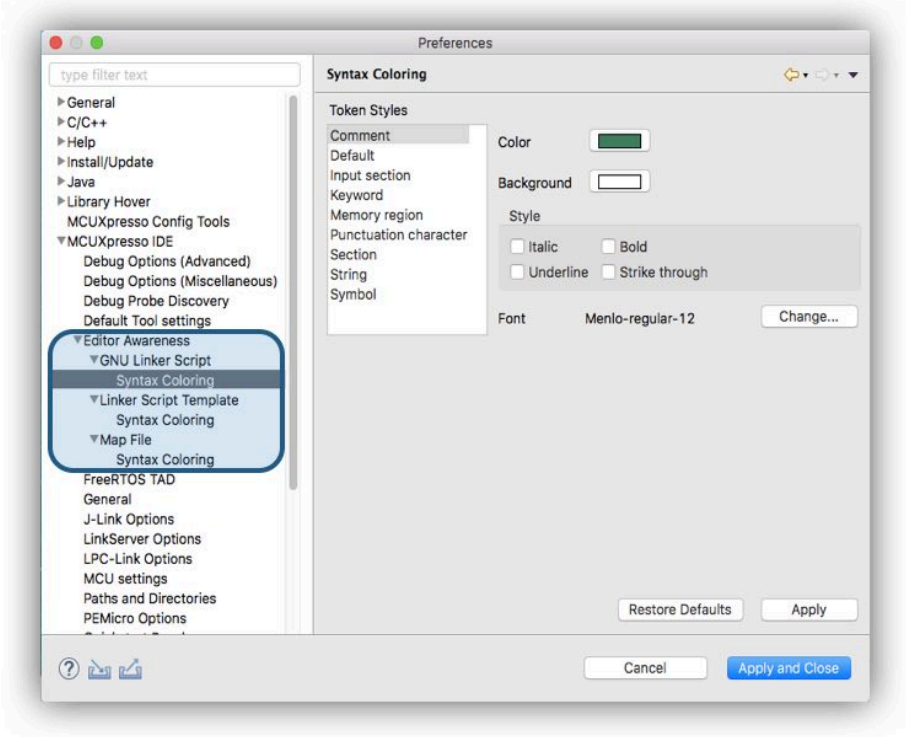


Figure 19.22. Editor Awareness Syntax Highlighting Preferences

19.8 Other Options affecting the Generated Image

19.8.1 LPC MCUs – Code Read Protection

Most of NXP's LPC Cortex-M based MCUs which have internal Flash memory contain "Code Read Protection" (CRP) support. This mechanism uses one of a number of known values being placed in a specific location in Flash memory to provide a number of levels of protection. When the MCU boots, this specific location in Flash memory is read and depending upon its value, the MCU may prevent access to the Flash memory by external devices. This location is typically at 0x2FC though for LPC18xx/43xx parts with internal Flash, the CRP location is at an offset of 0x2FC from the start of the Flash bank being used.

CRP : Preinstalled MCUs

Support for setting up the CRP memory location is provided via a combination of the Project Wizard, a header file and a number of macros. This support allows specific values to be easily placed into the CRP memory location, based on the user's requirements.

The New Project wizard contains an option to allow linker support for placing a CRP word to be enabled when you create a new project. This is typically enabled by default. This wizard option actually then controls the "Enable CRP" checkbox of the Project Properties linker Target tab.

In addition, the wizard will create a file, 'crp.c' which defines the 'CRP_WORD' variable which will contain the required CRP value. A set of possible values are provided by the NXP/crp.h header file that this then includes. Thus for example 'crp.c' will typically contain:

```
#include <NXP/crp.h>
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
```

which is then placed at the correct location in Flash by the linker script generated by the managed linker script mechanism:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
. = 0x000002FC ;
KEEP(*(.crp))
```

Note: the value CRP_NO_CRP ensures that the Flash memory is fully accessible. When you reach the stage of your project where you want to protect your image, you will need to modify the CRP word to contain an appropriate value.

Important Note: You should take particular care when modifying the value placed in the CRP word, as some CRP settings can disable some or all means of access to your MCU (including debug). Before making use of CRP, you are strongly advised to refer to the User Manual for the LPC MCU that you are using.

CRP : MCUs installed by Importing an SDK

The support for CRP in LPC parts imported into MCUXpresso IDE from an SDK, is generally similar to the Preinstalled MCUs. However rather than having a separate crp.c file, the CRP_WORD variable definition is generally found within the startup code.

19.8.2 Kinetis MCUs – Flash Config Blocks

Kinetis MCUs provides an alternative means of protecting the user's image in Flash using the Flash Configuration Block. The Flash Configuration Field is generally located at addresses 0x400-0x40F and unlike the LPC CRP mechanism only specific values give access, whereas any other values are likely to lock the part.

The value of the Flash Configuration block for a project is provided by the following structure which will be found in the startup code:

```
__attribute__ ((used,section(".FlashConfig"))) const struct {
    unsigned int word1;
    unsigned int word2;
    unsigned int word3;
    unsigned int word4;
} Flash_Config = {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE};
```

which is then placed appropriately by the linker script generated by the managed linker script mechanism.

```
/* Kinetis Flash Configuration data */
. = 0x400 ;
PROVIDE(__FLASH_CONFIG_START__ = .) ;
KEEP(*(.FlashConfig))
PROVIDE(__FLASH_CONFIG_END__ = .) ;
ASSERT(!(__FLASH_CONFIG_START__ == __FLASH_CONFIG_END__),
 "Linker Flash Config Support Enabled, but no .FlashConfig
 section provided within application");
/* End of Kinetis Flash Configuration data */
```

Important Note: The support for placing the Flash Configuration Block can be disabled by unticking a checkbox of the Project Properties linker Target tab. However this is generally not advisable as it is very likely to result in a locked MCU.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

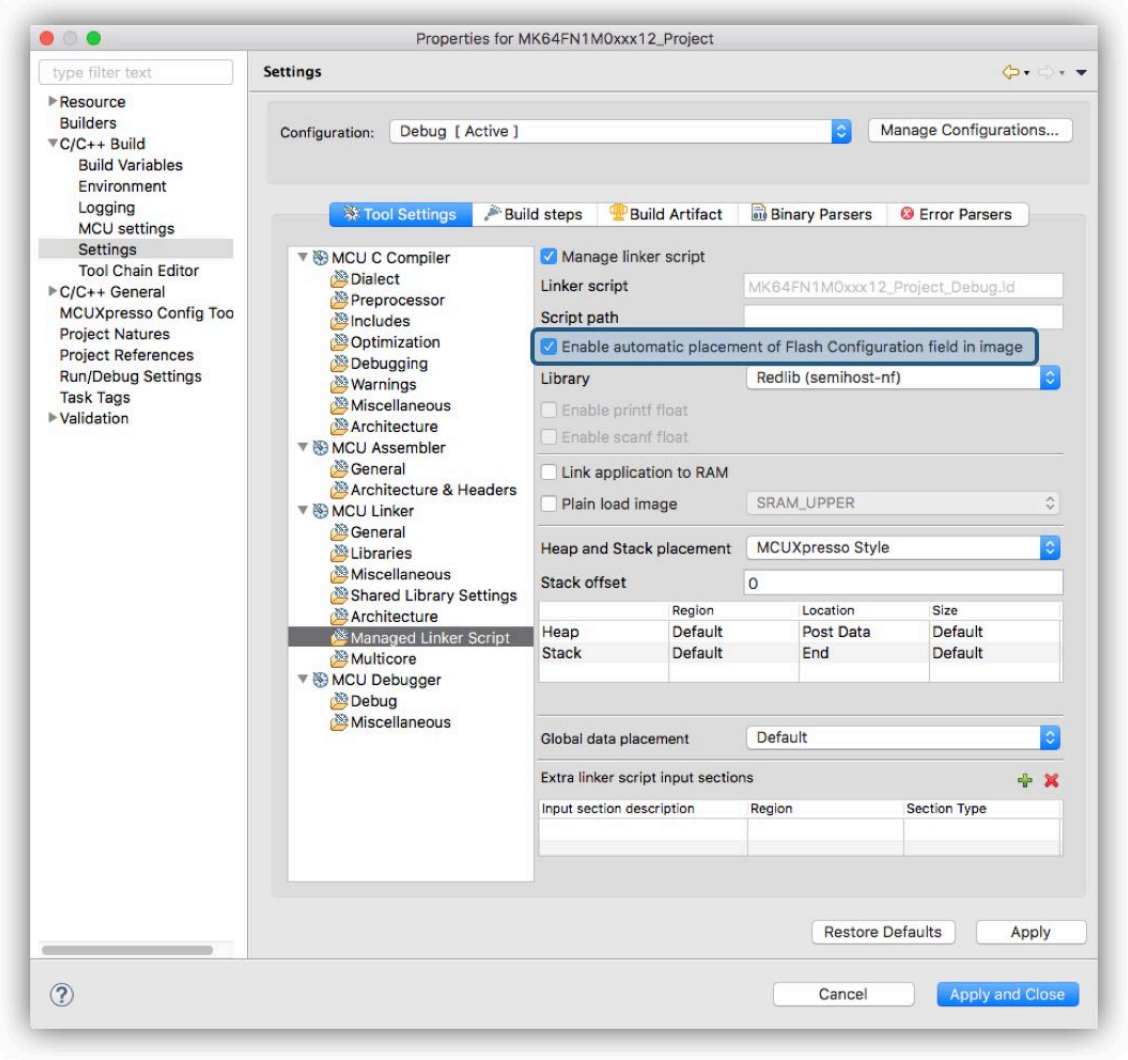


Figure 19.23. Linker Settings

19.8.3 Placement of USB Data

For MCUs where part support is imported from an SDK, the managed linker script mechanism supports the automatic placement of USB global data (as used by the SDK USB Drivers), including for parts with dedicated USB_RAM (small or large variants).

19.8.4 Plain Load Image

The LPC540xx family provides no built-in flash, but rather offers a quad SPI Flash Interface (SPIFI) so that external flash can be used. The most straight forward way of using external flash is that the image is built to be programmed into the external flash and executed directly from the same location (XIP – eXecute In Place).

However the LPC540xx boot rom also offers an alternative way of using the external flash – such that the application is programmed into the flash, but the boot rom will relocate it into a bank of the onboard SRAM for execution. Generally it is expected that the SRAMX bank (at address 0x0) will be used for this. An application that runs in this manner is known as a “plain load image”.

MCUXpresso IDE’s managed linker script mechanism offers a simple way of configuring an application project so that it will build as a plain load image. This can be controlled for a particular build configuration via:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Project -> Properties -> C/C++ Build -> Settings -> Tools Settings -> MCU Linker -> Managed Linker Script

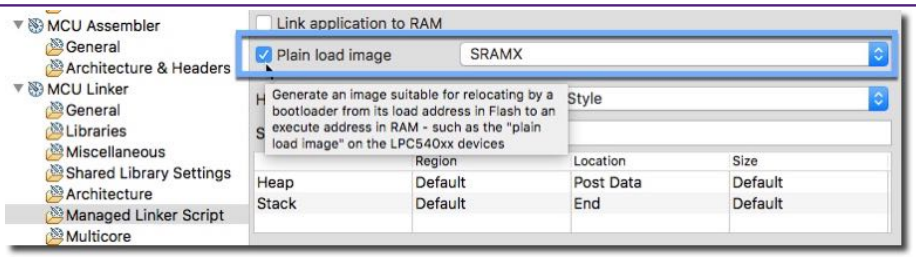


Figure 19.24. Plain Load Image

Please see also the Project Settings [155] shortcuts.

Enabling the “Plain load image” option will:

1. Modify the generated linker script so that the main code section is located so that it will be programmed into flash, but expect to be copied into specified RAM bank by the boot rom before being executed
2. Modify the startup code, using symbols provided from the generated linker script, so that the appropriate data is placed into the image so that the boot rom know that it needs to relocate the image from flash into RAM.

Note 1: This functionality requires the application project to be based on the LPC540xx part support from SDK v2.4.0 (or later).

Note 2: The size of the application image (including the initialised global data) must be less than the size of the RAM bank that the code will execute from.

Note 3: LPC540xx supports plain load images being executed from either address 0x0 or address 0x20000000. However if the RAM at 0x20000000 is used then the debugger will not be able to stop on the default breakpoint on main(). This is because a hardware breakpoint needs to be used (as the copying of the code from flash into RAM by the boot rom would overwrite a software breakpoint), but the Cortex-M4 cannot set a hardware breakpoint this high in the memory map.

19.8.5 Link Application to RAM

The MCUXpresso IDE *managed linker mechanism* defaults to placing the code and initialised data values to first Flash region listed within a project’s memory configuration as discussed in the Default Image Layout [201] section.

On occasion, it can be useful to debug a project directly from RAM since this offers some benefits such as avoiding the flash programming element of the debug session etc. Linking to RAM could be achieved by deleting the Flash memory regions from the project’s memory configuration and rebuilding the application – however this is not the most convenient approach!

Therefore MCUXpresso IDE offers the option to *tell* the managed linker script mechanism to simply ignore any flash regions listed in the project’s memory configuration via a simple checkbox at:

Project -> Properties -> C/C++ Build -> Settings -> Tools Settings -> MCU Linker -> Managed Linker Script

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

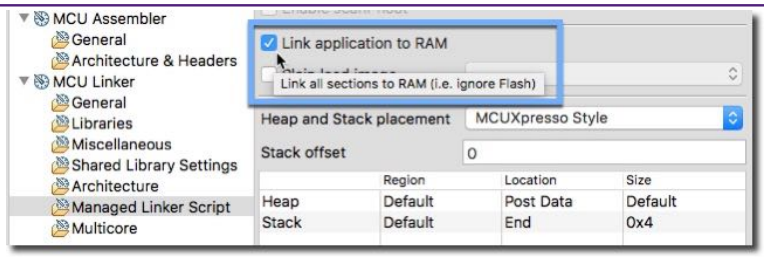


Figure 19.25. Link to RAM

Please see also the Project Settings [155] shortcuts.

With this option is set, the application will instead link to the first RAM region listed within the project's memory configuration.

There are two important considerations when developing with RAM based projects:

1. They require support from the debug environment to be run and so may not execute in the exactly the same manner as a true application running from an MCU reset. Please see the section RAM projects with LinkServer [271] for more information. **Please note**: if you are using debug solutions other than LinkServer, additional user setup may be required.
2. Unlike project running from Flash, global variable load and execute addresses will by default be the same. The consequence of this is that global variables values will persist at their current value if an application is restarted. Therefore this is not recommended, and instead a restart should be achieved by terminating and restarting the whole debug session. See also: Placement of specific code/data items [228]

Note: Some MCU/development boards make use of SDRAM. These memories are typically initialised by the MCU BootROM during reset and this initialisation may require user supplied configuration data to be programmed into flash. Therefore you **must** ensure that any SDRAM regions are correctly initialised before they are used for RAM based debug operations.

19.9 Modifying the Generated Linker Script / Memory Layout

The linker script generated by the managed linker script mechanism will be suitable for use, as is, for many applications. However in some circumstances you may need to make changes. MCUXpresso IDE provides a number of mechanisms to allow you to do this whilst still being able to use the managed linker script mechanism. These include:

- Changing the layout and order of memory using the Memory Configuration Editor
- Changing the size and location of the stack and heap using the Heap and Stack Editor
- Decorating the definitions of variables and functions in your source code with macros from the *cr_section_macros.h* to cause them to be placed into different memory blocks
- Providing project specific versions of FreeMarker linker script templates to change particular aspects of how the managed linker script mechanism creates the final linker script

The following sections describe these in more detail.

19.10 Using the Memory Configuration Editor

The Memory Configuration 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 MCUXpresso IDE itself (from installed or SDK part support).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

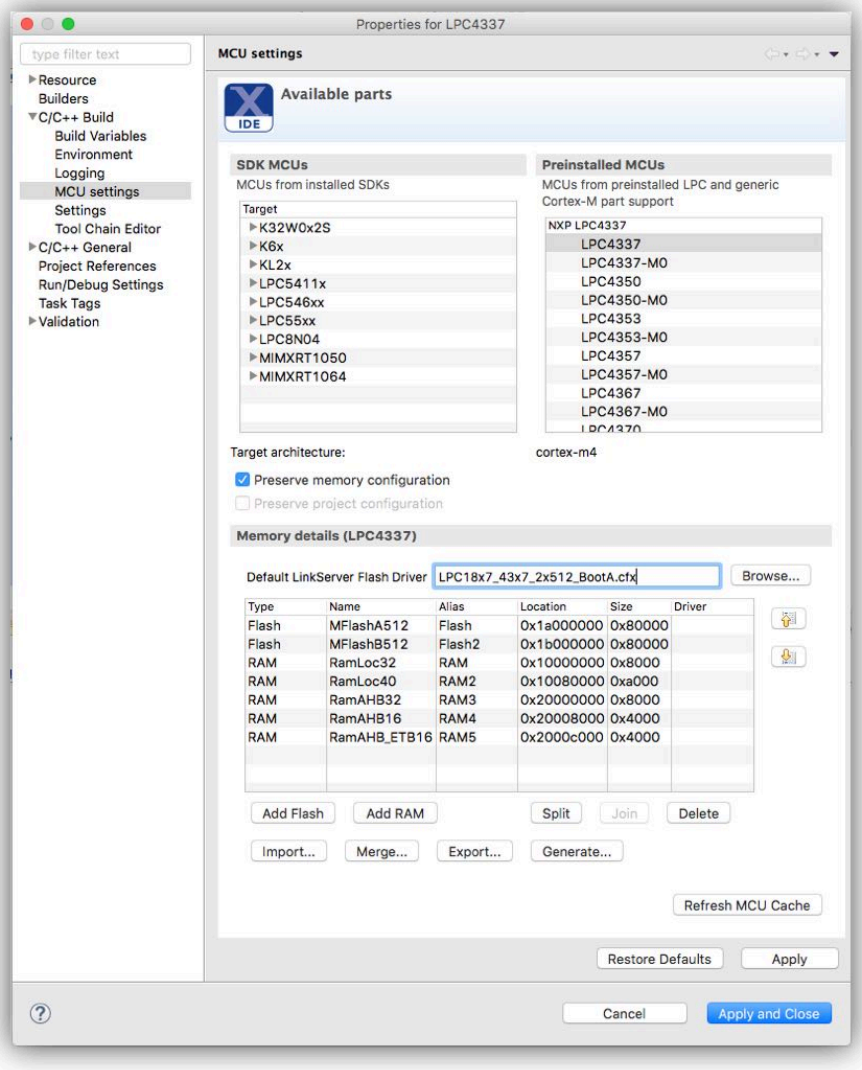


Figure 19.26. LPC4337... default memory regions

19.10.1 Editing a Memory Configuration

In the example below, we will show how the default memory configuration for an LPC4337... can be changed.

Introduced in MCUXpresso IDE version 10.3.0, the memory configuration can simply be edited in place to create the desired memory map.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

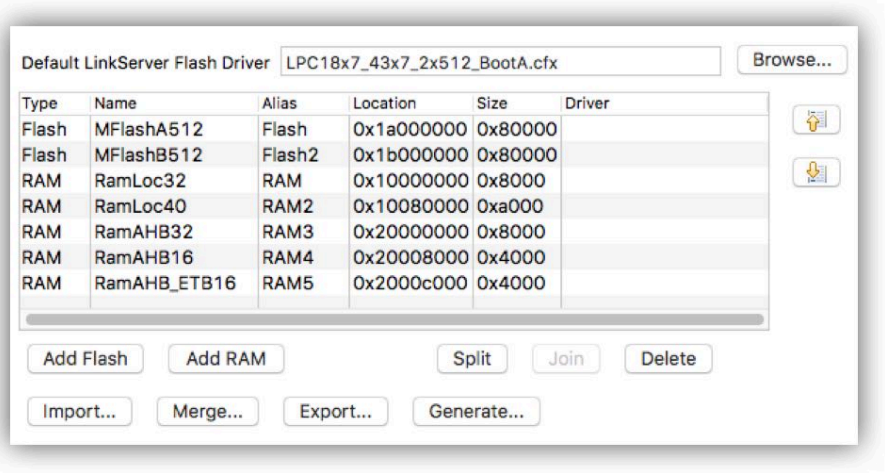


Figure 19.27. Memory configuration editor

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

For simplicity, the **additional** memory regions are given sequential aliases, starting from 2, so RAM2, RAM3 etc (as well as using their "formal" region name – for example RamAHB32).

Table 19.1. 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).
Generate	Generates local part support for the selected MCU.
Driver	Highlighted in blue, shows the selection of a per-Flash region Flash driver. Click this field to see a drop down of all available drivers. Please see: LinkServer Flash Support [173]
Browse(Flash driver)	Select the appropriate driver for programming the Flash memory specified in the memory configuration. For more information please see the section on flash drivers [173]

The name, location, and size of this new region can be edited in place. **Note**: 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`.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

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

The screenshot below shows the dialog after the “Add Flash” button has been clicked. Use the highlighted up/down buttons to move this region to be top in the list. This action forces MCUXpresso IDE’s managed linker script mechanism to link against this new flash region.

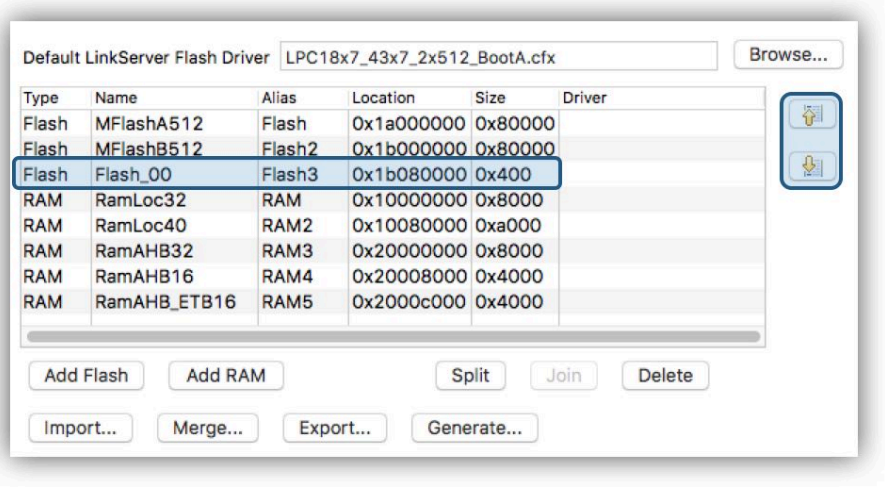


Figure 19.28. Effect of Add Flash



Tip

Once a change has been made, ensure a mouse click is made outside any changed *cell*, this action will force the change to be recognised by Eclipse

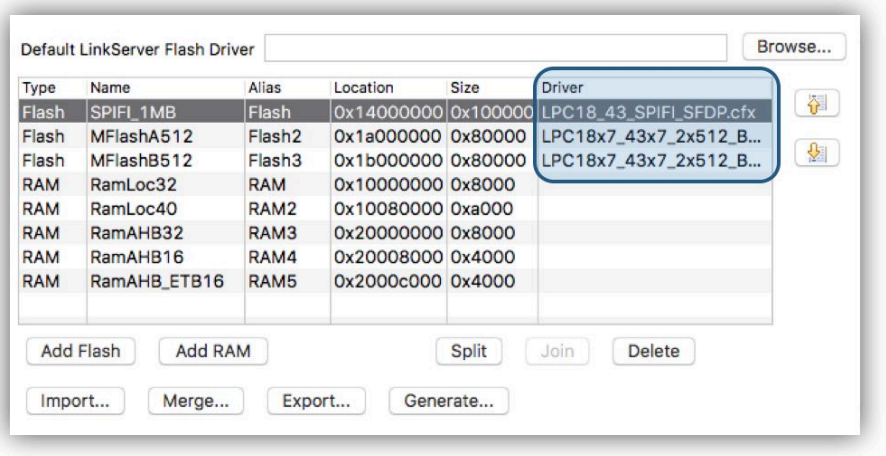


Figure 19.29. Updated MCU settings

Here you can see that the new region has been named SPIFI_1MB, its base address set to 0x14000000, its size to 1MB and the default Flash driver has been deleted and an SFDP SPIFI driver selected for the newly created SPIFI_1MB region.

MCUXpresso IDE provides extended support for the creation and programming of projects that span multiple Flash devices. In addition to a single default Flash driver, per region Flash drivers can also be specified (as above). Using this scheme projects can be created that span Flash regions and can be programmed in a single ‘debug’ operation.

Note: Once the memory details have been modified, the selected MCU as displayed on the “Status Bar” (at the bottom of the IDE window) will be displayed with an asterisk (*) next to it.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

This provides an indication that the MCU memory configuration settings for the selected project have been modified.

19.10.2 Device specific vs Default Flash Drivers

When a project is configured to use additional Flash devices via the Memory Configuration Editor, the Flash driver to be used for programming that Flash device has to be specified in the Driver column. Typically for a SPIFI device, this should be:

- *LPC18_43_SPIFI_GENERIC.cfx* (for *LPC18/LPC43* series MCUs)
- *LPC40xx_SPIFI_GENERIC.cfx* (for *LPC407x/8x* MCUs)
- *LPC5460x_SPIFI_GENERIC.cfx* (for *LPC5460x* MCUs).
- *LPC540xx_SPIFI_GENERIC.cfx* (for *LPC540xx* MCUs).

For further information please also see the section on flash drivers. [173]

19.10.3 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.

19.10.4 Copying Memory Configurations

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

19.11 Global Data Placement

By default, global data items are located at run time in the ‘default’ memory region (i.e. the first RAM block displayed in the memory configuration area).

However, MCUXpresso IDE version 10.2 introduced a mechanism to the Managed Linker Script mechanism to allow the user to specify a specific memory region to be used for the global data, without the need to change the order of the RAM blocks in the memory configuration editor.

This can be done via the Managed Linker Script page of Project Properties:

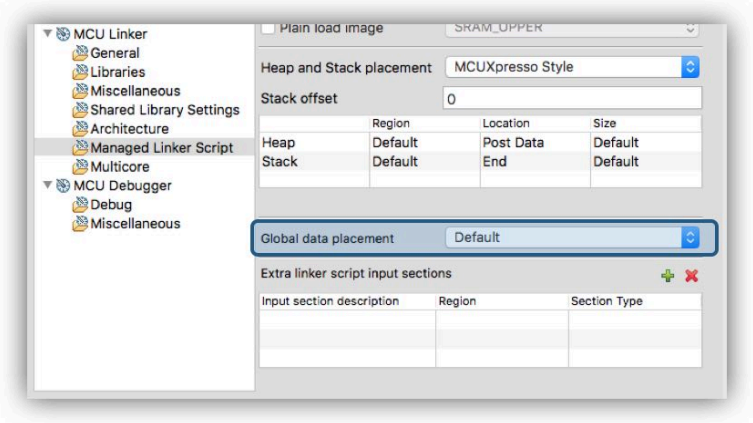


Figure 19.30. MCUXpresso IDE Global Data Placement

To change the memory region to be used, simply use the drop down box to select the memory region you wish to locate the global data.

Note: the above placement of global data applies to global data items that are not explicitly placed elsewhere in the memory map see: Placement of specific code/data items. [228]

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

19.12 Modifying heap/stack placement

MCUXpresso IDE provides two models of heap/stack placement. The first of these is the "LPCXpresso Style", which is the mechanism provided by the previous generation LPCXpresso IDE. This is the default model used for projects created for Preinstalled MCUs. The second model is the "MCUXpresso style". This is the default model used for projects created for MCUs imported from SDKs.

The heap/stack placement model being used for a particular project/build configuration can be modified by right clicking on the project and selecting:

Project Properties -> C/C++ Build -> Settings -> MCU Linker -> Managed Linker Scripts

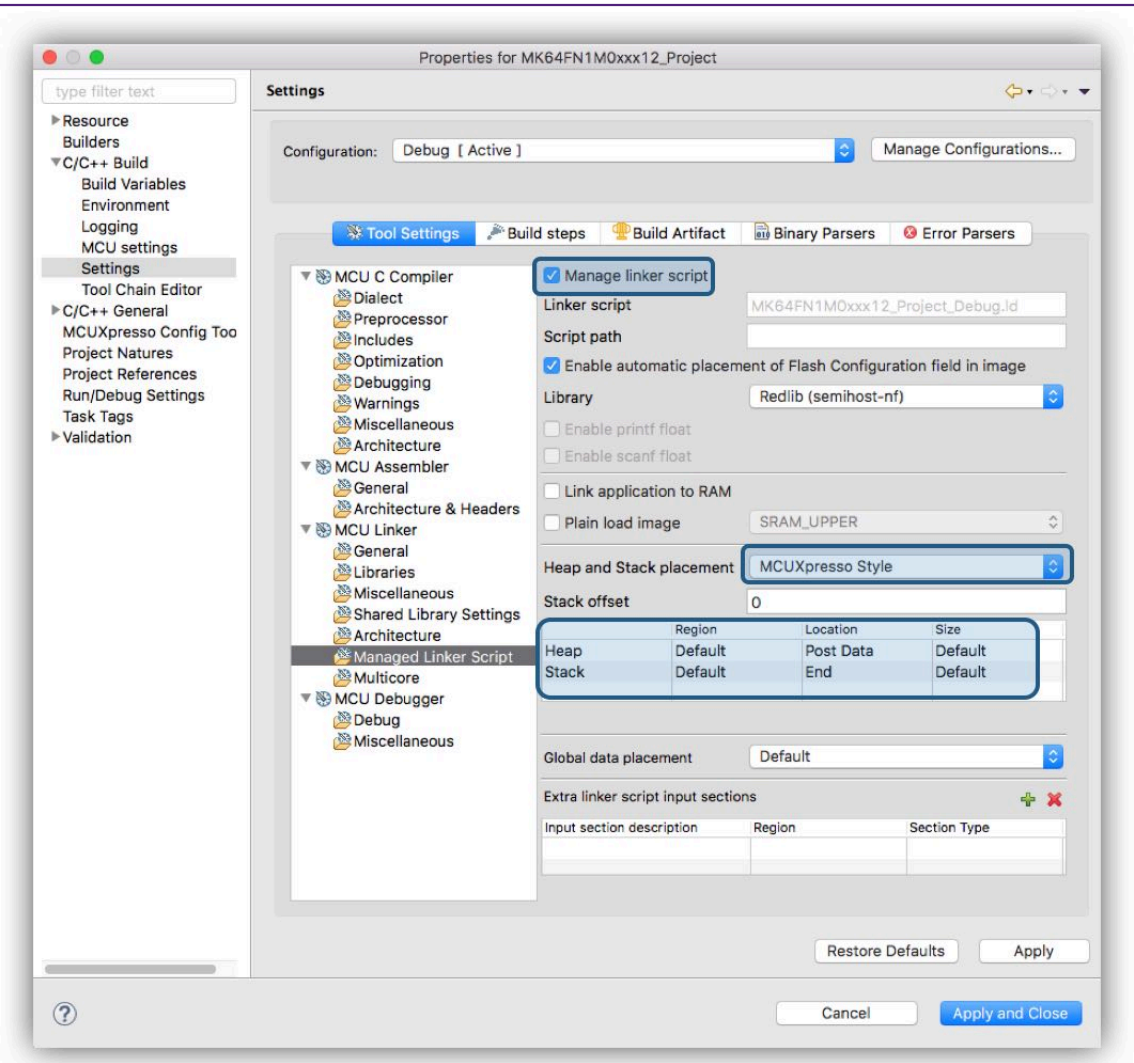


Figure 19.31. MCUXpresso IDE Linker Settings

In the dialogue above, highlights show the managed linker script option along with the selection of the MCUXpresso Style scheme.

19.12.1 MCUXpresso style Heap and Stack

By default the heap and stack are placed in the "default" memory region (i.e. the first RAM block displayed in the memory configuration area), with the heap placed after the application's data and the stack rooted at the top of this block.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

However, using the Heap and Stack editor in Project Properties, it is very simple to individually change the stack and heap locations (both the memory block used, and the location within that block), and also the size of the memory to be used by each of them.

Region

- Default : Place into first RAM bank as shown in Memory Configuration Editor
- List of memory regions, and aliases, as show in Memory Configuration Editor

Location

- Start : Place at start of specified RAM bank.
- Post Data : Place after any data in specified RAM bank. Default for heap.
- End : Place at end of specified RAM bank. Default for stack.

Size

- Default: 1/16th of the memory region size, up to a maximum of 4KB (and a minimum of 128bytes). Hovering the cursor over the field will show the current value that will be used.
- Value : Specify exact required size. Must be a multiple of 4. **Note**: 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 size of 32KB, enter 32768, 0x8000 or 32k.
  - A value of 0 can be entered to prevent any heap use by an application.
    - **Note**: For semihosted printf to operate without any heap space, you must enable the “character only” version. For Redlib, define the symbol “CR_PRINTF_CHAR” (at the project level) and remove other semihosting defines such as CR_INTEGER_PRINTF. Character only semihosted printf is significantly slower than the default version and may display differently depending on your debug solution.

Note: The MCUXpresso style of setting heap and stack has the advantage over the LPCXpresso style described below in that the memory allocated for heap/stack usage is also taken into account in the image size information displayed in the Build console when your project is built.

19.12.2 LPCXpresso style Heap and Stack

By default the heap and stack are still placed in the “default” memory region (i.e. the first RAM block displayed in the memory configuration area), with the heap placed after the application’s data and the stack rooted at the top of this block.

To relocate the stack or heap, or provide a maximum extent of the heap, then the linker “--defsym” option can be used to define one or more of the following symbols:

```
__user_stack_top
__user_heap_base
_pvHeapLimit
```

To do this, use the *MCU Linker -> Miscellaneous -> Other Options* box in Project Properties.

For example:

--defsym=__user_stack_top=__top_RAM2

- Locate the stack at the top of the second RAM bank (as listed in the memory configuration editor)
- **Note**: The symbol __top_RAM2 is defined in the project by the managed linker script mechanism at:

```
<projname>_<buildconfig>_mem.ld
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

--defsym=__user_heap_base=__end_bss_RAM2

- Locate the start of the heap in the second RAM bank, after any data that has been placed there

--defsym=_pvHeapLimit=__end_bss_RAM2+0x8000

- Locate the end of the heap in the second RAM bank, offset by 32KB from the end of any data that has been placed there

--defsym=_pvHeapLimit=0x10004000

- Locate the end of the heap at the absolute address 0x10004000

19.12.3 Reserving RAM for IAP Flash Programming

The IAP Flash programming routines available in NXP's LPC MCUs generally make use of some of the onchip RAM when executed. For example on the LPC1343 the top 32 bytes of onchip RAM are used. Thus if you are calling the IAP routines from your own application, you need to ensure that this memory is not used by your main application – which typically means by the stack.

However, with the managed linker script mechanism, it is easy to modify the start position of the stack (remember that stacks grow down) to avoid this clash with the IAP routines. To do this go to:

Project Properties -> C/C++ Build -> Settings -> MCU Linker -> Manager Linker Script

and modify the value in the "Stack Offset" field from 0 to 32. This will work whether you are using LPCXpresso style or MCUXpresso style of heap/stack placement.

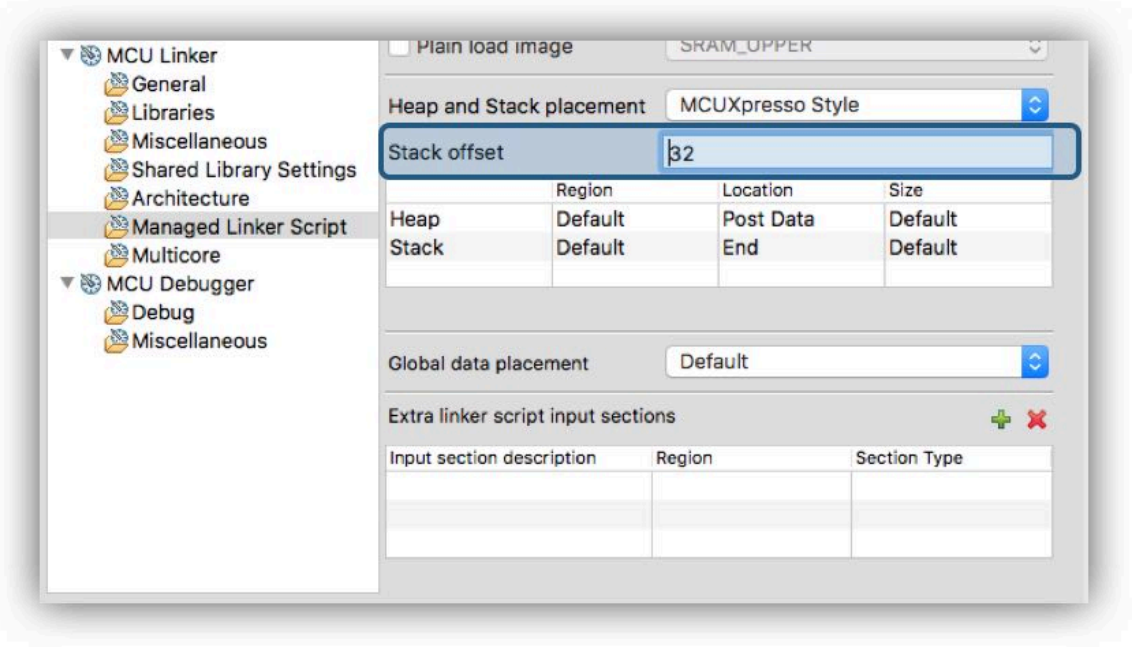


Figure 19.32. MCUXpresso IDE Linker Reserve Stack Space

The value you enter in this field must be a multiple of 4.

You are also advised to check the documentation for the actual MCU that you are using to confirm the amount of memory required by the IAP routines.

19.12.4 Stack Checking

Although, as described above, it is possible to define a size of memory to be used for the stack, Cortex-M CPUs have no support for hardware stack checking. Thus if you want to automatically

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

detect if the stack exceeds the memory set aside for it – other mechanisms must be used. For example:

- Identify a suitable memory region (or portion of one) that will fault for accesses below the regions base address, then locate the stack as desired within this region and watch for a possible fault
- Include code that sets the stack to a known value, and periodically checks whether the lowest address has been overwritten
- When debugging, set a watchpoint on the lowest address the stack is allowed to reach
- Use the Memory Protection Unit (MPU) to detect overflow, on parts which implement one

19.12.5 Heap Checking

By default, the heap used by the malloc() family of routines grows upwards from the end of the user data in RAM up towards the stack – a “one region memory model”.

When a new block of memory is requested, the memory allocation function _sbrk() will make a call to the following function to check for heap overflow:

```
unsigned __check_heap_overflow (void * new_end_of_heap)
```

This should return:

- **1** - If the heap will overflow
- **0** - If the heap is still OK

If 1 is returned, Redlib’s malloc() will set errno to ENOMEM and return a null pointer to the caller

The default version of **__check_heap_overflow()** built into MCUXpresso IDE supplied C libraries carry out no checking unless the symbol “_pvHeapLimit” has been created in your image, to mark the end location of the heap.

This symbol will have been created automatically if you are using the MCUXpresso style of heap and stack placement described earlier in this chapter. Or alternatively if using the LPCXpresso style of heap and stack placements, you can use the --defsym option to set this.

If you wish to use a different means of heap overflow checking, then you can find a reference implementation of **__check_heap_overflow()** in the file **_cr_check_heap.c** that can be found in the Examples subdirectory of your IDE installation.

This file also provides functionality to allow simple heap overflow checking to be done by looking to see if the heap has reached the current location of the stack point, which of course assumes that the heap and stack are in the same region. This check is not enabled by default implementation within the C library as it can break in some circumstances – for example when the heap is being managed by an RTOS.

19.12.6 Checking the Heap from your Application

The symbol **__end_of_heap** indicates the current end of the heap and can be used by user code to track heap usage. For instance:

```
extern unsigned int __end_of_heap;
:
end_of_heap = __end_of_heap;
myBuffptr=(uint32_t*)malloc(20*sizeof(uint32_t));
new_end_of_heap = __end_of_heap;
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

However it should be noted that the location this points to includes any last block that has been free'd. In other words it effectively provides the maximal extent of the heap so far, not the end of the currently "active" last block.

Thus in some cases, if you check __end_of_heap before calling malloc(), then again afterwards, it is possible that the value will not change if the heap request can be fulfilled using the free'd last block i.e. there is no need to extend the heap further. In certain cases, __end_of_heap can reduce, for example if a block at the end of the heap is freed and a smaller block is subsequently allocated.

19.13 Placement of specific code/data Items

It is possible to make changes to the placement of specific code/data items within the final image without modifying the FreeMarker linker script templates. Such placement can be controlled via macros provided in an MCUXpresso IDE supplied header file which can be pulled into your project using:

```
#include <cr_section_macros.h>
```

Alternatively Introduced in MCUXpresso IDE version 10.2, the managed linker script mechanism now also provides a means of placing arbitrarily named code or data sections into a specified memory region of the generated image and is described in the next section. (See also Global data placement). [223]

19.13.1 Placing code and data into different Memory Regions

Unlike the macros provided by cr_section_macros.h (described later), this method does not require any change to the source code declaring the affected code/data (which basically rename the generated code/data sections to match the memory region name). And in many cases it can avoid the need to provide project local FreeMarker linker script templates (described later in this chapter).

To place the code or data, you simply need to add the details of the section name, the memory region to place it in, and the type of the code/data, as per the below screenshot(s):

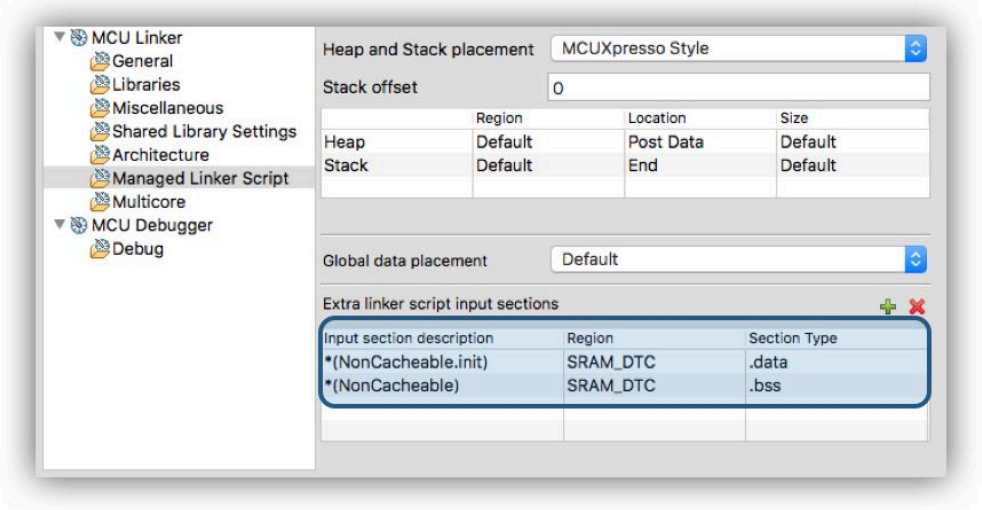


Figure 19.33. Adding an Extra Linker Section

which will modify the generated linker script to contain the sections specified in the appropriate region:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

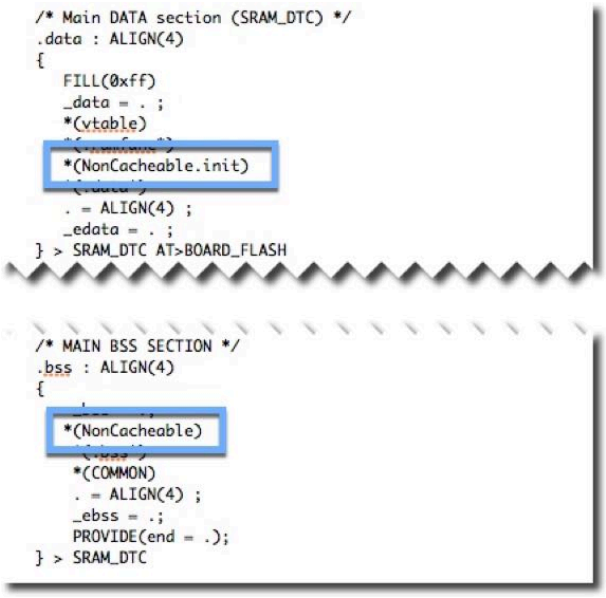


Figure 19.34. Extra Linker Section Script

The second example graphic shows both the placement of a constant data table and also the powerful technique of specifying a project source folder and placing the entire contents of that folder (flash2's .text sections) into a chosen flash device. Using this scheme the user can drag and drop source files within the project structure to choose which location will be used for their linkage and so their flash storage.

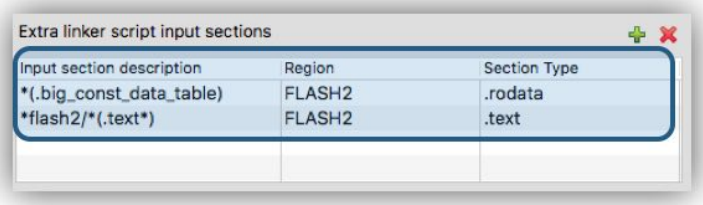


Figure 19.35. Adding an Extra Linker 2 Section

Note: that the format of the "input section description" is as detailed in the GNU Linker documentation, which can be found within the IDE's built-in help system :

Help -> Help Contents -> Tools (Compilers, Debugger, Utilities) -> GNU Linker -> Linker Scripts -> SECTIONS Command -> Input Section Description

or directly in the online GNU documentation at:

https://sourceware.org/binutils/docs/ld/Input-Section-Basics.html

Also, this functionality only allows you to add sections to the linker script, not to remove something that the managed linker script already puts in. Thus if you need to remove part of the generated linker script's contents – then you will still need to modify the underlying FreeMarker linker script templates.

Finally, remember that the GNU linker script mechanism functions such that the first match encountered for a section will win (not the best match found). Thus this mechanism is just a request, not a guarantee. Always check the generated linker script and the map file output by the link step to confirm the expected placement of sections. In some problem cases, you may

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

be able to force the required placement by use of an EXCLUDE in one memory region, as well as the section in the required region.

19.13.2 Placing data into different RAM blocks using Macros

Many MCUs provide more than one bank of RAM. By default the managed linker script mechanism will place all of the application data and bss (as well as the heap and stack) into the first bank of RAM.

However it is also possible to place specific data or bss items into any of the defined banks for the target MCU, as displayed in the Memory Configuration Editor, by decorating their definitions in your source code with macros from the cr_section_macros.h MCUXpresso IDE supplied header file

For simplicity, the **additional** memory regions are named sequentially, starting from 2, so RAM2, RAM3 etc (as well as using their “formal” region name – for example RamAHB32).

For example, the LPC1768 has a second bank of RAM at address 0x2007c000. The managed linker script mechanism creates a data (and equivalent bss) load section for this region thus:

```
.data_RAM2 : ALIGN(4)
{
    FILL(0xff)
    *(.data.$RAM2*)
    *(.data.$RamAHB32*)
} > RamAHB32 AT>MFlash512
```

To place data into this section, you can use the __DATA macro, thus:

```
// create an unitialised 1k buffer in RAM2
__DATA(RAM2) char data_buffer[1024];
```

Or the __BSS macro:

```
// create a zero-init buffer in RAM2
__BSS(RAM2) char bss_buffer[128];
```

In some cases you might need a finer level of granularity than just placing a variable into a specific memory bank, and rather need to place it at a specific address. In such a case you could then edit the predefined memory layout for your particular project using the “Memory Configuration Editor” to divide up (and rename) the existing banks of RAM. This then allows you to provide a specific named block of RAM into which to place the variable that you need at a specific address, again by using the attribute macros provided by the “cr_section_macros.h” header file.

19.13.3 Noinit Memory Sections

Normally global variables in an application will end up in either a “.data” (initialized) or “.bss” (zero-initialized) data section within your linked application. Then when your application starts executing, the startup code will automatically copy the initial values of “.data” sections from Flash to RAM, and zero-initialize “.bss” data sections directly in RAM.

MCUXpresso IDE’s managed linker script mechanism also supports the use of “.noinit” data within your application. Such data is similar to “.bss” except that it will not get zero-initialized during startup.

Note: Great care must be taken when using “.noinit” data such that your application code makes no assumptions about the initial value of such data. This normally means that your application

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

code will need to explicitly set up such data before using it – otherwise the initial value of such a global variable will basically be random (i.e. it will depend upon the value that happens to be in RAM when your system powers up).

One common example of using such .noinit data items is in defining the frame buffer stored in SDRAM in applications which use an onchip LCD controller (for example NXP LPC178x and LPC408x parts).

Making global variables Noinit

The linker script generated by MCUXpresso IDE's managed linker script mechanism will contain a section for each RAM memory block to contain ".noinit" items, as well as the ".data" and ".bss" items. **Note**: For a particular RAM memory block, all ".data" items will be placed first, followed by ".bss" items, and then ".noinit" items.

However, normally for a particular RAM memory block where you are going to be put ".noinit" items, you would actually be making all of the data placed into that RAM ".noinit".

The "cr_section_macros.h" header file then defines macros which can be used to place global variables into the appropriate ".noinit" section. First of all include this header file:

```
#include <cr_section_macros.h>
```

The __NOINIT macro can then be used thus:

```
// create a 128 byte noinit buffer in RAM2
__NOINIT(RAM2) char noinit_buffer[128];
```

And if you want ".noinit" items placed into the default RAM bank, then you can use the __NOINIT_DEF macro thus:

```
// create a noinit integer variable in the main block of RAM
__NOINIT_DEF int noinit_var ;
```

19.13.4 Placing code/rodata into different FLASH Blocks

Most MCUs only have one bank of Flash memory. But with some parts more than one bank may be available – and in such cases, by default, the managed linker script mechanism will still place all of the application code and rodata (consts) into the first bank of Flash (as displayed in the Memory Configuration Editor).

For example:

- most of the LPC18 and LPC43xx parts containing internal Flash (such as LPC1857 and LPC4357) actually provide dual banks of Flash.
- some MCUs have the ability to access external Flash (typically SPIFI) as well as their built-in internal Flash (e.g. LPC18xx, LPC40xx, LPC43xx, LPC546xx).

However it is also possible to place specific functions or rodata items into the second (or even third) bank of Flash. This placement is controlled via macros provided in the *"cr_section_macros.h"* header file.

For simplicity, the **additional** Flash region can be referenced as Flash2 (as well as using its "formal" region name – for example MFlashB512 – which will vary depending upon part).

First of all include this header file:

```
#include <cr_section_macros.h>
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Then, for example, to place a rodata item into this section, you can use the __RODATA macro, thus:

```
__RODATA(Flash2) const int roarray[] = {10,20,30,40,50};
```

Or to place a function into it you can use __TEXT macro:

```
__TEXT(Flash2) void systick_delay(uint32_t delayTicks) {
 :
}
```

In addition, the __RODATA_EXT and __TEXT_EXT macros can be used to place functions/ rodata into a more specifically named section, for example:

```
__TEXT_EXT(Flash2,systick_delay) void systick_delay(uint32_t delayTicks) {
 :
}
```

will be placed into the section ".text.$Flash2.systick_delay" rather than ".text.$Flash2".

19.13.5 Placing specific functions into RAM Blocks

In most modern MCUs with built-in Flash memory, code is normally executed directly from Flash memory. Various techniques, such as prefetch buffering are used to ensure that code will execute with minimal or zero wait states, even a higher clock frequencies. Please see the documentation for the MCU that you are using for more details.

However it is also possible to place specific functions into any of the defined banks of RAM for the target MCU, as displayed in:

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

and sometimes there can be advantages in relocating small, time critical functions so that they run out of RAM instead of Flash.

For simplicity, the **additional** memory regions are named sequentially, starting from 2, (as well as using their "formal" region name – for example RamAHB32). So for a device with 3 RAM regions, alias names RAM, RAM2 and RAM3 will be available.

This placement is controlled via macros provided in a header file which can be pulled into your project using:

```
#include <cr_section_macros.h>
```

The macro __RAMFUNC can be used to locate a function into a specific RAM region.

For example, to place a function into the main RAM region, use:

```
__RAMFUNC(RAM) void fooRAM(void) {...
```

To place a function into the RAM2 region, use:

```
__RAMFUNC(RAM2) void fooRAM2(void) {...
```

Alternatively, RAM can be selected by formal name (as listed in the memory configuration editor), for example:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
__RAMFUNC(RamAHB32) void HandlerRAM(void) {...
```

In order to initialize RAM based code (and data) into specified RAM banks, the managed linker script mechanism will create a “Global Section Table” in your image, directly after the vector table. This contains the addresses and lengths of each of the data (and bss) sections, so that the startup code can then perform the necessary initialization (copy code/data from Flash to RAM) .

Long branch veneers and Debugging

Due to the distance in the memory map between Flash memory and RAM, you will typically require a “long branch veneer” between the function in RAM and the calling function in Flash. The linker can automatically generate such a veneer for direct function calls, or you can effectively generate your own by using a call via a function pointer.

One point to note is that debugging code with a linker generated veneer can sometimes cause problems. This veneer will not have any source level debug information associated with it, so that if you try to step in to a call to your code in RAM, typically the debugger will step over it instead.

You can work around this by single stepping at the instruction level, setting a breakpoint in your RAM code, or by changing the function call from a direct one to a call via a function pointer.

19.13.6 Reducing Code Size when support for LPC CRP or Kinetis Flash Config Block is Enabled

One of the consequences of the way that LPC CRP and Kinetis Flash Configuration Blocks work is that the memory between the CPU’s vector table and the CRP word/ Flash Config Block is often left largely unused. This can typically increases the size of the application image by several hundred bytes (depending upon the MCU being used).

However this unused space can easily be reclaimed by choosing one or more functions to be placed into this unused memory. To do this, you simply need to decorate their definitions with the macro __AFTER_VECTORS which is supplied in the “cr_section_macros.h” header file

Obviously in order to do this effectively, you need to identify functions which will occupy as much of this unused memory as possible. The best way to do this is to look at the linker map file.

MCUXpresso IDE startup code already uses this macro to place the various initialization functions and default exception handlers that it contains into this space, thus reducing the ‘default’ unused space. But you can also place additional functions there by decorating their definitions with the macro, for example

```
__AFTER_VECTORS void myStartupFunction(void);
```

Note: you will get a link error if the __AFTER_VECTORS space grows beyond the CRP/Flash Configuration Block (when this support is enabled):

```
myproj_Debug.ld:98 cannot move location counter backwards (from 00000334
to 000002fc)
collect2: ld returned 1 exit status
make: *** [myproj.axf] Error 1
```

In this case, you will need to remove the __AFTER_VECTORS macro from the definition of one or more of your functions.

19.14 FreeMarker Linker Script Templates

By default, MCUXpresso IDE projects use a managed linker script mechanism which automatically generates a linker script file without user intervention – allowing the project code

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

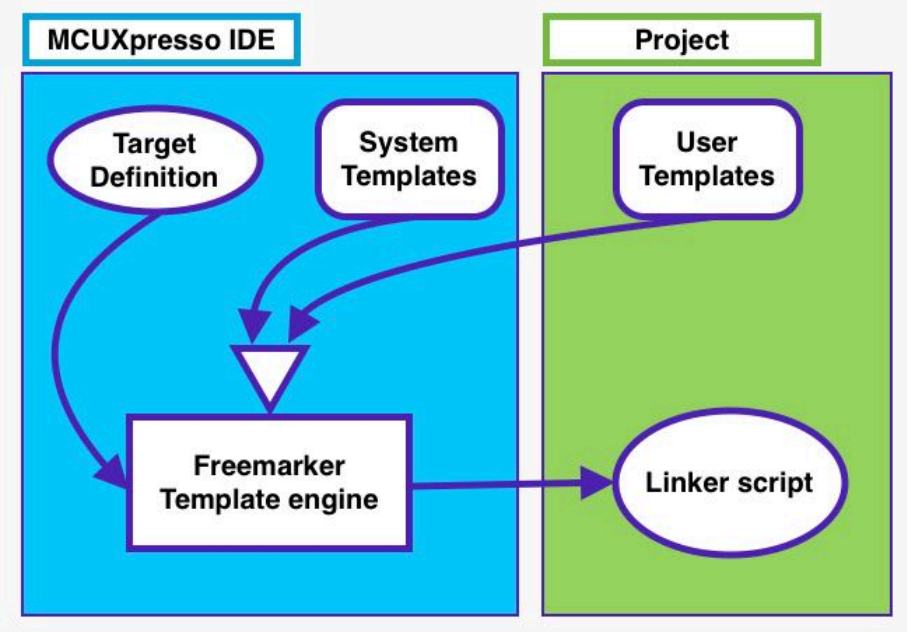
and data to be laid out in memory based on the IDE's knowledge of the memory layout of the target MCU.

However sometimes the linker script generated in this way may not provide exactly the memory layout required. MCUXpresso IDE therefore provides a highly flexible and powerful linker script template mechanism to allow the user to change the content of the linker script generated by the managed linker script mechanism

19.14.1 Basics

FreeMarker is a template engine: a generic tool to generate text output (HTML web pages, e-mails, configuration files, source code, etc.) based on templates and changing data. Built into MCUXpresso IDE are a set of templates that are processed by the FreeMarker template engine to create the linker script. Templates are written in the FreeMarker Template Language (FTL), which is a simple, specialized language, not a full-blown programming language like PHP. Full documentation for FreeMarker can be found at https://freemarker.org/docs/index.html .

MCUXpresso IDE automatically invokes FreeMarker, passing it a data model that describes the memory layout of the target together with a 'root' template that is processed to create the linker script. This root template, #includes further 'component' templates. This structure allows a linker script to be broken down into various components, and allows a user to provide their own templates for a component, instead of having to (re-)write the whole template. For example, component templates are provided for text, data and bss sections, allowing the user to provide a different implementations as necessary, but leaving the other parts of the linker script untouched.



19.14.2 Reference

FreeMarker reads input files, copying text and processing FreeMarker directives and 'variables', and writes an output file. As used by MCUXpresso IDE's managed linker script mechanism, the input files describe the various components of a linker script which, together with variables defined by the IDE, are used to generate a complete linker script. Any of the component template input files may be overridden by providing a local version in the project.

The component template input files are provided as a hierarchy, shown below, where each file #includes those files nested below. This allows for individual components of the linker script to be overridden without having to supply the entire linker script, increasing flexibility, while maintaining the benefits of Managed Linker Scripts.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Linker script template hierarchy

linkscript.ldt (top level)

- **user.ldt** (an empty file designed to be overridden by users that is included in linkscript, memory and library templates)
- **user_linkscript.ldt** (an empty file designed to be overridden by users that is included in linkscript only)
- **linkscript_common.ldt** (root for main content)
- header.ldt (the header for scripts)
  - listvars.ldt (a script to output a list of all predefined variables available to the template)
- includes.ldt (includes the memory and library scripts)
- section_top.ldt (top of the linker script SECTION directive)
- sgstubs_fixed.ldt (allow absolute veneer table for TrustZone application)
- text_section.ldt (text sections for each secondary Flash)
  - text_section_multicore.ldt (text sections for multicore targets)
    - text_section_multicore_checks_partfamily.ldt (part specific sanity checks)
  - extrasections_text.ldt (additional linker sections [228])
  - text.ldt (for inserting *text)
  - extrasections_rodata.ldt (additional linker sections [228])
  - rodata.ldt (for inserting rodata)
- boot_hdr.ldt (allows placement of optional header before main code section)
  - boot_hdr_partfamily.ldt
- main_text_section.ldt (the primary text section)
  - global_section_table.ldt (the global section table)
  - crp.ldt (the CRP information)
  - flashconfig.ldt (flash security prototype)
  - extrasections_text.ldt (additional linker sections [228])
  - main_text.ldt (for inserting *text)
  - extrasections_rodata.ldt (additional linker sections [228])
  - freertos_debugconfig.ldt (to force placement of FreeRTOSDebugConfig rodata section)
  - main_rodata.ldt (read-only data)
  - cpp_info.ldt (additional C++ requirements)
- exdata.ldt (placement of LPCXpresso style exdata sections)
- sgstubs.ldt (allow absolute veneer table for TrustZone application)
- end_text.ldt (end of text marker)
- usb_ram_section.ldt (placement of SDK USB data structures)
- stack_heap_sdk_start.ldt (placement of MCUXpresso style heap/stack)
- data_section.ldt (data sections for secondary ram)
  - data_section_multicore.ldt (data sections for multicore targets)
    - data_section_multicore_checks_partfamily.ldt (part specific sanity checks)
  - extrasections_data.ldt (additional linker sections [228])
  - data.ldt (for inserting *data)
- mtb_default_section.ldt (special section for MTB (cortex-m0+ targets)
- uninit_reserved_section.ldt (uninitialised data)
- main_data_section.ldt (primary data section)
  - extrasections_data.ldt (additional linker sections [228])
  - main_data.ldt (for inserting *data)
- ecrp.ldt (Enhanced Code Read Protection support)
- bss_section.ldt (secondary bss sections)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

  - extrasections_bss.ldt (additional linker sections [228])
  - bss.ldt (for inserting *bss)
- main_bss_section.ldt primary bss section)
  - extrasections_bss.ldt (additional linker sections [228])
  - main_bss.ldt (for inserting *bss)
- noinit_section.ldt (no-init data)
  - extrasections_noinit.ldt (additional linker sections [228])
- noinit_noload_section.ldt (no-load data)
- exdata_sdk.ldt (placement of MCUXpresso style exdata sections)
  - data_section_exceptions_multicore_sdk.ldt (additional multicore exdata sections information)
  - text_section_exceptions_multicore_sdk.ldt (additional multicore exdata sections information)
- stack_heap_sdk_postdata.ldt (placement of MCUXpresso style heap/stack)
- stack_heap_sdk_end.ldt (placement of MCUXpresso style heap/stack)
- stack_heap.ldt (define the stack and heap)
- checksum.ldt (create the LPC checksum)
- image_size.ldt (provide basic symbols giving location and size of image)
- symbols.ldt (provide additional symbols needed to built image)
  - symbols_partfamily.ldt (part specific “symbol”)
- section_tail.ldt (immediately before the send of linker SECTION directive)

library.ldt (the standard libraries used in the application)

- user.ldt (an empty file designed to be overridden by users that is included in linkscript, memory and library templates)
- user_library.ldt (an empty file designed to be overridden by users that is included in library only)

memory.ldt (the memory map)

- user.ldt (an empty file designed to be overridden by users that is included in linkscript, memory and library templates)
- user_memory.ldt (an empty file designed to be overridden by users that is included in memory only)

Linker script search paths

Whenever a linker script template is used, MCUXpresso IDE will search in the following locations, in the order shown:

- *project/linkscripts*
- the searchPath global variable
  - The searchPath can be set in a script by using the syntax <#global searchPath=“c:/windows/path;d:/another/windows/path”>
    - each directory to search is separated by a semicolon ';'
- *mcuxpresso_install_dir/ide/Data/Linkscripts*
  - linker templates can be placed in this directory to override the default templates for an entire installation
- MCUXpresso IDE internally provided templates (not directly visible to users)

Thus, a project can simply override any template by simply creating a linkscript directory within the project and placing the appropriate template in there. Using the special syntax “super@” an overridden template can reference a file from the next level of the search path

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

e.g. <*#include "super@user.ldt"*>

Linker script templates

Copies of the default linker script templates used within MCUXpresso IDE can be found in the Wizards/linker directory within the MCUXpresso IDE installation. These can be used as the basis of any project local scripts you wish to write.

Predefined variables (macros)

List (sequence) variables (used in *#list*)

libraries[]

- list of the libraries to be included in the "lib" script
  - for example (Redlib nohost)

```
libraries[0]=libcr_c.a
libraries[1]=libcr_eabihelpers.a
```

configMemory[] list of each memory region defined in the memory map for the project. Each entry has the following fields defined

- name – the name of the memory region
- alias – the alias of the memory region
- location – the base address of the memory
- size – the size of the memory region
- sizek – the printable size of the memory region in k or M
- mcuPattern
- defaultRAM – boolean indicating if this is the default RAM region
- defaultFlash – boolean indication if this is the default Flash region
- RAM – boolean indicating if this is RAM
- Flash – boolean indicating if this is Flash

for example:

```
configMemory[0]= name=MFlashA512 alias=Flash location=0x1a000000
size=0x80000 sizek=512K bytes mcuPattern=Flash flash=true RAM=false
defaultFlash=true defaultRAM=false
```

```
configMemory[2]= name=RamLoc32 alias=RAM location=0x10000000
size=0x8000 sizek=32K bytes mcuPattern=RAM flash=false RAM=true
defaultFlash=false defaultRAM=true
```

Slaves[] list of the Secondaries in a Multicore project. This variable is only defined in Multicore projects. Each entry has the following fields defined

- name – name of the Secondary reference
- enabled – boolean indicating if this Secondary reference is enabled
- objPath – path to the object file for the Secondary image
- linkSection – name of the section this Secondary entity is to be linked in
- runtimeSection
- textSection – name of the text section
- textSectionNormalized – normalized name of the text section
- dataStartSymbol – name of the Symbol defining the start of the data
- dataEndSymbol – name of the Symbol defining the end of the data

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

for example:

```
slaves[0] = name=M0APP objectPath=${workspace_loc:/MCB4357_Blinky_DualM0/Debug
/MCB4357_Blinky_DualM0.axf.o}linkSection=Flash2 runtimeSection= textSection=
.core_m0app textSectionNormalized=_core_m0appdata StartSymbol=__start_data
dataEndSymbol=__end_data enabled=true;</notextile>
```

Simple variables include:

- CODE – name of the memory region to place the default code (text) section
- CRP_ADDRESS – location of the Code Read Protect value
- DATA – name of the memory region to place the default data section
- LINK_TO_RAM – value of the “Link to RAM” linker option
- STACK_OFFSET – value of the Stack Offset linker option
- FLASHn – defined for each FLASH memory
- RAMn – defined for each RAM memory
- basename – internal name of the process
- bss_align – alignment for .bss sections
- buildConfig – the name of the configuration being built
- chipFamily – the chip family
- chipName – name of the target chip
- data_align – alignment for .data section
- date – date string
- heap_symbol – name of the symbol used to define the heap
- isCppProject – boolean indicating if this is a C++ project
- isSlave – boolean indicating if this target is a Secondary – true if is a secondary core in a multicore system
- library_include – name of the library include file
- libtype – C library type
- memory_include – name of the memory include file
- mtb_supported – boolean indicating if mtb is supported for this target
- numCores – number of cores in this target
- procName – the name of the target processor
- project – the name of the project
- script – name of the script file
- slaveName – is the name of the Secondary (only present for Secondaries)
- stack_section – the name of the section where the stack is to be placed
- start_symbol – the name of the start symbol (entry point)
- scriptType – the type of script being generated (one of “script”, “memory”, or “library”)
- text_align – alignment for .text section
- version – product version string
- workspace_loc – workspace directory
- year – the year (extracted from the date)

Extended variables

Two ‘extended’ variables are available:

environment

- The environment variable makes the host Operating System environment variables available. For example, the Path variable is available as ${environment[“Path”]}.

Note Environment variables are case sensitive.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

systemProperties

- The Java system properties are available through the systemProperties variable. For example the “os.name” system property is available as ${systemProperties[“os.name”]}. **Note**: System properties are case sensitive.

Outputting variables

A list of all predefined variables and their values can be output to the generated linker script by setting the Preference: *MCUXpresso IDE -> Default Tool settings ->* … and list predefined variables in the script

A list of extended variables and their values can be output to the generated linker script by creating a *linkscripts/user.ldt* file in the project with the content

```
<#assign listvarsext=true>
```

(This is likely to be used less often, hence the slightly longer winded method of specifying the option)

19.15 FreeMarker Linker Script Template Examples

The use of FreeMarker linker script templates allows more wide ranging changes to be made to the generated link script than is possible using the *cr_section_macros.h* macros. The following examples provide some examples of this.

19.15.1 Relocating code from FLASH to RAM

If you have specific functions in your code base that you wish to place into a particular block of RAM, then the simplest way to do this is to decorate the function definition using the macro __RAMFUNC described earlier in this chapter.

However once you want to relocate more than a few functions, or when you don’t have direct access to the source code, this becomes impractical. In such case the use of FreeMarker linker script templates will be a better approach. The following sections provide a number of such examples.

Relocating particular objects into RAM

In some cases, it may be required to relocate all of the functions (and rodata) from a given object file in your project into RAM. This can be achieved by providing three linker script template files into a *linkscripts* folder within your project. For example if it was required that all code/rodata from the files foo.c and bar.c were relocated into RAM, then this could be achieved using the following linker script templates:

main_text.ldt

```
*(EXCLUDE_FILE(*foo.o *bar.o) .text*)
```

main_rodata.ldt

```
*(EXCLUDE_FILE(*foo.o *bar.o) .rodata)
*(EXCLUDE_FILE(*foo.o *bar.o) .rodata.*)
*(EXCLUDE_FILE(*foo.o *bar.o) .constdata)
*(EXCLUDE_FILE(*foo.o *bar.o) .constdata.*)
. = ALIGN(${text_align});
```

main_data.ldt

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
*foo.o(.text*)
*foo.o(.rodata .rodata.* .constdata .constdata.*)
*bar.o(.text*)
*bar.o(.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
*(.data*)
```

What each of these EXCLUDE_FILE lines (in main_text.ldt and main_rodata.ldt) is doing in pulling in all of the sections of a particular type (for example .text), except for the ones from the named object files. Then in main_data.ldt, we specify explicitly that the text and rodata sections should be pulled in from the named object files. **Note**: that with the GNU linker, LD, the first match found in the final generated linker script is always used, which is why the EXCLUDE_FILE keyword is used in the first two template files.

Note: EXCLUDE_FILE only acts on the closest input section specified, which is why we have 4 separate EXCLUDE_FILE lines in the main_rodata.ldt file rather than just a single combined EXCLUDE_LINE.

Once you have built your project using the above linker script template files, then you can check the generated .ld file to see the actual linker script produced, together with the linker map file to confirm where the code and rodata have been placed.

Relocating particular libraries into RAM

In some cases, it may be required to relocate all of the functions (and rodata) from a given library in your project into RAM. One example of this might be if you are using a flashless LPC43xx MCU with an external SPIFI Flash device being used to store and execute your main code from, but you need to actually update some data that you are also storing in the SPIFI Flash. In this case, the code used to update the SPIFI Flash cannot run from SPIFI Flash.

This can be achieved by providing three linker script template files into a *linkscripts* folder within your project. For example if it was required that all code/rodata from the library MYLIBRARYPROJ were relocated into RAM, then this could be achieved using the following linker script templates:

main_text.ldt

```
*(EXCLUDE_FILE(*libMYLIBRARYPROJ.a:) .text*)
```

main_rodata.ldt

```
*(EXCLUDE_FILE(*libMYLIBRARYPROJ.a:) .rodata)
*(EXCLUDE_FILE(*libMYLIBRARYPROJ.a:) .rodata.*)
*(EXCLUDE_FILE(*libMYLIBRARYPROJ.a:) .constdata)
*(EXCLUDE_FILE(*libMYLIBRARYPROJ.a:) .constdata.*)
. = ALIGN(${text_align});
```

main_data.ldt

```
*libMYLIBRARYPROJ.a:(.text*)
*libMYLIBRARYPROJ.a:(.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
*(.data*)
```

Note: When extending this example to more than one library, please mind the semantic of EXCLUDE_FILE directive which is pulling in all of the sections of a particular type, except for the ones from the named object files. Therefore, all of the library objects need to be specified

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

in EXCLUDE_FILE for a particular section type (in main_text.ldt and main_rodata.ldt templates). For example:

```
*(EXCLUDE_FILE(*libMYLIBRARYPROJ1.a *libMYLIBRARYPROJ2.a *libMYLIBRARYPROJ3.a:) .text*)
```

On the other hand, the library objects in main_data.ldt template, since they are not using EXCLUDE_FILE, need to be listed on separate lines:

```
*libMYLIBRARYPROJ1.a:(.text*)
*libMYLIBRARYPROJ2.a:(.text*)
*libMYLIBRARYPROJ3.a:(.text*)
```

Relocating majority of an application into RAM

In some situations, you may wish to run the bulk of your application code from RAM – typically just leaving startup code and the vector table in Flash. This can be achieved by providing three linker script template files into a *linkscripts* folder within your project:

main_text.ldt

```
*startup_*.o (.text.*)
*(.text.main)
*(.text.__main)
```

main_rodata.ldt

```
*startup_*.o (.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
```

main_data.ldt

```
*(.text*)
*(.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
*(.data*)
```

The above linker template scripts will cause the main body of the code to be relocated into the main (first) RAM bank of the target MCU, which by default will also contain data/bss, as well as the stack and heap.

Important Note: The code that performs this relocation is executed early within the reset handler (within *startup_xx* file). However, there is the potential for other critical functions to be called **before** this relocation is performed, for example *SystemInit()* may be called first to perform essential operations such as enabling RAM!

Any function that is called before the relocation is performed **must not** itself be relocated! For the specific case above, the following changes to main_text.ldt and main_rodata.ldt are required:

main_text.ldt

```
*startup_*.o (.text.*)
*system_*.o (.text.*)
*(.text.main)
*(.text.__main)
```

main_rodata.ldt

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
*startup_*.o (.rodata .rodata.* .constdata .constdata.*)
*system_*.o (.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
```

Finally, If the MCU being targeted has more than one RAM bank, then the main body of the code could be relocated into another RAM bank instead. For example, if you wanted to relocate the code into the second RAM bank, then this could be done by providing the following data.ldt file instead of the main_data.ldt above:

data.ldt

```
<#if memory.alias=="RAM2">
*(.text*)
*(.rodata .rodata.* .constdata .constdata.*)
. = ALIGN(${text_align});
</#if>
*(.data.$${memory.alias}*)
*(.data.$${memory.name}*)
```

Note: memory.alias value is taken from the Alias column of the Memory Configuration Editor.

19.15.2 Configuring projects to span multiple Flash Devices

Most MCUs only have one bank of Flash memory. But with some parts more than one bank may be available – and in such cases, by default, the managed linker script mechanism will still place all of the application code and rodata (consts) into the first bank of Flash (as displayed in the Memory Configuration Editor)..

For example

- most of the LPC18 and LPC43xx parts containing internal Flash (such as LPC1857 and LPC4357) actually provide dual banks of Flash.
- some MCUs have the ability to access external Flash (typically SPIFI) as well as their built-in internal Flash (e.g. LPC18xx, LPC40xx, LPC43xx, LPC546xx).

The macros provided in the “cr_section_macros.h” header file provide some ability to control the placement of specific functions or rodata items into the second (or even third) bank of Flash. However the use of FreeMarker linkers script templates allow this to be done in a much more powerful and flexible way.

One typical use case for this is a project which stores its main code and data in internal Flash, but additional rodata (for example graphics data for displaying on an LCD) in the external SPIFI Flash.

For instance, consider an example project where such rodata is all contained in a set of specific files, which we therefore want to place into the external Flash device. One very simple way to do this is to place such source files into a separate source folder within your project. You can then supply linker script templates to place the code and rodata from these files into the appropriate Flash.

For example, for a project using the LPC4337 with two internal Flash banks, plus external SPIFI Flash, if the source folder used for this purpose were called ‘spifidata’, then placing the following files into a *linkscripts* directory within your project would have the desired effect:

text.ldt

```
<#if memory.alias=="Flash3">
 *spifidata/*(.text*)
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
</#if>
*(.text_${memory.alias}*) /* for compatibility with previous releases */
*(.text_${memory.name}*) /* for compatibility with previous releases */
*(.text.$${memory.alias}*)
*(.text.$${memory.name}*)
```

rodata.ldt

```
<#if memory.alias=="Flash3">
*spifidata/*(.rodata*)
</#if>
*(rodata.$${memory.alias}*)
*(rodata.$${memory.name}*)
```

Note: the check of the memory.alias being Flash3 is to prevent the code/rodata items from ending up in the BankB Flash bank (which is Flash2 by default).

19.16 Disabling Managed Linker Scripts

It is possible to disable the managed linker script mechanism if required and provide your own linker scripts, but this is not recommended for most users. In most circumstance, the facilities provided by the managed linker script mechanism, and its underlying FreeMarker template mechanism should allow you to avoid the need for writing your own linker scripts. But if you do wish to do this, then untick the appropriate option at:

Properties -> C/C++ Build -> Settings -> MCU Linker -> Managed Linker Script

And then in the field *Script Path* provide the name and path (relative to the current build directory) of your own, manually maintained linker script.

In such cases you can either create your own linker script from scratch, or you can use the managed linker scripts as a starting point. One very important point though is that you are advised not to simply modify the managed linker scripts in place, but instead to copy them to another location and modify them there. This will prevent any chance of the tools accidentally overwriting them if at some point in the future you turn the managed make script mechanism back on.

Note: if your linker script includes additional files (as the managed linker scripts do), then you will also need to include the relative path information in the include inside the top level script file.

For more details of writing your own linker scripts, please see the GNU Linker (ld) documentation:

Help -> Help Contents -> Tools (Compilers, Debugger, Utilities) -> GNU Linker

There is also a good introduction to linker scripts available in Building Bare-Metal ARM Systems with GNU: Part 3 at:

https://www.embedded.com/design/mcus-processors-and-socs/4026080/Building-Bare-Metal-ARM-Systems-with-GNU-Part-3

See also the section on Enhanced Syntax Highlighting [209] to review editor assistance when manually creating Linker Scripts.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

20. Multicore Projects

Additional information can be found our the MCUXpresso IDE Community pages specifically see the blog articles:

LPC55xx Multicore Applications with MCUXpresso IDE, and also the article: *Using LPC55S69 SDK Trustzone examples with MCUXpresso IDE v11.0.0*

20.1 Introduction

Multicore MCUs can be designed in many ways, however within MCUXpresso IDE there is an underlying expectation that one core (the Primary) will control the execution (or at least the startup) of code running on other (Secondary) core(s).

Multicore application projects as described below consists of two (or more) linked projects – one project containing the Primary core's code and the other project(s) containing the Secondary core's code. The 'Primary' project contains a link to the 'Secondary' project which will cause the output image from the 'Secondary' to be included into the 'Primary' image when the Primary project is built. Building the Primary project will trigger the Secondary project to be built first.

After a power-on or Reset, the Primary core boots and is then responsible for booting the Secondary core. However, this relationship only applies to the booting process; after boot, an application may treat either of the cores as Primary or Secondary.

For this concept to work, the memory configurations of these related projects must be carefully managed to avoid unintended overlap or contention. One way this can be achieved is by linking the Secondary application to execute entirely from a RAM location unused by the Primary. Our automatic linkerscript generation will then locate the Secondary's code within the Primary's generated image, this code will be relocated to the correct RAM location by the Primary projects initialisation code at run time.

In practice, the Primary project's memory configuration will be the same as for a single core project, where as the Secondary project's memory configuration will be set to use a 'spare' or dedicated Secondary RAM region. In addition, a shared region may be used for communication between the CPUs.

Note: MCUs supporting dedicated Flash regions for each core can also be supported by this scheme, in such cases the Secondary project would simply be linked to the Secondary core's Flash location.

To complete the story ... the Primary project is debugged first, which will lead to the combined image being programmed into Flash and the Primary code executed. The Primary core's initialisation code will (in addition to other things) copy the Secondary core's code into RAM (if appropriate) and then stop on Main. When the Secondary project is debugged, the launch configuration will automatically be set to 'Attach' by the IDE since there is no need for this code to be programmed/downloaded by the debugger. When the Primary application is resumed, it will release the Secondary core and both projects can be debugged as required.

Important Note: Multicore MCUs may offer significant flexibility in how they can be used. The mechanism described above (as used in example projects) is not necessarily the only way (or even the best way) for a user's multicore projects to be configured. However, it has been chosen as the simplest and safest way to demonstrate the concepts and issues involved.

MCUXpresso IDE allows for the easy creation of "linked" projects that support the targeting of Multicore MCUs.

The rest of this chapter will describe the use of the LPC5411x multicore MCU, however the concepts discussed will be the same (or similar) for other multicore MCUs such as the LPC43xx and LPC5410x.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

20.2 Creating a Primary / Secondary project Pair (using an SDK)

The example described below is based around the LPC5411x multicore MCU using the LPCXpresso54114 SDK.

Note: Be sure to have installed the LPCXpresso54114 SDK into MCUXpresso IDE if you wish to follow this example.

20.2.1 Creating the M0 Secondary project

As discussed above, the Primary project's configuration will need to reference the Secondary project, therefore the Secondary project should be created first.

Launch the New Project Wizard and select the LPCXpresso54114 SDK board. Entering 54114 into the boards filter will reduce the number of boards to help selection, then click Next.

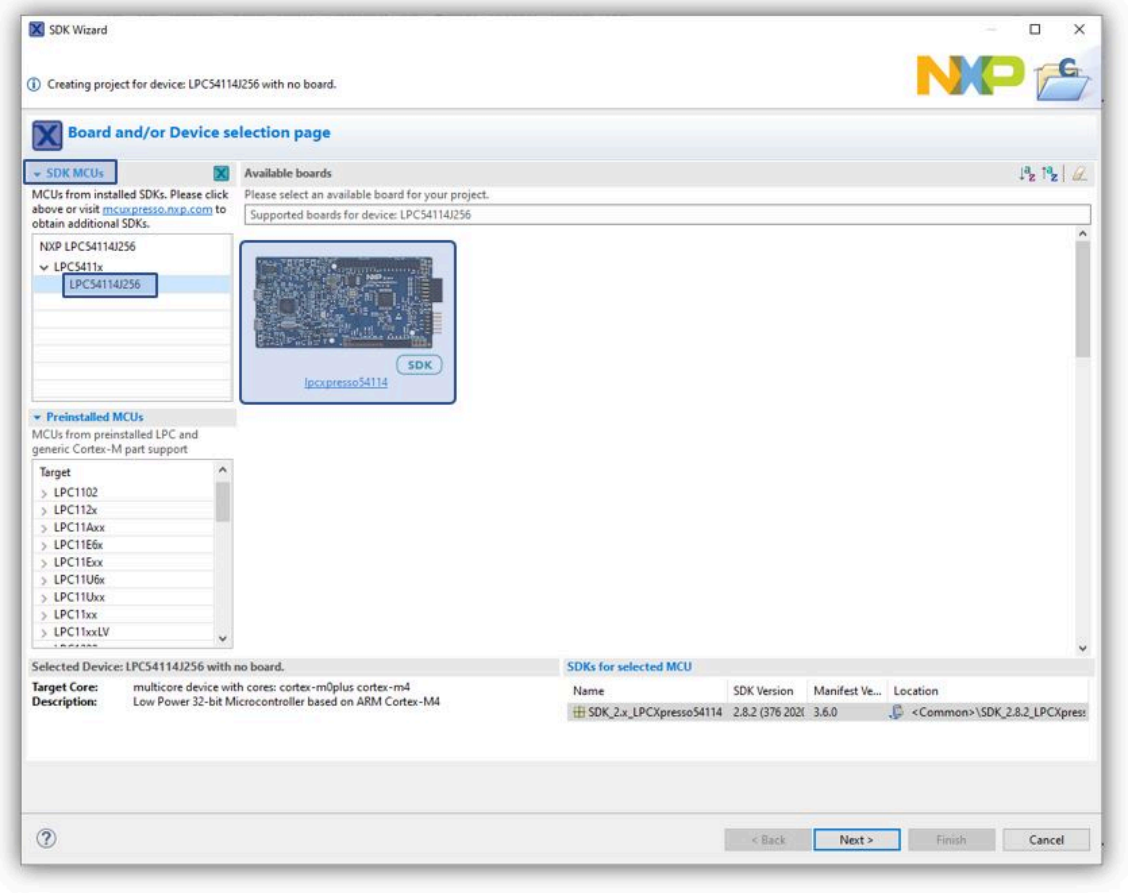


Figure 20.1. New Project Wizard SDK MultiCore M0

From the next wizard page, select the cm0plus Core, and see that the M0SLAVE is selected in the core options. Also note that the Project will automatically be given the suffix M0SLAVE. Drivers, utilities etc. can be selected at this stage for the Secondary project if required.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

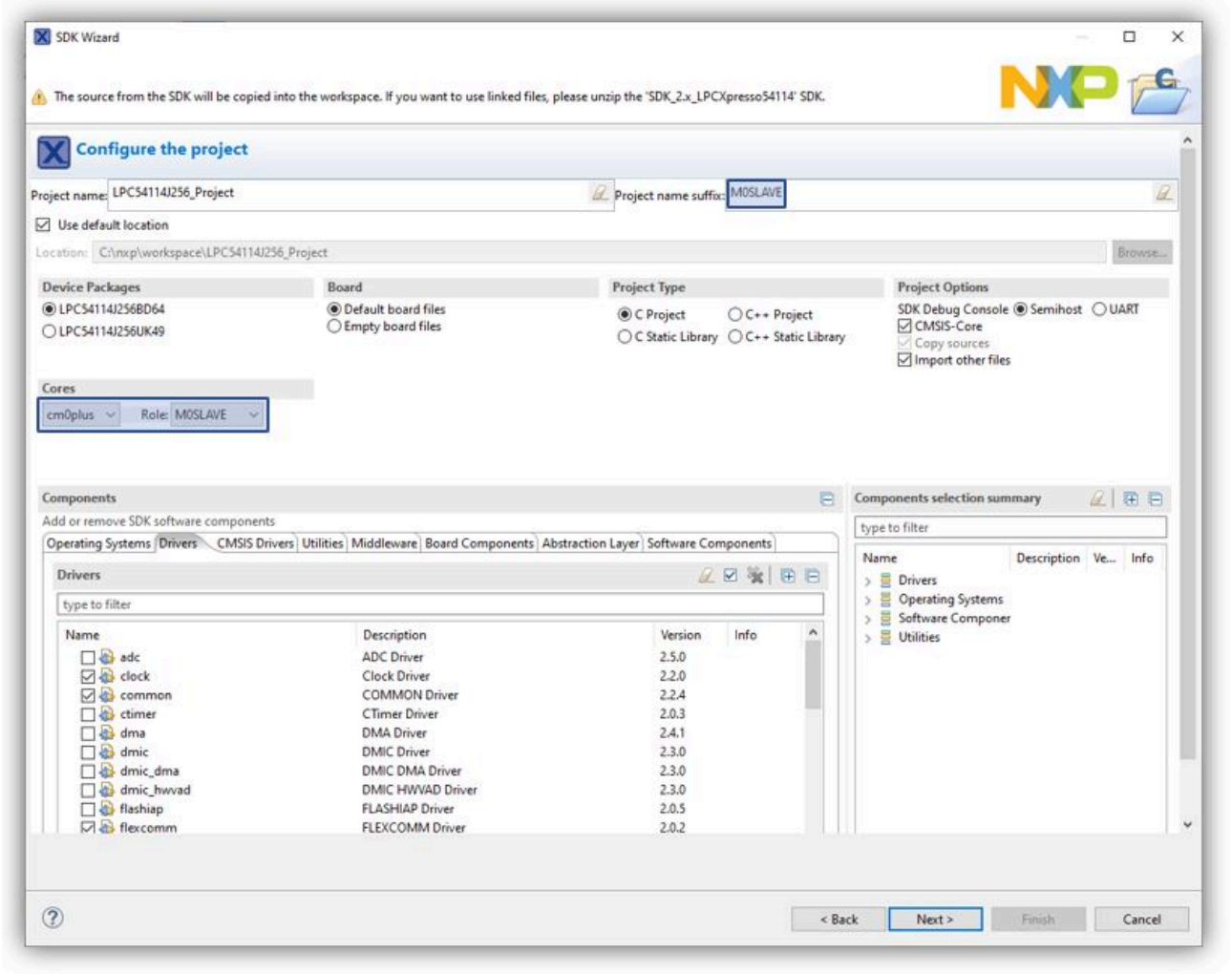


Figure 20.2. New Project Wizard SDK M0 Secondary

Next, the M0 Secondary memory configuration needs to be set.

Note: MCUXpresso IDE's managed linker script mechanism will default to link code to the first Flash region in this view (if one exists) and use the first RAM region for data, heap and stack..

To force our project to link to a private area of RAM, we must ensure that the Flash region is removed and the chosen RAM bank is at the top of the list of memory regions. Note here that the SDK we are using has presented the RAM regions in a non sequential order. In this example we will configure the memory so that the M0 Secondary project links to the RAM region starting at address 0x20010000 (the first region).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

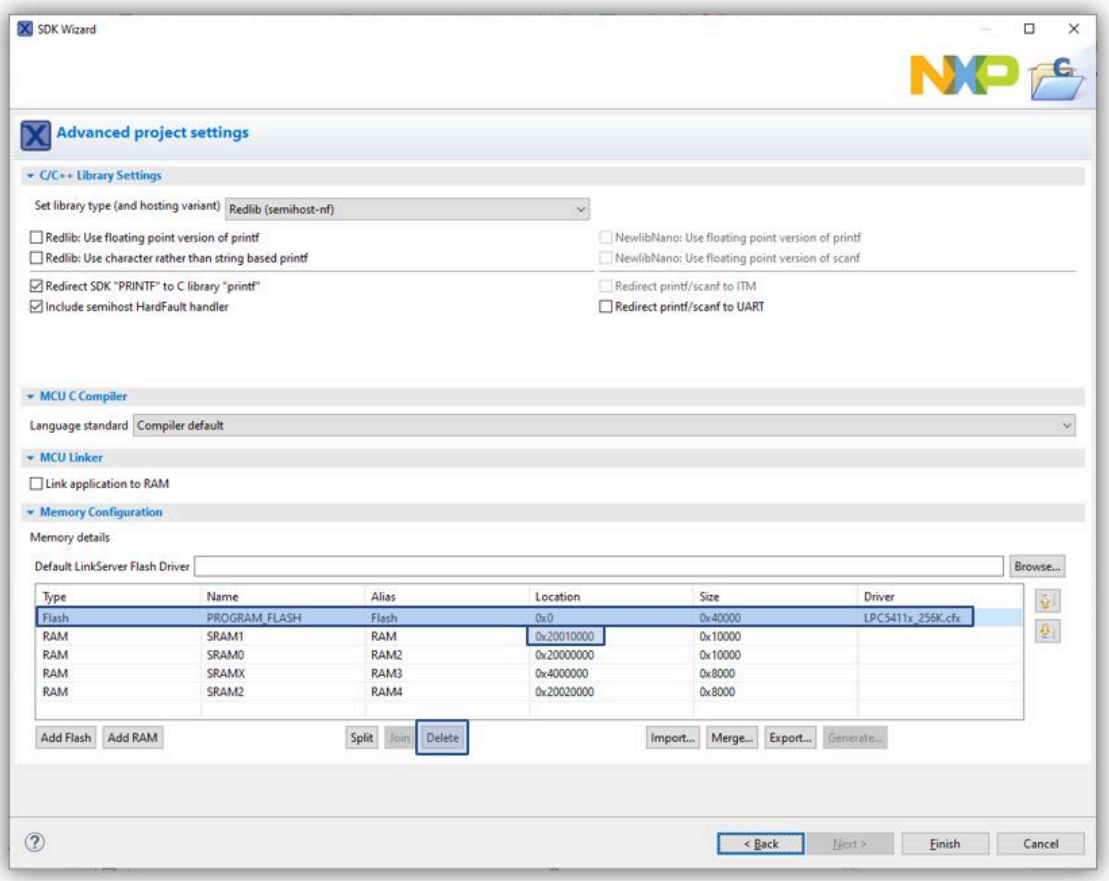


Figure 20.3. New Project Wizard SDK M0 Secondary Memory

From this wizard, select the PROGRAM_FLASH and click Delete to remove the region. Ensure that the top RAM region has the base address (location) 0x20010000, then click Finish to complete the creation of the Secondary project.



Tip

Memory regions can be reordered by selecting a region and using the up/down arrows to move the selected region.

20.2.2 Creating the M4 Primary project

To create the Primary project, launch the New Project Wizard and again select the LPCXpresso54114 SDK board and click Next. This time, select the cm4 Core, and **click the MASTER check box**, this configures the wizard to create a Multicore project. Note that the Project will automatically be given the suffix MASTER.

Drivers, utilities etc. can be selected at this stage for the Primary project if required.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

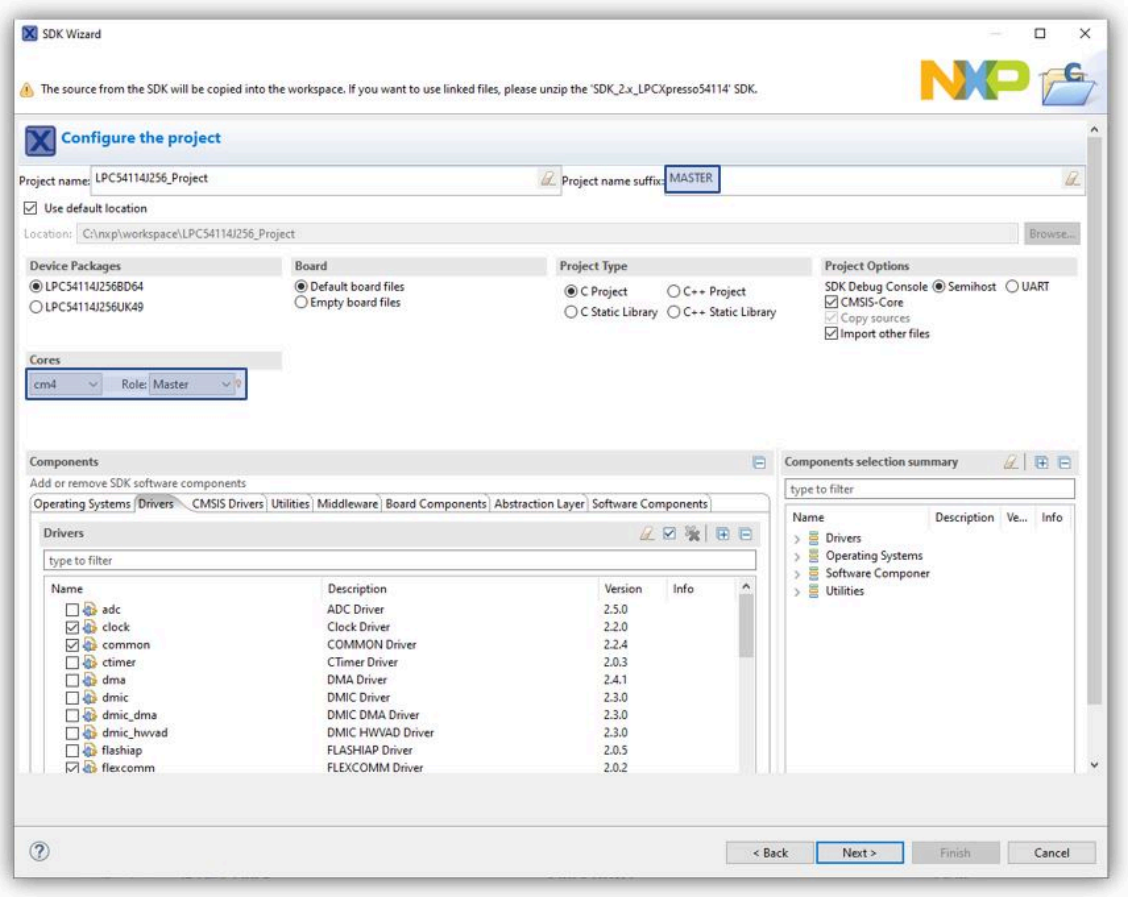


Figure 20.4. New Project Wizard SDK M4 Primary

Next, the M4 Primary project's memory configuration needs to be set. Typically we might leave the memory setting unaltered, however the SDK we are using presents the RAM regions in a non sequential way. In this example we wish to select the RAM region at 0x20000000 for the Primary projects data and the Flash at 0x0 for the Primary projects code (and also a copy of the Secondary projects code)

Note: MCUXpresso IDEs managed linker script mechanism will default to link code to the first Flash region in this view (if one exists) and use the first RAM region for data, heap and stack.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

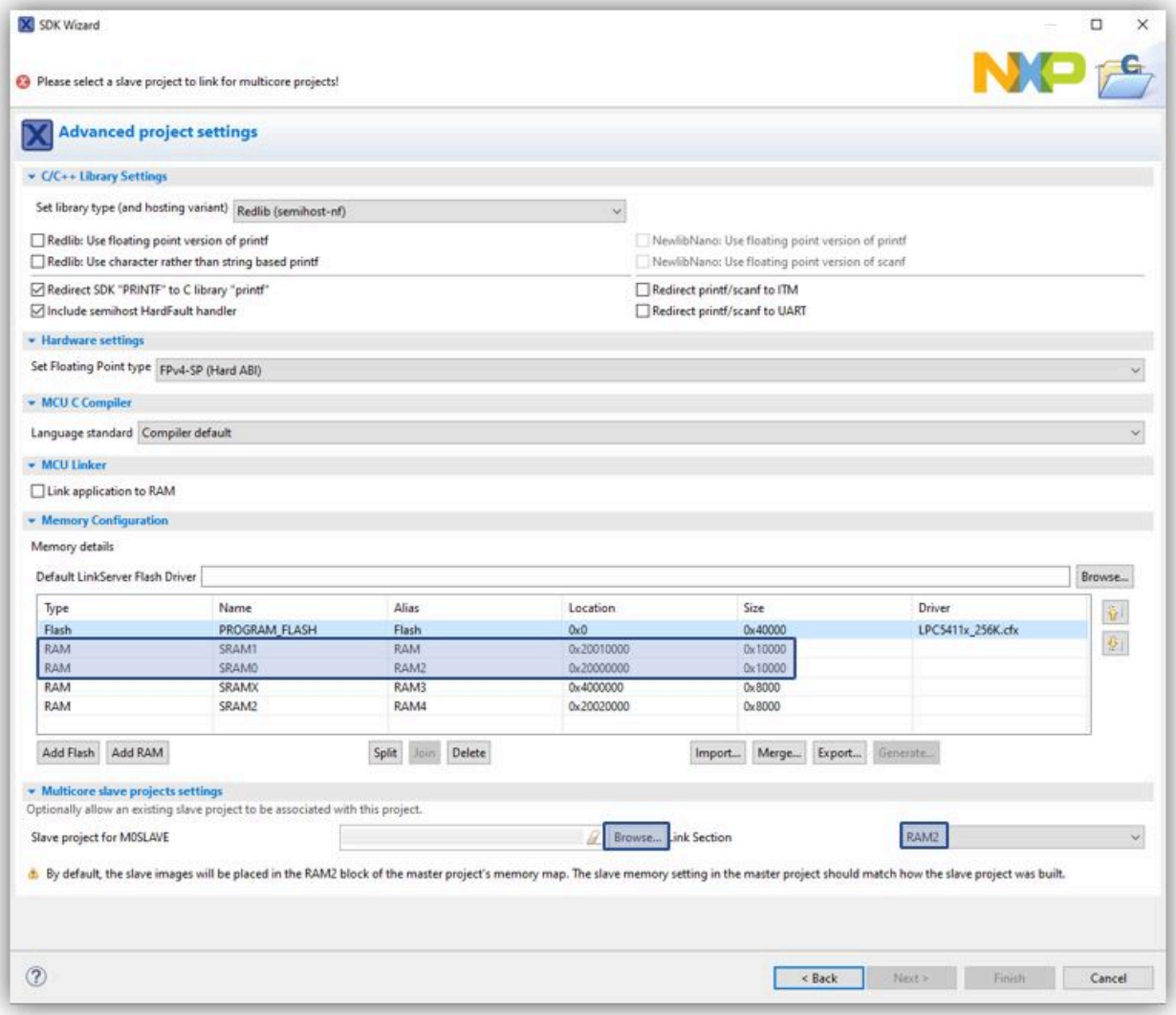


Figure 20.5. New Project Wizard SDK M4 Primary Memory

To adjust the memory layout, select the second RAM region (at location 0x20000000) and click the ‘Up’ arrow to move this to the top of the RAM regions. The highlighted regions as shown above will are effectively swapped.

Once this has been done, click ‘OK’.

Next, click Browse to locate a Secondary project within the Workspace and select the previously created Secondary project, then click ‘OK’.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

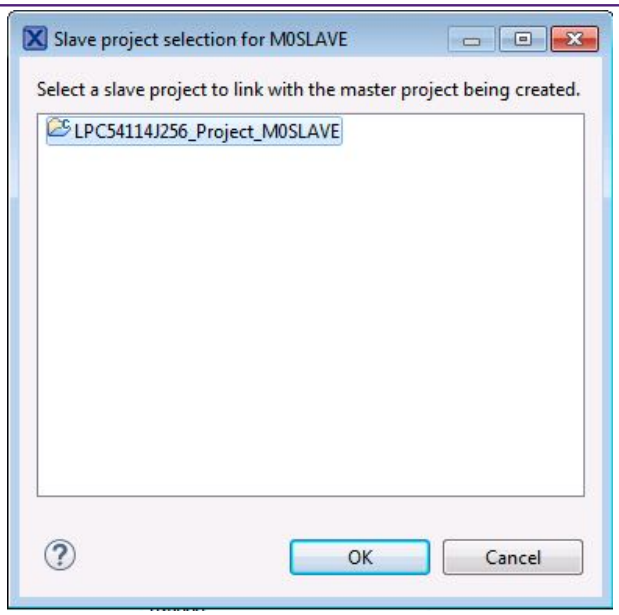


Figure 20.6. New Project Wizard SDK M4 Primary / Secondary Selection

Note: ensure the Link Section name (default of RAM2 highlighted) selects a Primary memory region that matches the linked address of the Secondary project. In this case RAM2 should correspond to the address 0x2001000. If required, other memory regions can be selected here but please note: the first Flash Region and the first RAM Region are not included in the drop down list because it is assumed that these will be used for the Primary Project. If required, this setting can be changed later from:

Project Properties -> C/C++ Build -> Settings -> Multicore

Where all of the memory regions are available for selection.

Below we can see the edited project settings for the Primary project.

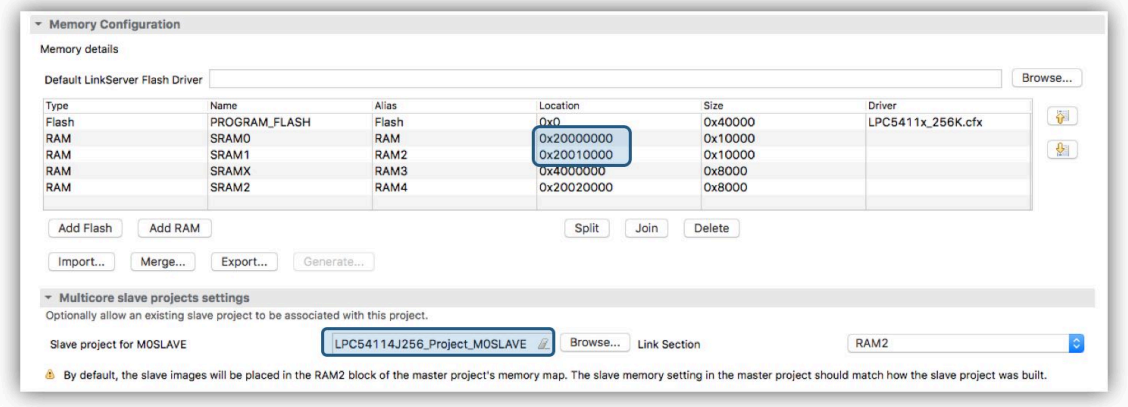


Figure 20.7. New Project Wizard SDK M4 Primary Project

Finally click Finish to generate the Primary project.

Note: if the memory regions of these projects overlap, the linker will generate an error similar to:

```
M0SLAVE execute address differs from address provided in source image
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

To fix this issue, review (and edit) the memory settings of the related projects so that their addresses do not overlap via *Project Properties -> C/C++ Build -> MCU settings* .

20.3 Creating a Primary / Secondary project Pair (using Preinstalled Part Support)

The example described below is base around the LPC5411x multicore MCU.

Note: It is recommended to create and build LPC541xx multicore projects which are linked to LPCOpen. Thus before you follow the below sequence, please ensure that you have imported the chip and (optionally) the board library projects (for both the M4 and M0+) from an LPCOpen package for the LPC5410x family or LPC5411x family (depending upon your target part).

20.3.1 Creating the M0 Secondary project

As discussed above, the Primary projects configuration will need to reference the Secondary project, therefore the Secondary project should be created first.

Launch the New Project Wizard and select the LPC54114-M0 from the Preinstalled MCUs.

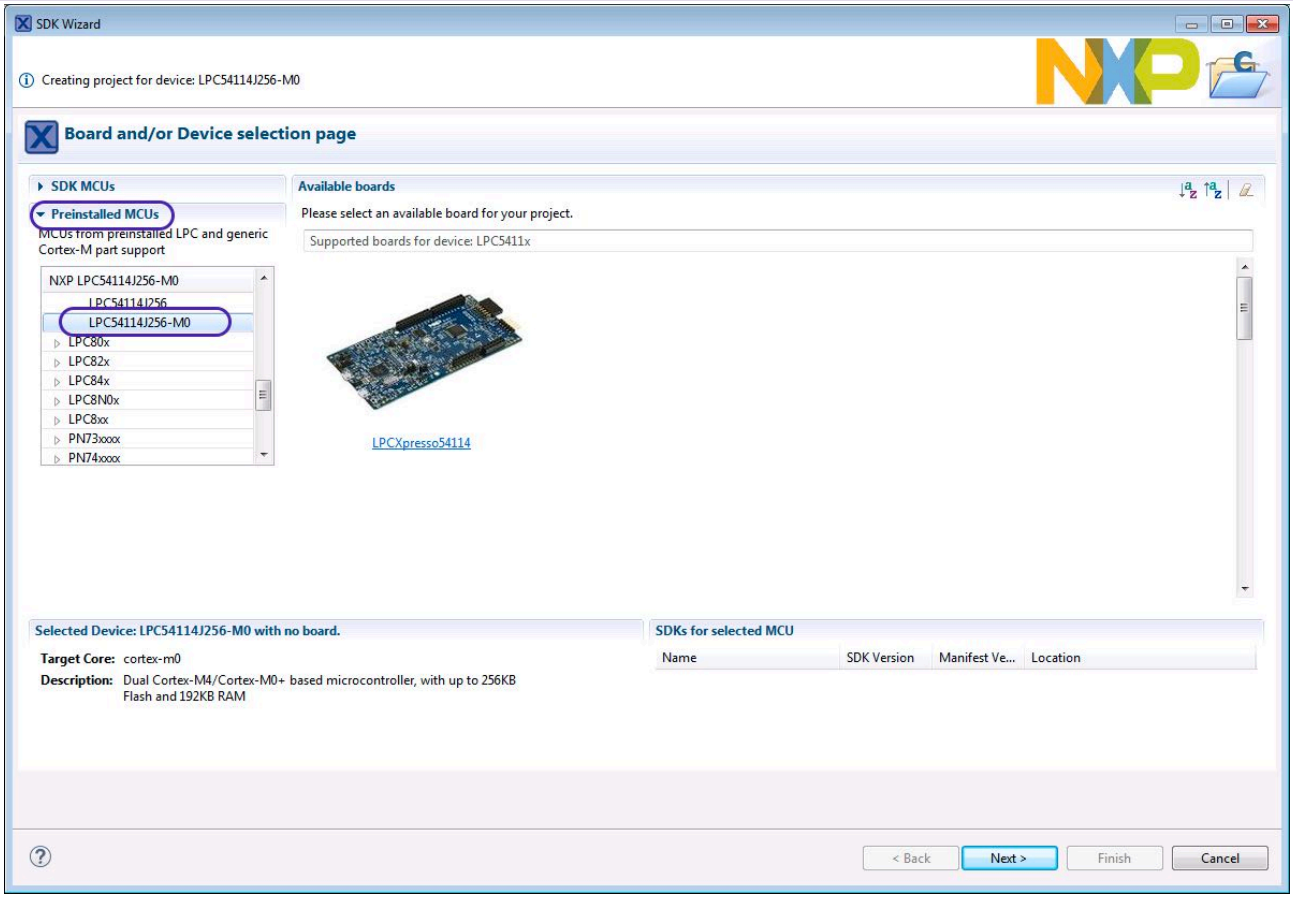


Figure 20.8. New Project Wizard Preinstalled M0

Next, select a MultiCore M0 Secondary project type, below we have selected an LPCOpen – C Project.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

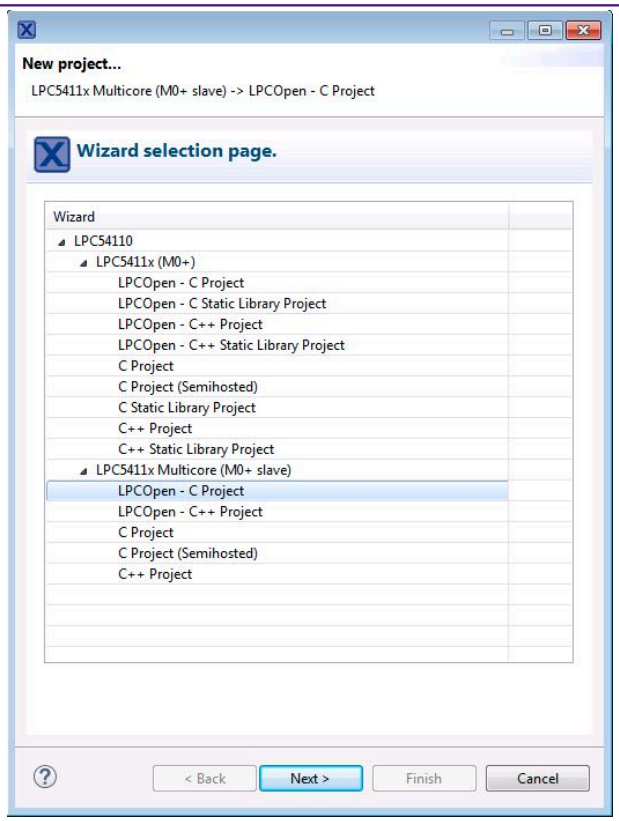


Figure 20.9. New Project Wizard Preinstalled M0 C Project

Next, name the project, for example LPC54114_M0_Slave, then click next until the Memory Configuration page is reached. From here we can see the MCU memory regions.

Note: MCUXpresso IDE's managed linker script mechanism will default to link code to the first Flash region in this view (if one exists) and use the first RAM region for data.

To force our project to link to a private area of RAM, simply delete the Flash and first RAM region (RAM0_64) from this view (since these will be use for the M4 Primary project). To do this, just select the regions and click Delete. Since there will no longer be any Flash region, the default Flash driver can also be removed.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

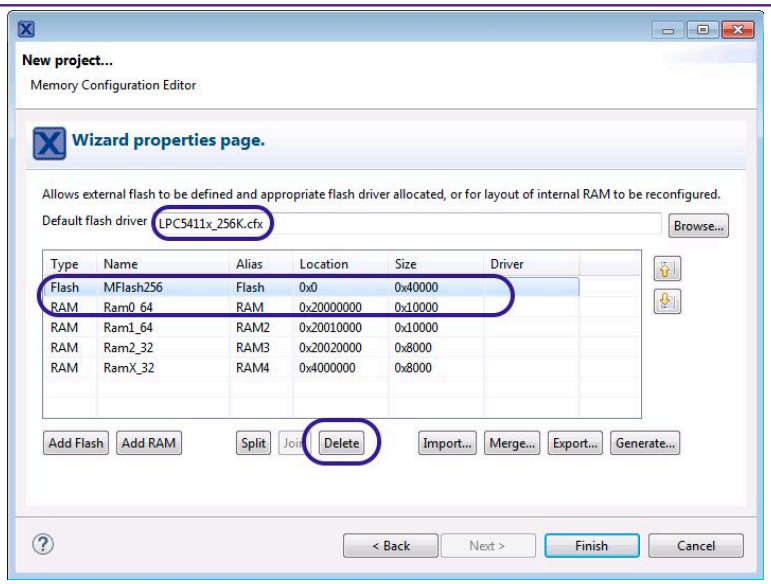


Figure 20.10. New Project Wizard Preinstalled M0 Memory

The memory setting should then look as below. In this case our Secondary projects code and data will be linked to address 0x20010000 with the stack set to the top of this region.

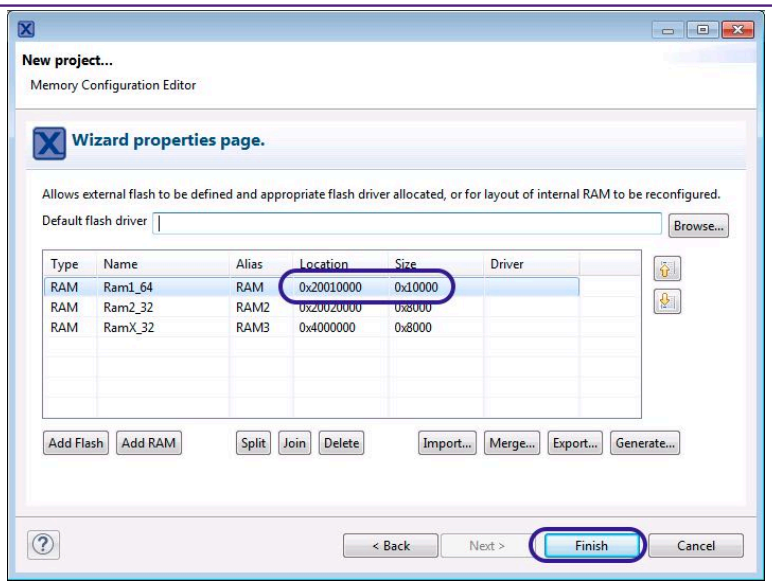


Figure 20.11. New Project Wizard Preinstalled M0 Memory edited

Now click Next -> Finish to complete the M0 Secondary projects creation.

20.3.2 Creating the M4 Primary project

To create the Primary, Launch the New Project Wizard again and this time select the LPC54114 (M4) part and click Next. Select the matching 'MultiCore M4 Master -> LPCOpen -C Project' and click Next again. Now, name the new project, for example LPC54114_M4_Master and click next until the Multicore Project Setup page is reached.

Note: The wizard will present an identical memory configuration page, but on this occasion, no editing is required since the default Flash and RAM setting are

From here, click browse to select the previously created Secondary project from the existing Workspace

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

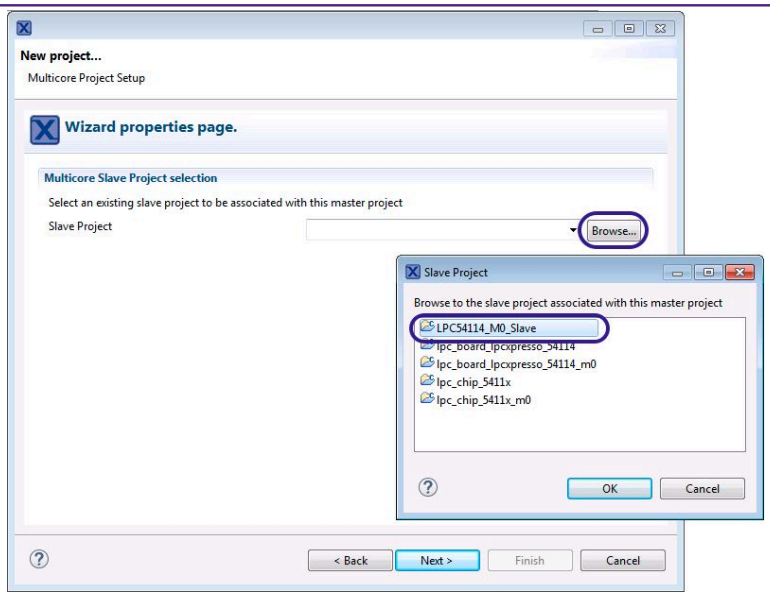


Figure 20.12. New Project Wizard Preinstalled M4 Select Secondary

Now click Next -> Finish to complete the M4 Primary projects creation.

20.4 Debugging MultiCore Projects

The debug story for MultiCore MCUs can vary with their implementation and also the chosen debug solution.

Our MultiCore model as described above, assumes that the Primary project will both copy the Secondary MCUs code and data (into RAM) but also release the Secondary core from reset. Therefore the Primary project should be run (debugged) first and typically run to main(). Once here, the instantiation of the Secondary core's code will be complete but the Secondary core will not have been released. On some MCUs, a debug connection can be made to the Secondary core before it has been released, but on others this will only be possible after they are released.

Note: Secondary projects debug launch configurations may require user modification before a debug connection can be made. Please see the section Secondary Project Debug [256]

In our example LPC54114, the Secondary core's debug connection can be made as soon as the Primare core reaches main(). The debug window will then look similar to that below.

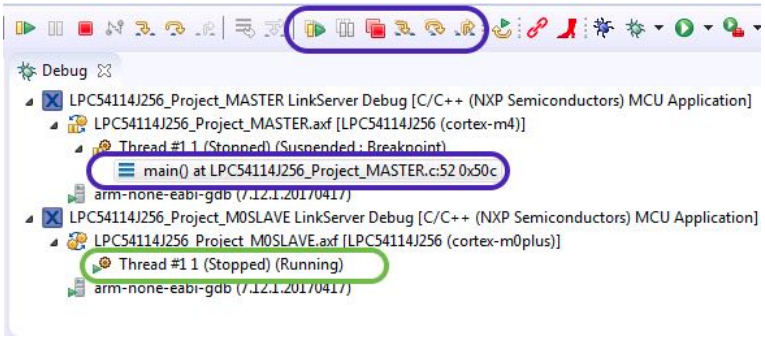


Figure 20.13. MultiCore Debug

Note above: that the MultiCore debug controls have been highlighted, these controls differ from the standard controls in that they operate on all cores being debugged. Via these, the system to be stepped, run, paused, terminated etc.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

In addition, the M4 Primary core's debug stack (blue) is shown stopped at main, while the Secondary core's stack (green) is waiting to be released by the Primary core; clicking between these stacks will change the IDE's debug scope from one core to the other. The currently selected core will be the one used for displaying many of the debug related views, such as Registers and Locals.

20.4.1 Controlling Debug Views

It is also possible to create copies of many of the debug related views, and then lock each copy to a particular core (as described below).

For example, to create two register views, one for the M4 and one for the M0+ ...First of all, use the 'Open New view' button in the Registers view to create a second Registers view:

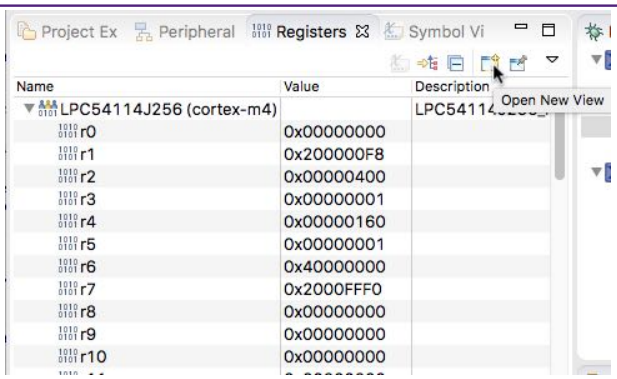


Figure 20.14. MultiCore Debug New View

Now pin the original view to the core currently selected in the Debug, using the 'Pin to Debug context' button :

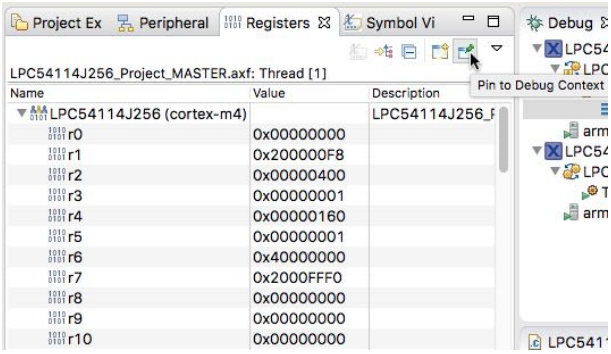


Figure 20.15. MultiCore Debug Pin View

Now select the other core in the Debug view, and go to the second Register view. Use this view's 'Pin to Debug Context' button to lock this second Registers view to the selected core:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

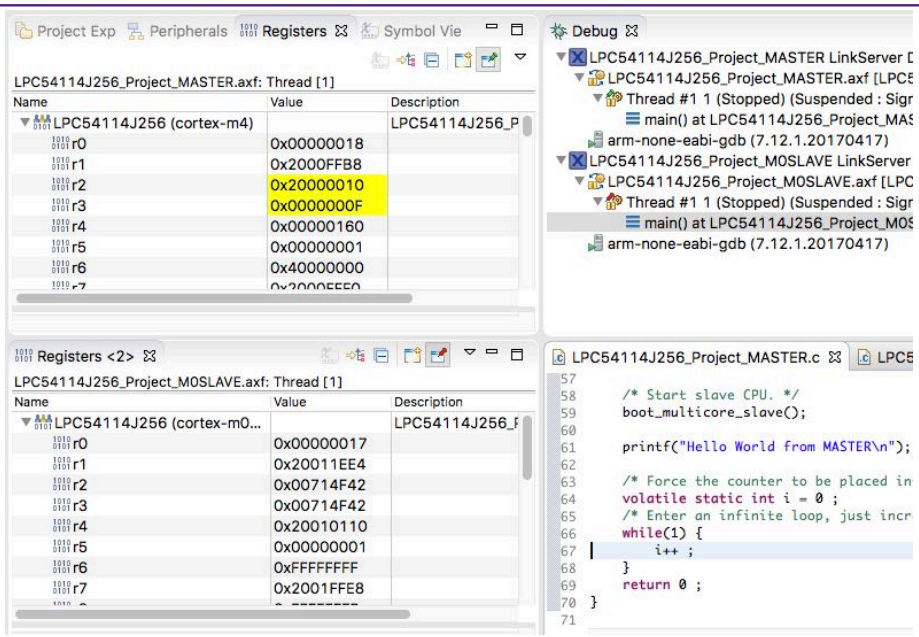


Figure 20.16. MultiCore Debug Registers

20.4.2 Secondary Project Debug

Typically, the Primary project will be debugged first in exactly the same way as a single CPU project. However the secondary projects debug launch configuration may require special settings in order to establish a debug connection to the secondary CPU.

MCUXpresso IDE will automatically configure the correct settings for secondary launch configurations on all debug solutions. However, there might be situations when the Secondary project's debug settings may require modifications.

- **Core Selection** - within a MultiCore MCU there will be more than one CPU (sometimes referred to as a device). The debug connection needs to be made to the appropriate internal CPU for both the Primary and Secondary projects.
  - **LinkServer CMSIS-DAP Debug**: this process is **automatic** and hidden from the user. The selection details are stored within the project's build configuration folder(s) and will take the suffix .jtag or .swd
  - **PEmicro Debug**: appropriate cores inside Primary and Secondary launch configurations are **automatically** selected by the IDE.
  - **SEGGER Debug**: appropriate cores inside Primary and Secondary launch configurations are **automatically** selected by the IDE. However, there might be certain situations when Secondary core should be manually selected. In this case, the IDE will display a warning with regards to incapability of finding a matching core (based on project description).
- **Attach mode** for the Secondary CPU – as described above, the debug connection to the secondary core(s) should be via an attach
  - **LinkServer CMSIS-DAP Debug**: this option is set **automatically** when the LinkServer debug launch configuration is created.
  - **PEmicro Debug**: this option is set **automatically** when the PEmicro debug launch configuration is created.
  - **SEGGER Debug**: this option is set **automatically** when the J-Link debug launch configuration is created.
- **Managing the Debug Server** - this is the low level interface between the debugger and target
  - **LinkServer CMSIS-DAP Debug**: the LinkServer launch configuration is **automatically** correctly configured when the debug connection is made.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- **PEmicro Debug**: the PEmicro launch configuration is **automatically** correctly configured when the debug connection is made.
- **SEGGER Debug**: the J-Link launch configuration is **automatically** correctly configured when the debug connection is made.

20.4.3 Auto-debug secondary project(s) for multicore projects

When using LinkServer as a debug connection, secondary projects can be automatically debugged once initiating debug on the primary project. This behavior is controlled by a LinkServer-specific preference that can be accessed via Eclipse menu -> **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options -> Miscellaneous -> "Enable auto-debug secondary project(s) for multicore projects"**. By default, the preference is enabled in a fresh workspace, meaning that debug sessions for secondary projects will be automatically started.

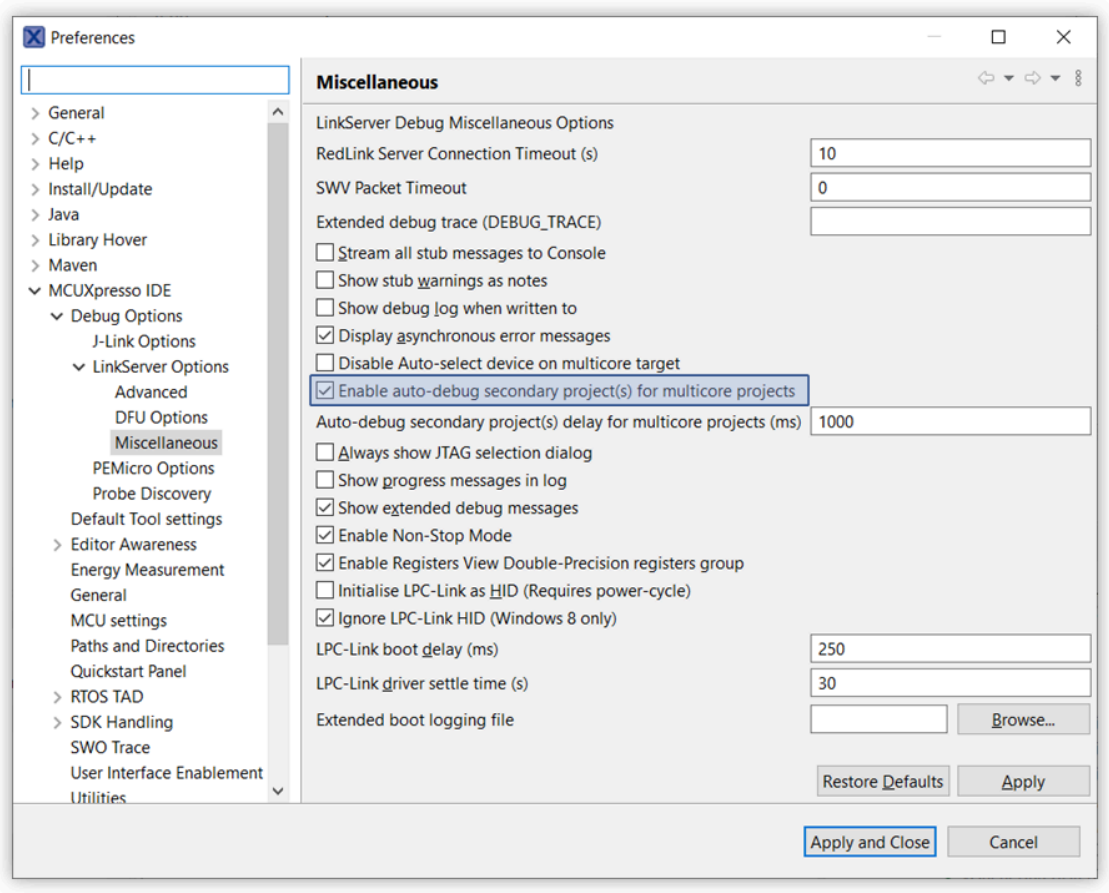


Figure 20.17. Auto-debug secondary project enable option

Subsequent debug sessions started for secondary projects are delayed and the actual delay is specified in another preference, as depicted below. If required, users can change the default 1000ms value.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

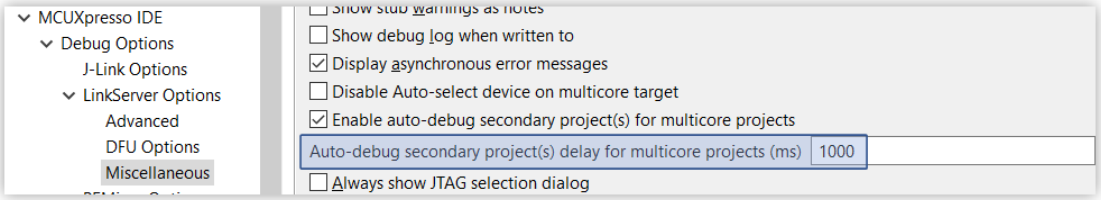


Figure 20.18. Auto-debug secondary project delay option

In the case you don't want to have this feature enabled (so if you want to start debug sessions for each core independently), uncheck this option.

Similar to LinkServer, the option for auto-debug of secondary project(s) for multicore projects becomes enabled by default for multicore debug purpose when using PEmicro and J-Link. For configuring the enablement, go to the appropriate Preferences page:

- **PEmicro**: **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> PEMicro Options -> "Enable auto-debug secondary project(s)"**.
- **J-Link**: **Window -> Preferences -> MCUXpresso IDE -> Debug Options -> J-Link Options -> "Enable auto-debug secondary project(s)"**.

20.5 MultiCore Projects additional Information

20.5.1 Defines

A number of compiler defines are automatically created for LPC5410x projects to allow conditional compilation of certain blocks of code depending upon whether a specific project is configured to be a Secondary, a Primary or neither.

__MULTICORE_MASTER

- Defined automatically for a project that has been configured to be a Primary project

__MULTICORE_MASTER_SLAVE_M0SLAVE

- Defined automatically for a project that has been configured to be a Primary project and has had a Secondary project associated with it (hence indicating to the Primary project which CPU type the Secondary project is for).

__MULTICORE_M0SLAVE

- Appropriate one defined automatically for a project that has been configured to be a Secondary project

__MULTICORE_NONE

- Defined automatically for a project which has not been configured as either a Secondary or Primary project

Note: The multicore support within MCUXpresso IDE is highly flexible and provides functionality beyond that required for the LPC5411x family. Thus the symbols __MULTICORE_MASTER_SLAVE_M4SLAVE and __MULTICORE_M4SLAVE are also provided for completeness.

20.5.2 Secondary Boot Code

boot_multicore_slave() is called by the Primary project code created directly by the New project wizard to release the Secondary core from sleep.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Note: The source files containing this function will be included in all LPC541xx projects, but will be conditionally compiled so that it is included only when required. This has been done to allow projects originally created, for example, as a Secondary project, to be reconfigured (via the project properties – linker multicore tab) as a Primary project.

20.5.3 Reset Handler code

When configured as a Primary project, the LPC541xx startup file will be built with additional (assembler) code at the beginning of the reset handler, ResetISR(), with the ‘standard ‘ reset handler code moved to ResetISR2().

This additional code is required in order to allow correct booting of both the Primary and Secondary cores. It is written in assembler in order to force it to be ‘Thumb1’ code, and hence runnable by both cores.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21. Appendix – Additional Hints and Tips

These additional hints and tips extend the information provided in the main body of this guide.

21.1 Part Support Handling from SDKs

MCUXpresso IDE needs specific device information provided by the SDK in order to properly:

- Create/import projects
  - with part specific startup code
- Define memory layout
- Create debugging launch configuration
- Perform flash programming

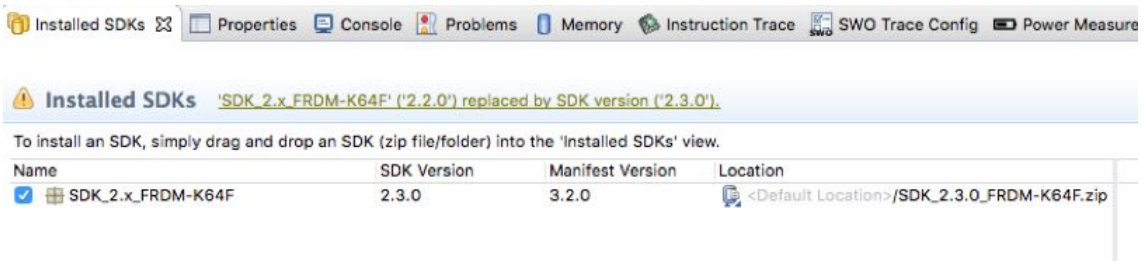
This detailed part knowledge is known as **Part Support**.

21.1.1 SDK Version control

MCUXpresso IDE obtains new Part Support from *installed* SDKs. The IDE's internal database only uses SDKs with the highest version number (latest version is v2.9). For example, a user may have installed two SDKs for a single part:

- SDK_2.3.0_FRDM-K64F
- SDK_2.0.0_FRDM-K64F

The IDE loads only the 2.3.0 version of that SDK, and also provides a warning in the SDK View header:



In this situation, it is likely that the user no longer needs the older version of the SDK. Therefore the IDE provides an option to delete this older SDK via clicking on the warning message, and clicking the 'X'.



Note: Once a new SDK for a part is installed, it will always replace any older installed SDK for that part, even if the new SDK is deactivated (by unchecking the associated tick box). The effect of deactivating an SDK is that part support and wizard will be removed from internal views. These will be restored if the SDK is activated again.

21.1.2 SDK Manifest versioning

Along with SDK versioning, also the internal manifest in an SDK can have multiple versions. MCUXpresso IDE loads the manifest associated to its internal version head info. Thus, assuming an IDE with internal head version set to 3.3, we could have an SDK with the following manifests:

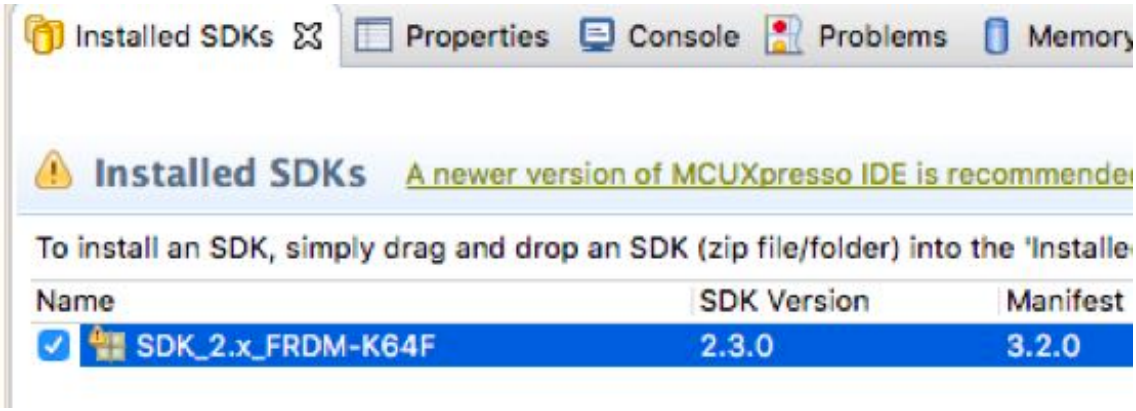
All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Manifest version 3.3
- Manifest version 3.2
- Manifest version 3.1
- Manifest version 3.0

In such case, the IDE will load the manifest version 3.3.

After loading, the IDE validates the manifest against the schema version head, and if for any reason this is not valid, it will try with the other schema versions. If the manifest 3.3 cannot be validated, then it tries with manifest 3.2, validating it, and so on. The manifest version is shown in the SDK View and any validation errors are shown in the Error Log.

In the case that the IDE loads an older manifest, or in the case the SDK contains a manifest 3.4 and the IDE manifest head is 3.3, the SDK image in the SDK view is decorated with a warning and, by clicking on the SDK, a message appears in the SDK view header:



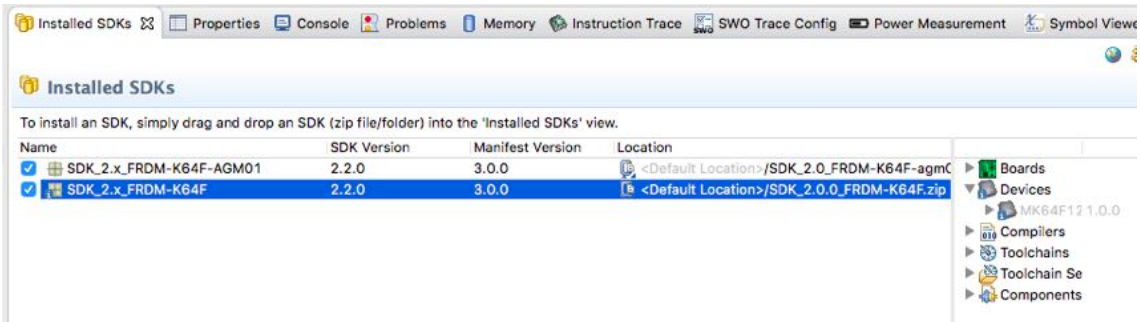
The full error will look like: *"A newer version of the MCUXpresso IDE is recommended for use with the selected SDK. Please update your MCUXpresso IDE in order to get full SDK features"*

Note Even if not intended, newer SDKs may be released supporting features not understood by the current version of the IDE. A message will be used to warn users that there is a mismatch between the SDK and IDE capabilities.

21.1.3 Device versions

If the user installs more than one SDK containing the same device (i.e. a device with the same identifier), the IDE loads the part support from the device with the highest version number, regardless of which SDK it is located within. If two or more SDKs have the same device with the same version number, then the order these are presented to the IDE by the host OS will determine which SDK is used.

If an SDK in the Installed SDK view contains a device that is not installed (because it is supplied by another SDK), its image (and the device in the SDK tree) will be decorated with an icon:



All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

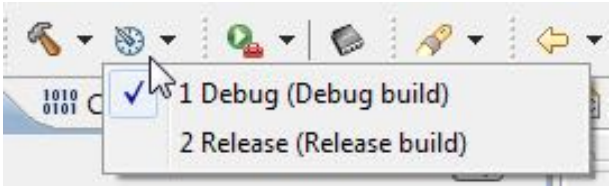
21.2 How do I switch between Debug and Release builds?

By default, MCUXpresso IDE projects will automatically have two build configurations, *Debug* and *Release*. Typically a project will be developed using the Debug build variant, but switched to Release late in the development cycle to benefit from more compilation optimisations.

21.2.1 Changing the build configuration of a single project

You can switch between Debug and Release build configurations by selecting the project you want to change the build configuration for in the Project Explorer view, then using one of the below methods:

- Select the menu item **Project->Build Configuration->Set Active** and select **Release** or **Debug** as necessary
- Use the drop down arrow next to the 'sundial' (Manage configurations for the current project) icon on the main toolbar (next to the 'hammer' icon) and select **Release** or **Debug** as necessary. Alternatively, you can use the drop down next to the 'hammer' icon to change the current configuration and then immediately trigger a build.



- Right click in the Project Explorer view to display the context sensitive menu and select **Build Configurations->Set Active** entry.

21.2.2 Changing the build configuration of multiple projects

It is also possible to set the build configuration of multiple projects at once. This may be necessary if you have a main application project linked with a library project, or you have linked projects for a multicore MCU such as an LPC43xx or LPC541xx (one project for the master Cortex-M4 CPU and another for a slave Cortex-M0/M0+ CPU).

To do this, you first of all you need to select the projects that you wish to change the build configuration for in the Project Explorer view – by clicking to select the first project, then use shift-click or control-click to select additional projects as appropriate. If you want to change all projects, then you can simply use Ctrl-A to select all of them.

Note: it is important that when you select multiple projects, you should ensure that none of the selected projects are opened out – in other words, when you selected the projects, you must not have been able to see any of the files or the directory structure within them. If you do not do this, then some methods for changing the build configuration will not be available.

Once the required projects are selected, you then need to simply change the build configuration as you would do for a single project.

21.3 Editing Hints and Tips

The editor view within Eclipse, which sits under the MCUXpresso IDE, provides a large number of powerful features for editing your source files.

21.3.1 Multiple views onto the same file

The **Window -> Editor** menu provides several ways of looking at the same file in parallel.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- **Clone** : two editor views onto the same file
- **Toggle Split Editor** : splits the view onto the current file into two (either horizontally or vertically)

21.3.2 Viewing two edited files at once

To see more than one file at the same time, simply click the file tabs that you have open in the editor view, and then keep the mouse button held down and drag that file tab across to the right. After you've moved to the side, or below, an outline should appear showing you where that tab will be placed once you release the mouse button.

21.3.3 Source folding

Within the editor view, functions, structures etc. may be folded to show the structure and hide the detail.

Folding is controlled via, right click in the margin of the editor view to bring up the context sensitive menu, then select **Folding -> <option required>**

When folding is enabled, you can then click on the **+** or **-** icon that now appear in the margin next to each function, structure, etc, to expand or collapse it, or use the **Folding -> Expand all** and **Folding -> Collapse all** options from the context sensitive menu

Various settings for Folding can also be controlled through:

Preferences -> C/C++ -> Editor -> Folding

21.3.4 Editor templates and Code completion

Within the editor, a number of related pieces of functionality allow you to enter code quickly and easily.

First of all, templates are fragments of code that can be inserted in a semi-automatic manner to ease the entering of repetitive code – such as blocks of code for C code structures such as for loops, if-then-else statements and so on.

Secondly, the indexing of your source code that is done by default by the tools, allows for auto completion of function and variable names. This is known as "content assist".

- Ctrl-Space at any point will list available editor template, function names etc.
- Ctrl-Shift-Space will display function parameters
- Alt-/ for word completion (press multiple times to cycle through multiple options).

In addition, the predefined templates are user extensible via:

Preferences -> C/C++ -> Editor -> Templates

21.3.5 Brace matching

The editor can highlight corresponding open and closing braces in a couple of ways.

First of all, if you place the cursor immediately to the right of a brace (either an opening or closing brace), then the editor will display a rectangle around the corresponding brace.

Secondly, if you double click immediately to the right of a brace, then the editor will automatically highlight all of the text between this brace and the corresponding one.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.3.6 Syntax coloring

Syntax Coloring specifies how your source code is rendered in the editor view, with different colors used for different elements of your source code. The settings used can be modified in:

Preferences -> C/C++ -> Editor -> Syntax Coloring

Note that general text editor settings such as the background color can be configured in:

Preferences -> General -> Text Editors

Fonts may be configured in:

Preferences -> General -> Appearance -> Colors and Fonts

21.3.7 Comment/uncomment block

The editor offers a number of ways of comment in or out one or more lines of text. These can be accessed using the Source entry of the editor context-sensitive menu, or using the following keyboard shortcuts...

- Select the line(s) to comment, then hit Ctrl-/ to comment out using // at the start of the line, or uncomment if the line is currently commented out.
- Select the line(s) to comment, then hit Ctrl-Shift-/ to block comment out (placing /* at the start and */ at the end).
- To remove a block comment, hit Ctrl-Shift-\.

21.3.8 Format code

The editor can format your code to match the coding standards in use (**Preferences -> C/C ++ -> Code Style**). This can automatically deal with layout elements such as indentation and where braces are placed. This can be carried out on the currently selected text using the Source->Format entry of the editor context-sensitive menu, or using the keyboard shortcuts Ctrl-Shift-F. If no text is selected, then the format will take place on the whole of the current file.

21.3.9 Correct Indentation

As you enter code in the editor, it will attempt to automatically indent your code appropriately, based on the code standards in use, and also the layout of the preceding text. However, in some circumstances, for example after manually laying text out, you may end up with incorrect indentation.

This can usually be corrected using the Source->Correct Indentation entry of the editor context-sensitive menu, or using the keyboard shortcuts Ctrl-I.

Alternatively, use the “Format code” option which will fix other layout issues in addition to indentation.

21.3.10 Insert spaces for tabs in editor

You can configure the IDE so that when editing a file, pressing the TAB key inserts spaces instead of tab characters. To do this go to

Preferences -> General -> Editors -> Text Editors

and tick the “Insert spaces for tabs” box. If you tick “Show white-space characters” you can see whether a tab character or space characters are being inserted when you press the TAB key

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.3.11 Replacing tabs with spaces

To replace existing tabs with spaces throughout the file, open the Code Style preferences:

Preferences -> C/C++ -> Code Style

- Select a Code Style profile and then select Edit...
- Choose the Indentation tab
- For the Tab policy, select Spaces only
- Apply the changes.
  - **Note**: If the Code Style has not been edited before, the Profile must be renamed before the change can be applied.
- The new style will be applied when the source is next formatted using Source -> Format

21.4 Hardware Floating Point Support

Most ARM-based systems – including those based on Cortex-M0, M0+ and M3, have historically not implemented any form of floating point in hardware. This means that any floating point operations contained in your code will be converted into calls to library functions that then implement the required operations in software.

However, many Cortex-M4 based MCUs do incorporate a single precision floating point hardware unit. **Note**: that the optional Cortex-M4 floating point unit implements single precision operations (C/C++ float) only. Thus if your code makes use of double precision floating point (C/C++ double), then any such floating point operations contained in your code will still be converted into calls to library functions that then implement the required operations in software.

Similarly, Cortex-M7 based MCUs may incorporate a single precision or double precision floating point hardware unit.

21.4.1 Floating Point Variants

When a hardware floating point unit is implemented, ARM define that it may be used in one of two modes.

SoftABI

- Single precision floating point operations are implemented in hardware and hence provide a large performance increase over code that uses traditional floating point library calls, but when calls are made between functions any floating point parameters are passed in ARM (integer) registers or on the stack.
- SoftABI is the ‘most compatible’ as it allows code that is not built with hardware floating point usage enabled to be linked with code that is built using software floating point library calls.

HardABI

- Single precision floating point operations are implemented in hardware, and floating point registers are used when passing floating point parameters to functions.

HardABI will provide the highest absolute floating point performance, but is the ‘least compatible’ as it means that all of the code base for a project (including all library code) must be built for HardABI.

21.4.2 Floating point use – Preinstalled MCUs

When targeting preinstalled MCUs, MCUXpresso IDE generally assumes that when Cortex-M4 hardware floating point is being used, then the SoftABI will be used. Thus generally this is the

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

mode that example code (including for example LPCOpen chip and board libraries) are compiled for. This is done as it ensures that components will tend to work out of the box with each other.

When you use a project wizard for a Cortex-M4 where a hardware floating point unit may be implemented, there will be an option to enable the use of the hardware within the wizard's options. This will default to SoftABI – for compatibility reasons.

Selecting this option will make the appropriate changes to the compiler, assembler and linker settings to cause SoftABI code to be generated. It will also typically enable code within the startup code generated by the wizard that will turn the floating point unit on.

You can also select the use of HardABI in the wizards. Again this will cause appropriate tool settings to be used. But if you use this, you must ensure that any library projects used by your application project are also configured to use HardABI. If such projects already exist, then you can manually modify the compiler/assembler/linker settings in Project Properties to select HardABI.

Warning : Creating a project that uses HardABI when linked library projects have not been configured and built with this option will result in link time errors.

21.4.3 Floating point use – SDK installed MCUs

When targeting SDK installed MCUs, MCUXpresso IDE generally assumes that when hardware floating point is available, then the HardABI will be used. This will generally work without problem as generally projects for such MCUs contain all required code (with no use of library projects).

However, it is still possible to switch to using SoftABI using the "Advanced Properties settings" page of the |New project" and "Import SDK examples" wizards.

21.4.4 Modifying floating point configuration for an existing project

If you wish to change the floating point ABI for an existing project (for example to change it from using SoftABI to HardABI), then go to:

Quickstart -> Quick Settings -> Set Floating Point type

and choose the required option.

Alternatively, you can configure the settings manually by going to:

Project -> Properties -> C/C++ Build -> Settings -> Tool Settings

and changing the setting in **ALL** of the following entries:

- MCU C Compiler -> Architecture -> Floating point
- MCU Assembler -> Architecture & Headers -> Floating point
- MCU Linker -> Architecture -> Floating point

Note: For C++ projects, you will also need to modify the setting for the MCU C++ Compiler. **Warning**: Remember to change the setting for all associated projects, otherwise linker errors may result.

21.4.5 Do all Cortex-M4 MCUs provide floating point in hardware?

Not all Cortex-M4 based MCUs implement floating point in hardware, so please check the documentation provided for your specific MCU to confirm.

In particular with some MCU families, some specific MCUs may not provide hardware floating point, even though most of the members of the family do (for example the LPC407x_8x). Thus it

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

is a good idea to double check the documentation, even if the project wizard in the MCUXpresso IDE for the family that you are targeting suggests that hardware floating point is available.

21.4.6 Why do I get a hard fault when my code executes a floating point operation?

If you are getting a hard fault when your application tries to execute a floating point operation, then you are almost certainly not enabling the floating point unit. This is normally done in the LPCOpen or SDK initialisation code, or else in the startup file that MCUXpresso IDE generates. But if there are configuration issues with your project, then you can run into problems.

For more information, please see the Cortex-M4 Technical Reference Manual, available on the ARM website.

21.5 LinkServer Scripts

The LinkServer debug server supports a Basic like programming language that can be used to script low level target operations. Within a LinkServer debug connection, we provide two call outs where scripts can be referenced (if required). The first call out is intended to assist with the initial debug connection, via a Connect Script, and the second is to assist with the targets reset via a Reset Script.

These scripts are specified within a LinkServer launch configuration file and will be preselected if needed for projects performing standard connections to known debug targets.

21.5.1 Supplied Scripts

A set of scripts are supplied within the MCUXpresso IDE installation at:

```
<install dir>/ide/binaries/Scripts
```

These scripts will be used to prepopulate LinkServer launch configuration files when needed.

The purpose of certain scripts will be described below:

- **kinetismasserase.scp** - invoked by the GUI Flash Programmer to Resurrect locked Kinetis device
- **kinetisunlock.scp** - if for any reason the GUI Flash Programmer fails to resurrect a locked part (as above), this script can be specified in place of the above and the recovery attempt repeated
- **delayexample.scp** - an example script showing how a delay can be performed

Note: Some chips also require a preconnect script that prepares the target MCU for the initial debug connection. A set of preconnect scripts can be found within the MCUXpresso IDE installation at:

```
<install dir>/ide/binaries/ToolScripts
```

21.5.2 User Scripts

Additional user generated scripts can be added directly to the product installation but more typically they should be located within a project. The LinkServer launch configuration allows the location of scripts to be either project relative, absolute or product local.

21.5.3 Debugging code from RAM

[This section is deprecated – please see Converting Projects to Run from RAM with LinkServer [271] for details of the improved scheme]

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

MCUs have well defined boot strategies from reset, typically they will first run some internal manufacturer boot ROM code that performs some hardware setup and then control passes to code in flash (i.e. the users Application).

On occasion it can be useful to run and debug code directly from RAM. Since an MCU will not boot from RAM a scheme is needed to take control of the debuggers reset mechanism. This can be achieved the use of a LinkServer reset script.

Within MCUXpresso IDE, certain pre-created scripts are located at:

```
{install dir}/binaries/Scripts
```

Contained in this directory is a script called *kinetisRamReset.scp* (see below).

```
10  REM Kinetis K64F Internal RAM (@ 0x20000000) reset script
20  REM Connect script is passed PC/SP from the vector table in the image by the debugger
30  REM For the simple use case we pass them back to the debugger with the location of \
    the reset context.
40  REM
50  REM Syntax here is that '~' commands a hex output, all integer variables are a% to z%
70  REM Find the probe index
80  p% = probefirstfound
90  REM Set the 'this' probe and core
100 selectprobecore p% 0
110 REM NOTE!! Vector table presumed RAM location is address 0x20000000
120 REM The script passes the SP (%b) and PC (%a) back to the debugger as the reset context.
130 b% = peek32 this 0x20000000
140 a% = peek32 this 0x20000004
145 d% = 0x20000000
150 print "Vector table SP/PC is the reset context."
160 print "PC = "; ~a%
170 print "SP = "; ~b%
180 print "XPSR = "; ~c%
185 print "VTOR = "; ~d%
190 end
```

This reset script makes an assumption that the user intends to run code from RAM at 0x20000000 – this is the value of the SRAM_Upper RAM block on Kinetis parts.

Note: To build a project to link against RAM, you can simply delete any flash entries within the projects memory configuration. If the MCUXpresso IDEs default linker settings are used then project will link to the first RAM block in the list. For many Kinetis parts, this address will match the expected address within the script. For some parts (for example KLxx) however, the first RAM block may take a different value. This problem can be resolved by editing the script or modifying the projects RAM addresses.

For users if LPC parts, the RAM addresses will be different but the principal remains the same. Within the *Scripts* directory, you will find an RAM reset script for the LPC18LPC43 parts, this script is identical to the one above apart from the assumed RAM address.

Finally, to use the script, simply edit the projects launch configuration for the ‘Reset Script’ entry, browse to the appropriate ‘RAMReset.scp’ script.

Note: When executing code from RAM, the projects Vector table will also be located at the start of the RAM block. Cortex M MCUs can locate their vector table using an internal register called *VTOR* (the vector table offset register). Typically this register will be set automatically by a projects startup or init code. However, if execution fails when an interrupt occurs, check that this register is set the correct value.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.5.4 LinkServer Scripting Features

LinkServer scripts are written in a simple version of the BASIC programming language. In this variant of BASIC, 26 variables are available (%a thru %z). On entry to the script some variable have assigned values:

```
    a% is the PC
    b% is the SP
    c% is the XPSR
    d% is the VTOR
```

On exit from the script a% is loaded into the PC, b% is loaded into the SP and d% is loaded into the VTOR, thus providing a way for the script to change the startup behavior of the application.

They offer functionality as shown below:

Generic BASIC like functions that only work inside scripts

```
GOTO 'LineNumber'
IF 'relation' THEN 'statement'
REPEAT : Start of a repeat block
UNTIL 'relation' : End with condition of repeat block
BREAKREPEATTO 'LineNumber' : Premature end of a repeat loop
GOSUB 'LineNumber'
RETURN
TIME : Returns a 10ms incrementing count from the host
```

Generic BASIC like functions

```
PEEK8  {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address>
PEEK16 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address>
PEEK32 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address>
POKE8  {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
POKE16 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
POKE32 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
QPOKE8  {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
QPOKE16 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
QPOKE32 {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> <Data>
QSTARTTRANSFERS {[THIS] | [<ProbeIndex> <CoreIndex>]} <NumReads>
MEMDUMP {[THIS] | [<ProbeIndex> <CoreIndex>]} <Byte Address> <Length>
MEMLOAD {[THIS] | [<ProbeIndex> <CoreIndex>]} <FileName> <Byte Address> <Length
 Limit>  Loads binary file data to memory
MEMSAVE {[THIS] | [<ProbeIndex> <CoreIndex>]} <FileName> <Byte Address> <Length>
 Saves memory to binary file
PRINT "TEXT"[;[~]Variable | Constant]: Print statement. Prints quoted text
 and/or value of an internal variable (a%% - z%%), or constant integer
 expression in decimal, or hexadecimal[~] format
TIME : Returns an incrementing centisecond count from the host
TIMEMS : Returns an incrementing millisecond count from the host
WAIT <msec> : Wait for the number of milliseconds before proceding
LIST: Lists a loaded script
NEW: Erases a loaded script from memory
RENUMBER <Delta>: Renumber script lines with Delta increment (default is 10)
LOAD <"FILENAME">: Loads a script from the current, absolute, or relative directory
SAVE <"FILENAME">: Saves a script to the current, absolute, or relative directory
```

Probe related functions

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
PROBELIST : Enumerates and returns an indexed list of known probe types
PROBENUM : Returns the number of probes attached
PROBEOPENBYINDEX <ProbeIndex> [<"FILENAME">] : Opens the probe associated with ProbeIndex
FILENAME is text of <key = value> pairs used for internal configuration
PROBEOPENBYSERIAL <"SerialNumber"> : Opens the probe associated with SerialNumber
PROBECLOSEBYINDEX <ProbeIndex> : Closes the probe associated with ProbeIndex
PROBECLOSEBYSERIAL <"SerialNumber"> : Closes the probe associated with SerialNumber
PROBEFIRSTFOUND : Returns the THIS ProbeIndex or index of the first probe in the
 enumerated list
PROBETIME <ProbeIndex> : Returns elapsed time from firmware boot, if supported
PROBESTATUS [<ProbeIndex>]: Returns an indexed list summary of the status of the
 probes connected to the system
PROBEVERSION <ProbeIndex>: Returns version information about probe firmware
PROBEISOPEN <ProbeIndex>: Returns TRUE or FALSE
PROBEHASJTAG <ProbeIndex>: Returns TRUE or FALSE
PROBEHASSWD <ProbeIndex>: Returns TRUE or FALSE
PROBEHASSWV <ProbeIndex>: Returns TRUE or FALSE
PROBEHASETM <ProbeIndex>: Returns TRUE or FALSE
```

Core/TAP related functions

```
CORECONFIG {[THIS] | [<ProbeIndex>]}: Queries the scan chain configuration
CORESCONFIGURED <ProbeIndex>: Returns TRUE or FALSE
APLIMIT {[THIS] | [<ProbeIndex>]}:  <APIndex>:  Limit the AP Query (set once)
APLIST  {[THIS] | [<ProbeIndex>]}: [<APLimit>]: Detailed list of APs
 connected to the specified probe. APLimit restricts queries to AP index.
CORELIST {[THIS] | [<ProbeIndex>]}: [<APLimit>]: Detailed list of APs/Cores
 connected to the specified probe. APLimit restricts queries to AP index.
COREREADID {[THIS] | [<ProbeIndex> <CoreIndex>]}: Returns the DpID
DEBUGMAILBOXREQ {[THIS] | [<ProbeIndex> <APIndex>]} <Request>: Debug Mailbox Request
```

Wire related functions

```
WIRESWDCONNECT {[THIS] | [<ProbeIndex>]}: Configures the wire for SWD and
 returns the DpID
WIREJTAGCONNECT {[THIS] | [<ProbeIndex>]}: Configures the wire for JTAG
WIREDISCONNECT {[THIS] | [<ProbeIndex>]}: Closes the wire connection (SWD/JTAG)
WIREISPRESET {[THIS] | [<ProbeIndex>]}: Resets an LPC part into the ISP
 bootloader
WIREBOOTCONFIGSET {[THIS] | [<ProbeIndex>]} <"DATA">: Stores boot configuration data
 that will be automatically applied during subsequent reset commands.
 DATA is a string with up to 4 characters describing how each ISP_CTRL[3..0] pin
 should be handled: '0' (= drive low), '1' (= drive high), 'x' (= do not drive)
WIREBOOTCONFIGGET {[THIS] | [<ProbeIndex>]}: Returns previously stored configuration data
WIREBOOTCONFIGREAD {[THIS] | [<ProbeIndex>]}: Returns the current state of ISP_CTRL[3:0] pins
WIREBOOTCONFIGAPPLY {[THIS] | [<ProbeIndex>]} <1/0>: Immediately starts/stops driving
 the ISP_CTRL pins based on previously stored boot configuration data
WIRETIMEDRESET <ProbeIndex> <ms>: Asserts (Low) reset for ms milliseconds and
 returns the end state of the wire
WIREHOLDRESET <ProbeIndex> <State> : Asserts/Releases (Low/High) reset and
 returns the end state of the wire
WIRESETSPEED <ProbeIndex> <Hz>: Requests a particular wire speed in Hz
WIREGETSPEED <ProbeIndex> : Returns the current wire speed
WIRESETIDLECYCLES <ProbeIndex> <Cycles>: Sets the number of idle cycles between
 debug transactions
WIREGETIDLECYCLES <ProbeIndex> : Returns the current number of debug idle cycles
WIREISCONNECTED <ProbeIndex>: >: Returns TRUE or FALSE if WIRESWDCONNECT or
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

```
WIREJTAGCONNECT is complete
WIREGETPROTOCOL <ProbeIndex>: Returns SWD or JTAG
SELECTPROBECORE <ProbeIndex> <CoreIndex> : Sets the THIS parameter Probe/Core
 pair
THIS : Displays the current Probe, Core pair
```

Cortex-M related functions

```
CMINITAPDP {[THIS] | [<ProbeIndex> <CoreIndex>]}: Initialize a CMx core ready
 for debug connections
CMUNINITAPDP {[THIS] | [<ProbeIndex> <CoreIndex>]}: UnInitialize a CMx core
 (de-assert debug and system power-up)
CMWRITEDP {[THIS] | [<ProbeIndex> <CoreIndex>]} <REG> <DATA>: Returns zero on
 success
CMWRITEAP {[THIS] | [<ProbeIndex> <CoreIndex>]} <REG> <DATA>: Returns zero on
 success
CMREADDP {[THIS] | [<ProbeIndex> <CoreIndex>]} <REG>: Returns data
CMREADAP {[THIS] | [<ProbeIndex> <CoreIndex>]} <REG>: Returns data (handles
RDBUF on AP reads)
CMCLEARERRORS {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMHALT {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMRUN {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMSTEP {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMREGS {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMDEBUGSTATUS {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMWRITEREG {[THIS] | [<ProbeIndex> <CoreIndex>]} <RegNumber> <Value>
CMREADREG {[THIS] | [<ProbeIndex> <CoreIndex>]} <RegNumber>
CMWATCHLIST {[THIS] | [<ProbeIndex> <CoreIndex>]}
CMWATCHSET {[THIS] | [<ProbeIndex> <CoreIndex>]} <DWTIndex> <Address> [<RW|R|W>]
CMWATCHCLEAR {[THIS] | [<ProbeIndex> <CoreIndex>]} <DWTIndex>
CMBREAKLIST {[THIS] | [<ProbeIndex> <CoreIndex>]} : List the FPB breakpoints
CMBREAKSET {[THIS] | [<ProbeIndex> <CoreIndex>]} <Address> : Set an FPB
CMBREAKCLEAR {[THIS] | [<ProbeIndex> <CoreIndex>]} [<Address>] : Clear an FPB
CMSYSRESETREQ {[THIS] | [<ProbeIndex> <CoreIndex>]} : System reset request
CMVECTRESETREQ {[THIS] | [<ProbeIndex> <CoreIndex>]} : Core reset request
CMRESETVECTORCATCHSET {[THIS] | [<ProbeIndex> <CoreIndex>]} : Enable reset
 vector catch
CMRESETVECTORCATCHCLEAR {[THIS] | [<ProbeIndex> <CoreIndex>]} : Disable reset
 vector catch
```

Miscellanious

```
HELP : display help on LinkServer commands
VERSION : returns the LinkServer version
CONNECTIONS : display active connections
```

Scripts can be specified within a LinkServer launch configuration to be run before a connection and/or before a reset.

21.6 RAM projects with LinkServer

MCUs have well defined boot strategies from reset, typically they will first run internal manufacturer boot ROM code to perform some hardware setup and then pass control to code in flash (i.e. the users Application).

Most examples and wizards create projects to run from MCU flash memory but on occasion it can be useful to debug code directly from RAM. There are two stages to such a task:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

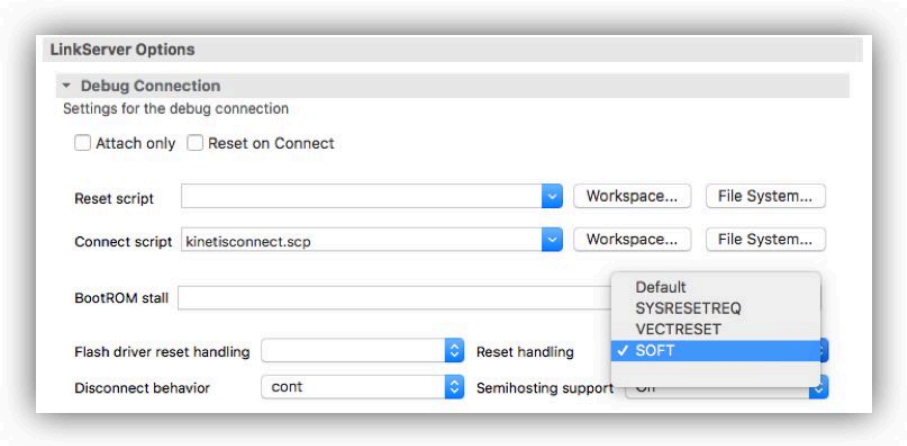
1. Modify a project to that it links to run from RAM
2. Modify the default reset mechanism to ensure that the RAM image is executed

To build a project to link against RAM, simply delete any flash entries within the projects memory configuration. If the MCUXpresso IDEs default linker settings are used then the project will then link against the first RAM block in the list (provided no Flash entry is present). Alternatively, from:

Project Properties -> C/C++ Build -> Settings -> MCU Linker -> Manager Linker Script, you can check the entry *Link application to RAM*.

Note: if the project has already been built to link to flash, then it should be cleaned before being rebuilt.

Since an MCU will not automatically boot from RAM, a scheme is needed to take control of the debuggers reset mechanism. This can be achieved via the use of a **SOFT** reset type. LinkServer launch configurations can take an additional option, add the line *--reset soft* to override the default reset type. Or preferably, set the reset type to *'SOFT'* as shown below.



A soft reset is performed by setting the PC to the images resetISR() address, the stack pointer to the top of the first RAM region and VTOR (Vector Table Offset Register) to the base address of the first RAM region.

Note: Typically, MCU RAM sizes will be smaller that Flash sizes, therefore such a scheme may not be suitable for larger images.

21.6.1 Advantages of developing with RAM projects

There are a number of advantages when debugging from RAM:

- Breakpoints in RAM do not require dedicated HW resources, essentially there is no limit of the number of breakpoints that can be set.
- Flash programming step is not required, so the build and debug cycle will be faster.
- Development of secondary bootloaders is free from BootROM considerations
- No risk of accidently triggering Flash security features.
- No requirement to understand or have flash programming capability allowing code (including flash drivers) can be developed.
- Any flash contents are preserved while debugging
- Unit development of large applications

Note: It should be remembered that since the MCU will not undergo a true hardware reset, peripheral configurations will be inherited from one debug session to the next.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.7 The Console View

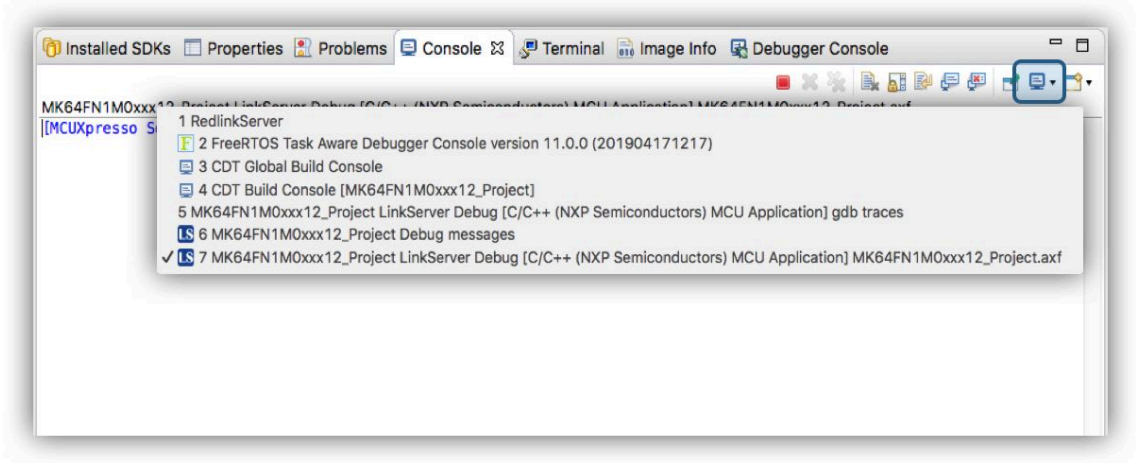
The Console View contains a number of different consoles providing textual information about the operation of various parts of MCUXpresso IDE. It is located by default in the bottom right of the Debug Perspective, in parallel with a number of other views – including the 'Installed SDKs' view.

The actual consoles available within the Console view will depend upon what operations are currently taking place – in particular a number of consoles will only become available once a debug session is started.

The currently displayed console will provide a local toolbar, with icons to do things like copying the contents of the console or clearing its contents.

To see the list of currently available consoles, and, if required, change to a different one..

1. Switch to the Console View
2. Using the toolbar within the Console View click on the drop-down arrow next to the **Display Selected Console** icon (which looks like a small monitor)
3. Select the require console from the drop down list



21.7.1 Console types

Consoles you will typically see include the following...

Build Console and Global Build Console

The Build Console (sometimes referred to as the *Build Log*) is used by the MCUXpresso IDE build tools (compiler,linker, etc.) to display output generated when building your project. In fact MCUXpresso IDE has two build consoles – one of which records the output from building the current project, and the second a global build console which will record the output from building all projects.

By default, the number of lines stored in the Build Console is limited to 500 lines. You can increase this to any reasonable number as follows:

1. Select the **Windows->Preferences** menu option
2. Now choose **C/C++ -> Build -> Console**
3. Increase the **"Limit Console out (number of lines)"** to a larger number, for instance 5000.

Note: This setting, like most within the MCUXpresso IDE is saved as part of your workspace. Thus you will need to make this change each time you create a new workspace.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Other options that can be set in Preferences include whether the console is cleared before a build, whether it should be opened when a build starts, and whether to bring the console to the top when building.

Once your build has completed, then if you have any build errors displayed in the console, clicking on them will, by default, cause the appropriate source file to be opened at the appropriate place for you to fix the error.

FreeRTOS Task Aware Debugger Console

This console displays status about the FreeRTOS TAD views. Enablement and persistence of the logs can be controlled via the Preferences page. For more details, please see the *MCUXpresso IDE FreeRTOS Debug Guide*.

Azure RTOS ThreadX Task Aware Debugger Console

This console displays status about the Azure RTOS ThreadX TAD views. Enablement and persistence of the logs can be controlled via the Preferences page. For more details, please see the *MCUXpresso IDE Azure RTOS ThreadX Debug Guide*.

Zephyr RTOS Task Aware Debugger Console

This console displays status about the Zephyr RTOS TAD views. Enablement and persistence of the logs can be controlled via the Preferences page. For more details, please see the *MCUXpresso IDE Zephyr RTOS Debug Guide*.

gdb traces and arm-none-eabi-gdb Consoles

These consoles give access to the GDB command line debugger, that sits underneath the MCUXpresso IDE's graphical debugging front end.

RedlinkServer/LinkServer Console

This console gives access to the server application that sits at the bottom of the debug stack when using a debug probe connected via the MCUXpresso IDEs native "LinkServer" debugging mechanism. LinkServer commands can be entered from this console.

Debug messages Console

The Debug Messages Console (sometimes referred to as the *Debug Log*) is used by the debug driver to display additional information that may be helpful in understanding connection issues when debugging your target MCU.

Semihosting Console

This console, generally displayed with *.axf*, allows semihosted output from the application running on the MCU target to be displayed, and potentially for input to be sent down to the target.

SWO and Trace Console

This console displays all information related to CoreSight components creation and configuration, starting with the base address of the ROM table that is being scanned for the purpose of CoreSight identification. For more information please refer to *MCUXpresso_IDE_SWO_Trace.pdf* documentation.

21.7.2 Copying the contents of a console

Occasionally, you may wish to copy out the contents of a console. For instance, the MCUXpresso IDE support team may ask you to provide the details of your Build Console in a forum thread. To do this:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

1. Clean, then build your project.
2. Select the appropriate Build Console as above:
3. Select the contents (e.g. Ctrl-A)
4. Copy to the clipboard (e.g. Ctrl-C).
5. Paste from clipboard into forum thread (e.g. Ctrl-V). If there is a large amount of text in the build console, it is advisable to paste it into a text file, which can be ZIPed if appropriate.

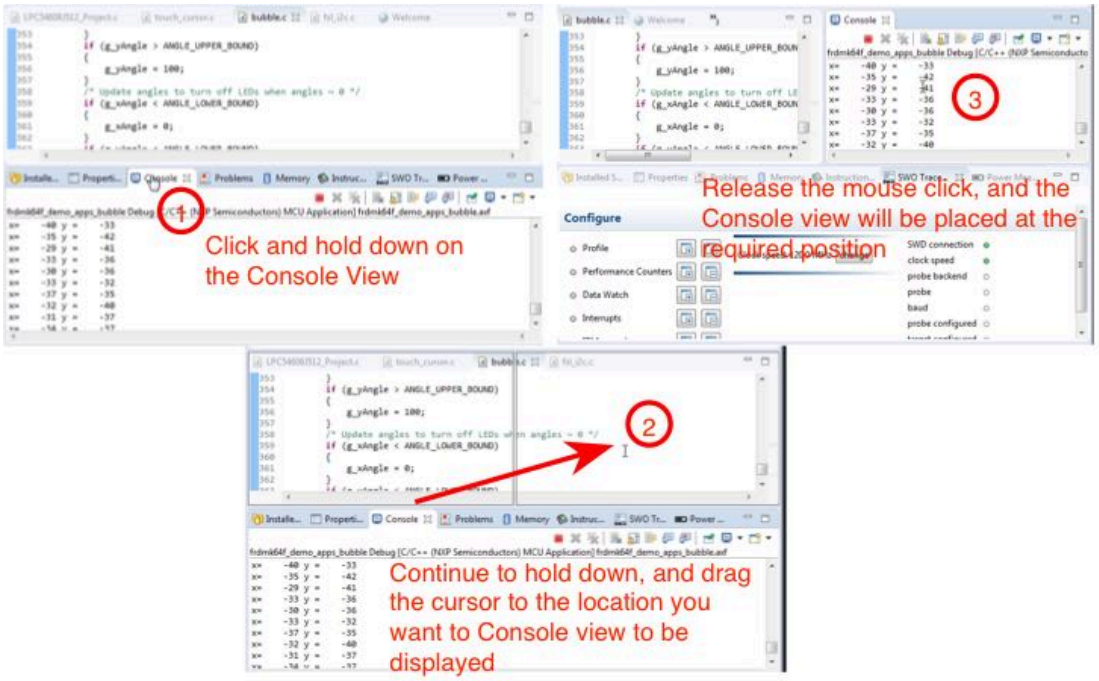
Note: some console will provide a button in their local toolbar to copy or save out their contents.

21.7.3 Relocating and duplicating the Console view

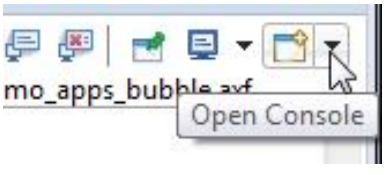
By default the Console View is positioned in parallel with a number of other views. This can mean, if a console is being regularly updated with new output (for instance the view displaying semihosted output from the application running on the target MCU), then by default this may cause the console to keep jumping to the foreground – hence hiding other views that you are using (for instance one of the SWO Trace views)

To avoid this you may wish to relocate the Console. To do this:

1. Click and hold down on the Console View
2. Continue to hold down, and drag the cursor to the location you want to Console view to be displayed
3. Then release the mouse click, and the Console view will be placed at the required position

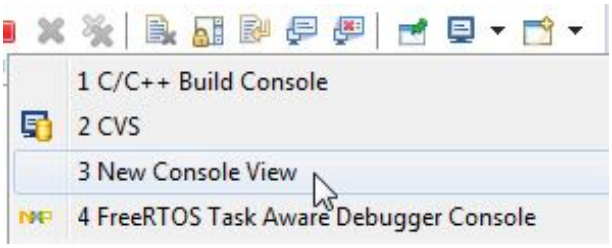


Another alternative is to spawn a duplicate instance of the Console view. This allows multiple consoles to be visible at the same time. To do this use the Open Console button on the Console view local toolbar

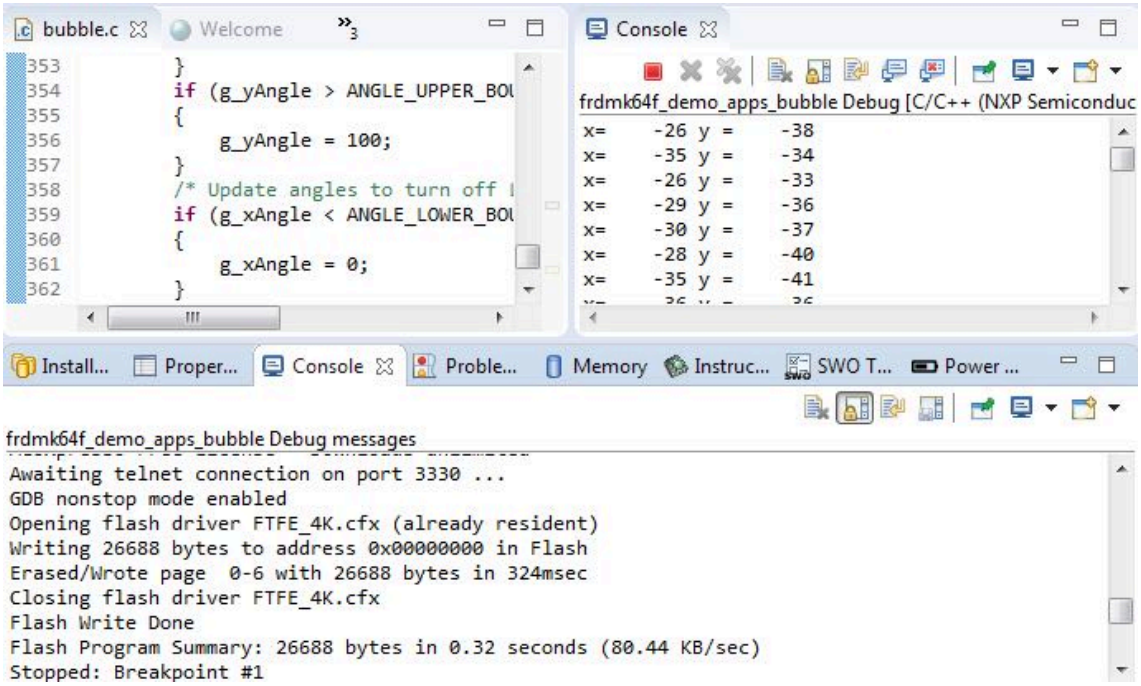


and then select "New Console View"

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



This will then display a second console view, which can be drag and dropped to a new location within in the Perspective, as shown for the single Console view case described above.



Having opened a second console view, select which console you want displayed in it, and then use the “Pin Console” button to ensure that it does not switch to one of the other consoles when output is displayed.



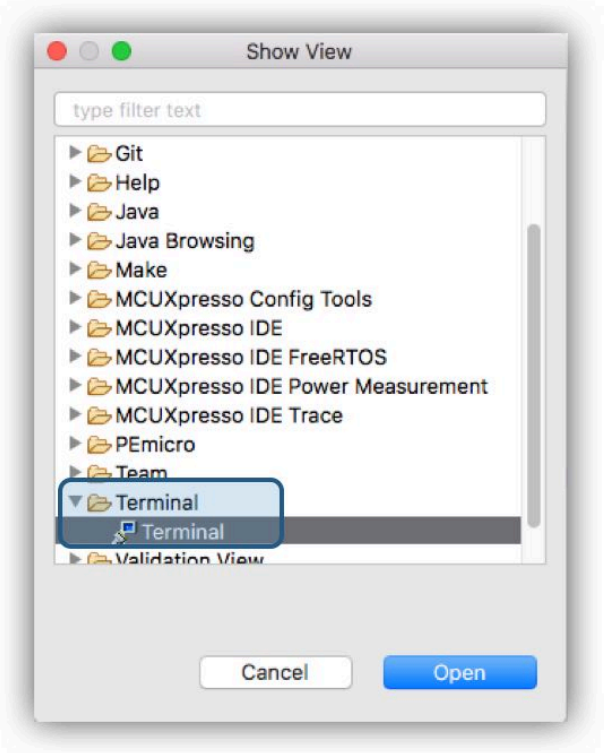
21.8 Using Terminal View for UART communication with target

MCUXpresso IDE provides a Terminal View, which can be used to display UART (serial) input/output between a host PC and the target MCU. In situations where a debug probe is built into the target board, UART comms will often be possible via a VCOM connection over the same USB cable as the debug connection. However, where this is not the case a serial_to_USB cable can be used, alternatively, if the target MCU has built in USB then a VCOM port can implemented in the application code running on the target MCU.

Using a Terminal View offers an alternative way of interacting with the target when compared to semihosting output via debug channel (which is displayed in the Console View). There are pros and cons to both approaches, but one distinct advantage to using the Terminal View for serial output is that you can interact with the target MCU without a debug session being active!

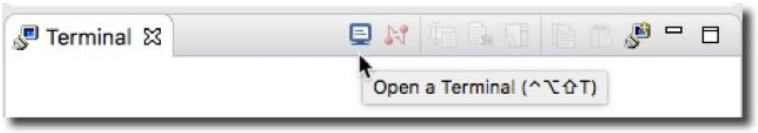
All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

To use the Terminal View within MCUXpresso IDE, the first thing you will need to do is open it (as it is not visible by default). To do this go to: *Window -> Show View -> Other* and select Terminal.



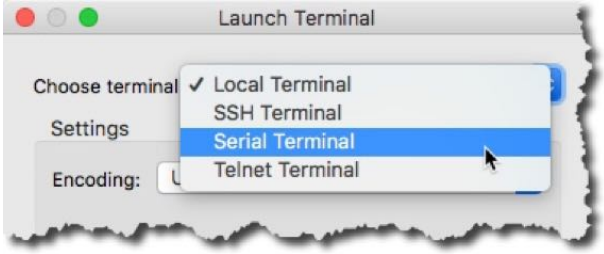
Alternatively, just type “Terminal” into the “Quick Access” button in the top right of the IDE’s window.

Next, ensuring that the serial connection between your PC and the target MCU is active first, click on the “Open a Terminal” button in the Terminal View’s toolbar:



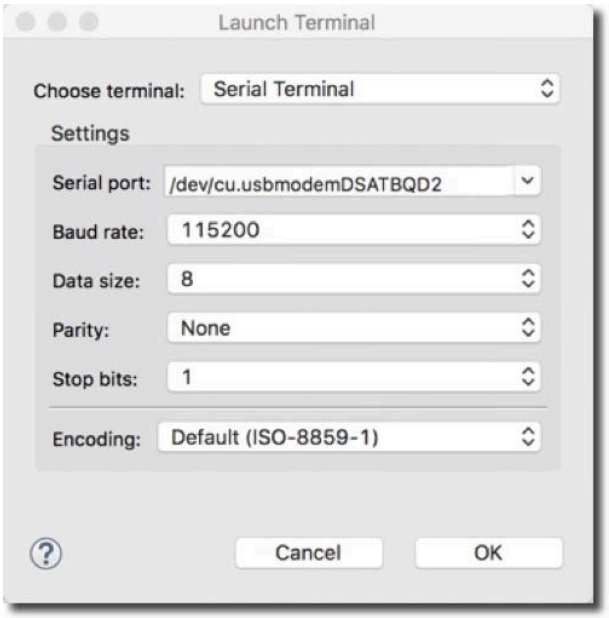
Note: If using the LPC-Link2 built into many LPCXpresso boards, then you need to make sure the probe has been booted before the serial connection will be available. You can do this manually by using the “Boot Debug probe” button in the tool bar towards the top of the IDE window. Or else you can pre-program the probe firmware into flash using LPCScrypt.

Now select the type of terminal required – a serial one :



And then select the appropriate settings:

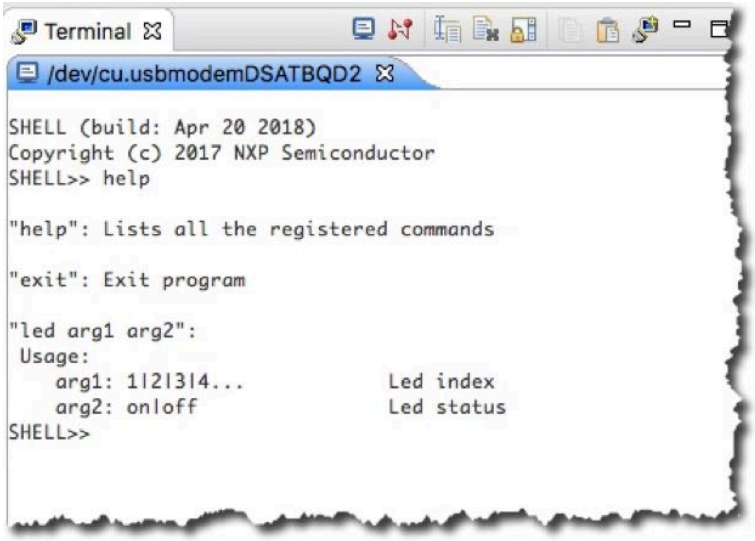
All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Note: that if you are receiving serial output via USB (for instance over a VCOM port from the debug probe), then the default settings should normally be fine. The one setting you do need to get correct is the Serial port to use. This will vary depending upon what devices are connected within your PC, what OS you are running, and what the source for your serial port will be.

For instance if you are running on Windows, then the simplest way to identify the required serial port is to open “Device Manager” (typically via the “Start Menu”), then expand the “Ports” tab. This should allow you to identify the appropriate COM port needed.

After configuring the settings as required, click on the “OK” button. You should now see serial output from the application running on the target MCU within the Terminal View:



Note: the Terminal view only offers a simple terminal mechanism with a small number of configuration options. If you require more control over the way the terminal behaves, you may still need to use a standalone terminal application, such as PuTTY, CoolTerm and Tera Term.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.9 Using and troubleshooting LPC-Link2

21.9.1 LPC-Link2 hardware

LPC-Link2 is a powerful, low cost debug probe design from NXP Semiconductors based on the LPC43xx MCU. It has been implemented into a number of different systems, including:

- The standalone LPC-Link2 debug probe
- The debug probe built into the range of LPCXpresso V2/V3 boards.

For more details, see https://www.nxp.com/lpcxpresso-boards

21.9.2 Softloaded vs Pre-programmed probe firmware

One thing that most LPC-Link2 implementation offer is the ability to either softload the debug probe firmware (using USB DFU functionality) or to have the debug probe firmware pre-programmed into flash.

Programming the firmware into flash has some advantages, including:

- Allows the use of the LPC-Link2 with toolchains that, unlike MCUXpresso IDE, do not support softloading of the probe firmware.
- Better supports the use of LPC-Link2 as a small production run programmer
- Allows the LPC-Link2 to be used with SEGGER J-Link firmware as an alternative to the normal CMSIS-DAP firmware. For more details please visit https://www.segger.com
- Avoids issues that the re-enumeration of the LPC-Link2 can sometimes trigger as the firmware softloads (particularly where virtual machines are in use).

The recommended way to program the firmware into the flash of LPC-Link2 is NXP's LPCScrypt flash programming tool. For more details, see https://www.nxp.com/LPCSCRYPT

However, when used with MCUXpresso IDE, softloading the probe firmware is the recommended method of using LPC-Link2 in most circumstances.

This ensures that the firmware version matching the MCUXpresso IDE version can automatically be loaded when the first debug session is started (so normally the latest version). It also allows different probe firmware variants to be softloaded, depending on current user requirements.

For this to work, you need to make sure that the probe hardware is configured to allow DFU booting. To do this:

- For standalone LPC-Link2: remove the link from header JP1 (nearest USB)
- For LPCXpresso V2/V3: add a link to the header "DFU link"

21.9.3 LPC-Link2 firmware variants

As well as providing debug probe functionality, NXP's CMSIS-DAP firmware for LPC-Link2 by default also includes bridge channels to provide:

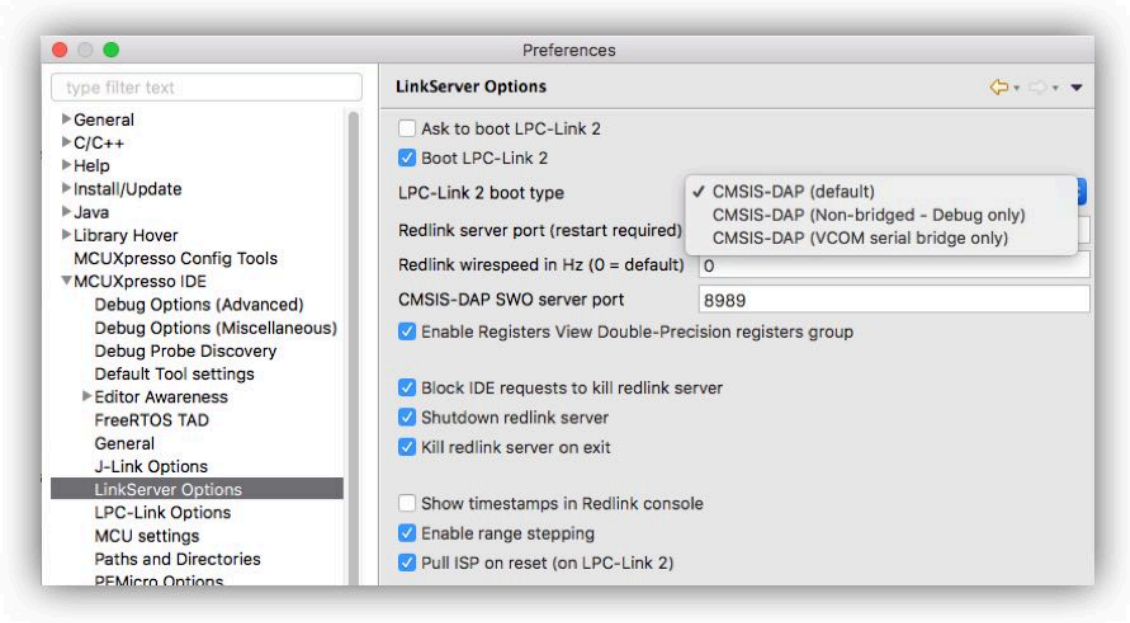
- Support for SWO Trace capture from the MCUXpresso IDE
- Support for Power Measurement from the MCUXpresso IDE (certain LPCXpresso V3 boards only)
- Support for a UART VCOM port connected to the target processor (LPCXpresso V2/V3 boards only)
- Support for a LPCSIO bridge that provides communication to I2C and SPI slave devices (LPCXpresso V3 boards only)

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

However, two other variants of the CMSIS-DAP firmware are provided that remove some of these bridge channels.

- **"Non Bridged":** This version of firmware provides debug features only – removing the bridged channels such as trace, power measurement and VCOM. By removing the requirement for these channels, USB bandwidth is reduced, therefore this firmware may be preferable if multiple debug probes are to be used concurrently. The non-bridged build will also provide an increase in download and general debug performance.
- **"VCOM Only":** This version of firmware provides only debug and VCOM features. The removal of the other bridges allows better VCOM performance (though generally the bridged firmware provides more than good enough VCOM performance).

A particular workspace can be switched to softload a different firmware variant via: **Preferences -> MCUXpresso IDE -> Debug Options -> LinkServer Options -> LPC-Link2 boot type**.



Note: If a mix of bridged and unbridged debug probes is required, then it is recommended that these probes are pre-programmed with the required debug firmware. This can easily be done via LPCScrypt.

21.9.4 Manually booting LPC-Link2

The recommended way to use LPC-Link2 with the MCUXpresso IDE is to allow the GUI to boot and softload a debug firmware image at the start of a debug session.

Normally, LPC-Link2 is booted automatically (when configured to operate in DFU mode) however, under certain circumstances – such as when troubleshooting issues, or using the LinkServer command line flash utility, you may need to boot it manually.

LPC-Link2 USB Details

The standard utilities to explore USB devices on MCUXpresso IDE supported host platforms are:

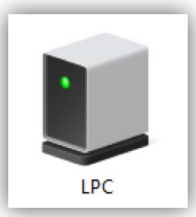
- Windows – Device Manager
  - MCUXpressoIDE also provides a listusb utility in:
    - *install_dir/ide/binaries/Scripts*
- Linux – terminal command: lsusb
- Mac OS X – terminal command: system_profiler SPUSBDataType

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

Before boot, LPC-Link2 appears as a USB device with details:

```
Device VendorID/ProductID:   0x1FC9/0x000C    (NXP Semiconductors)
```

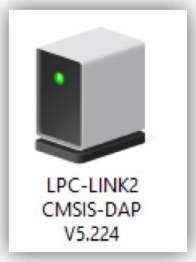
and will appear in Windows -> Devices and Printers, as below:



After boot, LPC-Link2 will by default appear as a USB device with details:

```
Device VendorID/ProductID:   0x1FC9/0x0090
```

and will appear in Windows -> Devices and Printers similar to below:



Note: Text details will vary depending on version number and which probe firmware variant is booted.

Booting from the command line

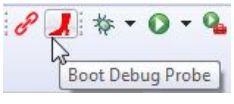
MCUXpresso IDE provides a boot script for all supported platforms. To make use of this script first of all connect the LPC-Link2 to your PC then enter the commands into a DOS command prompt (or equivalent):

```
cd <install_dir>\ide\binaries
boot_link2
```

This will invoke the dfu-util utility to download the probe firmware into the RAM of the LPC-Link2's LPC43xx MCU and then re-enumerate the probe.

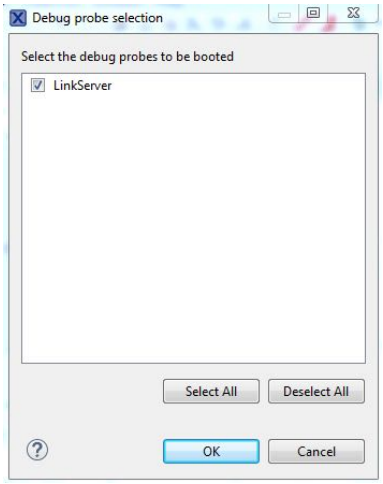
Booting from the GUI

It is also possible to manually boot LPC-Link2 from the MCUXpresso IDE GUI, which may be a more convenient solution than using the command line. To do this, first of all connect the LPC-Link2 to your PC, then locate the red Boot icon on the Toolbar:



and then click OK in the dialog displayed :

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



21.9.5 LPC-Link2 windows drivers

The drivers for LPC-Link2 are installed as part of the main MCUXpresso IDE installation process.

Note: One thing to be aware of is that the first time you debug using a particular LPC-Link2 on a particular PC, the drivers will need to be loaded. This first time can take a variable period of time depending upon your PC and operating system version. This may mean that the first debug attempt fails, as the IDE may time out waiting for the booted LPC-Link2 to appear. In such as case, a second debug attempt should complete successfully. Otherwise, try booting the LPC-Link2 manually and checking the drivers load correctly.

If you need to reinstall the drivers, then the installer can be found at:

```
C:\nxp\<install_dir>\Drivers\lpc_driver_installer.exe
```

21.9.6 LPC-Link2 failing to enumerate

On some systems, after booting LPC-Link2 with CMSIS-DAP firmware, the booted debug probe does not enumerate correctly and the MCUXpresso IDE (or other toolchain) is unable to see the debug probe. This problem is normally caused by on old, obsolete, version of the VCOM driver being found by Windows instead of the correct driver. To see if this is the cause of a problem on your computer, find the version number of the LPC-Link2 VCOM driver. The obsolete driver version is 1.0.0.0.

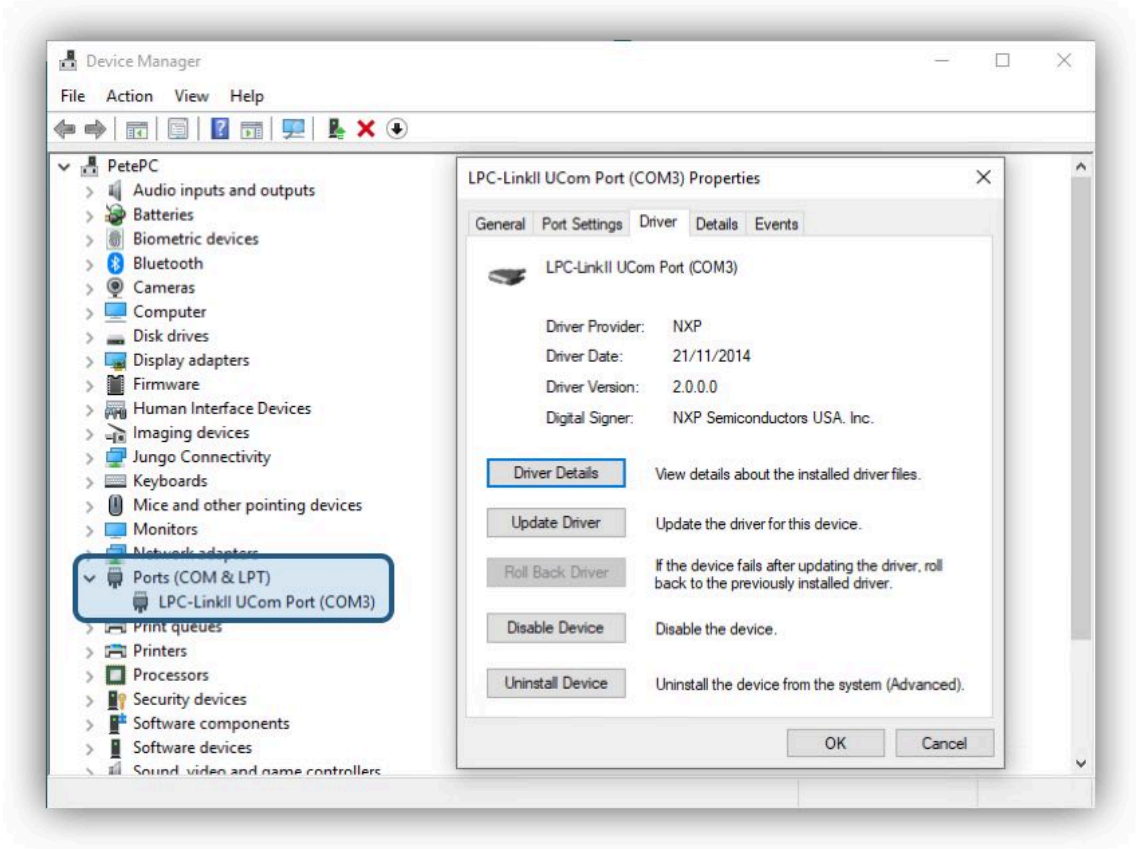
To find the version number of the LPC-Link2 VCOM driver

If you are using a soft-booted LPC-Link2 debug probe, start by booting your LPC-Link2, as described in Manually booting LPC-Link2 [280]. If your LPC-Link2 debug probe is booting from an image preprogrammed into the flash, you can skip this step.

Once your LPC-Link2 has booted, find the device in Device Manager and look at the driver version number.

- Open the Windows Device Manager
- Expand the “Ports (COM and LPT)” section
- Right-click on “LPC-LinkII UCom Port”, and select Properties
- Click on the Driver tab of the Properties dialog

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Note: that this image shows the current correct version of the driver (2.0.0.0).

Removing the obsolete 1.0.0.0 LPC-LinkII UCOM driver

To remove the obsolete driver, perform the following actions:

1. In Device Manager, right-click on the LPC-LinkII UCOM device and select Uninstall
2. If there is an option to delete the driver software, make sure it is checked, and press OK
3. Select the menu item Action->Scan for hardware changes
4. In Windows Control Panel, select Add/Remove program or Uninstall a program option
5. Find the LPC Driver Installer, right-click on choose Uninstall
6. Let the uninstaller complete
7. Switch back to the Device Manager and Scan for hardware changes again
8. If the LPC-LinkII UCOM driver version is still present, Uninstall it again (steps 1 through 3) and repeat until the LPC-LinkII UCOM driver no longer appears
9. Now run the lpc_driver_installer.exe found in the MCUXpresso IDE “Drivers” directory

Note: A reboot is recommended after running the lpc_driver_installer.exe installer.

Now manually reboot the probe again (if softloading) and check **Windows – Devices and Printers** to see if the device now appears correctly as an LPC-Link2 CMSIS-DAP Vx.xxx.

If this fails to correct the problem, there is one final thing to try:

- Open a Command Prompt as the Administrative user and run the following commands

```
cd %temp%
pnputil -e >devices.txt
notepad devices.txt
```


All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Search devices.txt for an entry similar to this, and note down the Published name (oemXX.inf)

```
Published name :            oem38.inf
Driver package provider :   NXP
Class :                     Ports (COM & LPT)
Driver date and version :   09/12/2013 1.0.0.0
Signer name :               NXP Semiconductors USA. Inc.
```

- Using the name notes above, run the following command (replacing XX with the number found above)

```
pnputil -f -d oemXX.inf
```

21.9.7 Troubleshooting LPC-Link2

If you have been able to use LPC-Link2 in a debug session but now see issues such as “No compatible emulator available” or “Priority 0 connection to this core already taken” when trying to perform a debug operation ...

- Ensure you have shut down any previous debug session
  - You must close a debug session (press the Red ‘terminate’ button) before starting another debug session
- It is possible that the debug driver is still running in the background. Use the task manager or equivalent to kill any tasks called:
  - redlinkserv
  - arm-none-eabi_gdb*
  - crt_emu_*

MCUXpresso IDE provides an IDE button  to kill all low level debug executables.

A failure occurring while initiating a debug connection might also be caused by the GDB client being unable to communicate with the GDB stub. In this case, the error usually indicates a networking-related error, such as “Connection timed out”. Firewall and the launch configuration should be checked. Note that the stub will need to be listening on a networking port in order to communicate with the GDB client. For more details about the Debug Server Connection parameters, see also Editing a Launch Configuration (LinkServer) [120] section.

If your host has never worked with LPC-Link2, then the following may help to identify the problem:

- Try manually booting your LPC-Link2 as per Manually booting LPC-Link2, and ensure that the drivers have installed correctly.
- Try a different USB cable!
- Try a different USB port. If your host has USB3 and USB2, then try a USB2 port
  - there are known issues with motherboard USB3 firmware, ensure your host is using the latest driver from the manufacturer. **Note**: this is not referencing the host OS driver but the motherboard firmware of the USB port
- If using a USB hub, first try a direct connection to the host computer
- If using a USB hub, try using one with a separate power supply – rather than relying on the supply over USB from your PC.
- Try completely removing and re-installing the host device driver. See also LPC-Link2 fails to enumerate [282] above.
- If using Windows 8.1 or later, then sometimes the Windows USB power settings can cause problems. For more details use your favourite search engine to search for “windows usb power settings” or similar.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

21.10 Using and troubleshooting MCU-Link

21.10.1 MCU-Link hardware

MCU-Link is a new powerful and cost effective debug probe architecture that can be used seamlessly with MCUXpresso IDE, and is also compatible with 3rd party IDEs that support CMSIS-DAP protocol.

There are a range of debug solutions based on the MCU-Link architecture, which include the standalone very low cost base model (MCU-Link probe), a fully featured MCU-Link Pro probe and various implementations built into NXP evaluation boards. MCU-Link Pro includes many additional features to facilitate embedded software development, like energy consumption analysis and support of peripheral and host emulation via USB bridging functions. On-board implementations support all the base model features and can optionally support additional features available on MCU-Link Pro.

MCU-Link solutions are based on the powerful, low power LPC55S69 microcontroller and all versions run the same firmware from NXP.

MCU-Link common features:

- CMSIS-DAP firmware to support all NXP Arm Cortex®-M based MCUs with SWD or JTAG debug interfaces
- High speed USB host interface
- USB to target UART bridge (VCOM)
- SWO profiling and I/O features

MCU-Link Pro additional features:

- SEGGER J-Link firmware option
- Circuitry to measure the target's supply voltage and current drawn
- Trigger-based measurement
- Analog signal trace input
- A second USB to target UART bridge (VCOM)
- USB SPI and I2C bridges for programming/provisioning and host-based application development
- Option to power target system at up to 350mA (at 1.8V or 3.3V)
- On-board, user-programmable LPC804 for peripheral emulation
- Multiple status LEDs for diagnosis of issues
- Target reset button

For more details, please refer to the Getting Started guides available on the product web pages on nxp.com:

- MCU-Link debug probe: https://www.nxp.com/pages/:MCU-LINK
- MCU-Link Pro debug probe: https://www.nxp.com/pages/:MCU-LINK-PRO

21.10.2 MCU-Link CMSIS-DAP firmware

MCU-Link debug probes are factory programmed with NXP's CMSIS-DAP protocol based firmware, which also supports all other features supported in hardware.

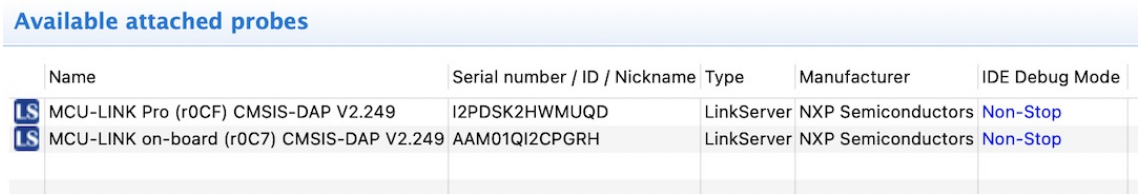
Besides SWD/JTAG debug probe functionality, NXP's CMSIS-DAP firmware for MCU-Link by default also includes bridge channels to provide:

- Support for SWO Trace capture from the MCUXpresso IDE

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Support for a UART VCOM port (UART interface to target)
- Support for a second UART VCOM port [MCU-Link Pro]
- Support for USB serial I/O (SPI, I2C, GPIO) bridge compatible with LIBUSBSIO [MCU-Link Pro]
- Support for energy measurement from the MCUXpresso IDE [MCU-Link Pro]

MCU-Link probe type and firmware details are displayed in MCUXpresso IDE's Probes Discovered dialog when the probe is attached:



Probes Discovered will provide an indication if a newer firmware version is available. It is recommended to update the MCU-Link firmware to the latest version using the provided firmware update utility. Go to the Design Resources section of the board web page and navigate to "Development software" from the SOFTWARE section. Installation packages for each host OS will be shown. Download and run the installer for your host OS. A step-by-step installation guide is provided at the board web page on nxp.com

CMSIS-DAP versions

Firmware versions V2.xxx implement an older version of CMSIS-DAP 1.1.0 and use USB HID as interface to the host PC.

Firmware versions V3.xxx are based on the latest CMSIS-DAP version 2.1.0 and use WinUSB as interface to the host PC and is therefore faster. Since the firmware implements Microsoft descriptors to declare WCID (Windows Compatible ID), no additional WinUSB driver is required on Windows.



Note

Firmware versions 3.xxx are supported in MCUXpresso IDE 11.7.0 or newer. If using an older MCUXpresso IDE product, please install MCU-Link Firmware version V2.263.

MCU-Link USB Details

The standard utilities to explore USB devices on MCUXpresso IDE supported host platforms are:

- Windows – Device Manager
  - MCUXpressoIDE also provides a listusb utility in:
    - *install_dir/ide/binaries/Scripts*
- Linux – terminal command: lsusb
- macOS – terminal command: system_profiler SPUSBDataType

In ISP mode (firmware update enabled), MCU-Link will appear as a USB device with details:

```
Device VendorID/ProductID:   0x1FC9/0x0021
```

In normal use, MCU-Link will appear as a USB device with details:

```
Device VendorID/ProductID:   0x1FC9/0x0143
```

MCU-Link will appear in Windows Control Panel -> Hardware and Sound -> Devices and Printers similar to below:

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



Note: Text details will vary depending on firmware version and probe configuration.

21.10.3 MCU-Link host drivers

MCU-Link debug probes are supported on Windows 10, macOS and Ubuntu Linux platforms. MCU-Link probes use standard OS drivers, however on Windows an inf driver is provided to allow displaying friendly names in Device Manager for the MCU-Link VCom Port(s). The driver for MCU-Link is installed as part of MCUXpresso IDE installation process as well as during the installation of the firmware update utility. If you need to reinstall the driver, it can be found at:

```
MCUXpresso IDE: <install_dir>\Drivers\MCU-Link
MCU-LINK_installer: <install_dir>\drivers
```

To install the driver, navigate to MCU-Link drivers and install the file by right-click –> Install:

- mcu-link-vcom.inf

21.10.4 MCU-Link JLink-compatible firmware

A custom version of J-Link firmware to make MCU-Link Pro compatible with SEGGER's popular J-Link LITE is also available, but note that this firmware is limited to supporting debug (including SWO) and VCOM features only.

21.10.5 Troubleshooting MCU-Link

If you have been able to use MCU-Link in a debug session but now see issues such as "No compatible emulator available" or "Priority 0 connection to this core already taken" when trying to perform a debug operation ...

- Ensure you have shut down any previous debug session
  - You must close a debug session (press the Red 'terminate' button) before starting another debug session
- It is possible that the debug driver is still running in the background. Use the task manager or equivalent to kill any tasks called:
  - redlinkserv
  - arm-none-eabi_gdb*
  - crt_emu_*

Use MCUXpresso IDE button  to kill all low level debug executables.

A failure occurring while initiating a debug connection might also be caused by the GDB client being unable to communicate with the GDB stub. In this case, the error usually indicates a networking-related error, such as "Connection timed out". Firewall and the launch configuration should be checked. Note that the stub will need to be listening on a networking port in order to communicate with the GDB client. For more details about the Debug Server Connection parameters, see also Editing a Launch Configuration (LinkServer) [120] section.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

If your host has never worked with MCU-Link, then the following may help to identify the problem:

- Try updating the MCU-Link firmware and ensure that the drivers have installed correctly. See MCU-Link firmware [285] above.
- Make sure it is plugged into a high speed USB 2.0 port!
- Try a different USB cable!
- Try a different USB port. If your host has USB3 and USB2, then try a USB2 port
  - there are known issues with motherboard USB3 firmware, ensure your host is using the latest driver from the manufacturer. **Note**: this is not referencing the host OS driver but the motherboard firmware of the USB port
- If using a USB hub, first try a direct connection to the host computer
- If using a USB hub, try using one with a separate power supply – rather than relying on the supply over USB from your PC.
- Try completely removing and re-installing the host device driver. See MCU-Link Windows Drivers [287] above.

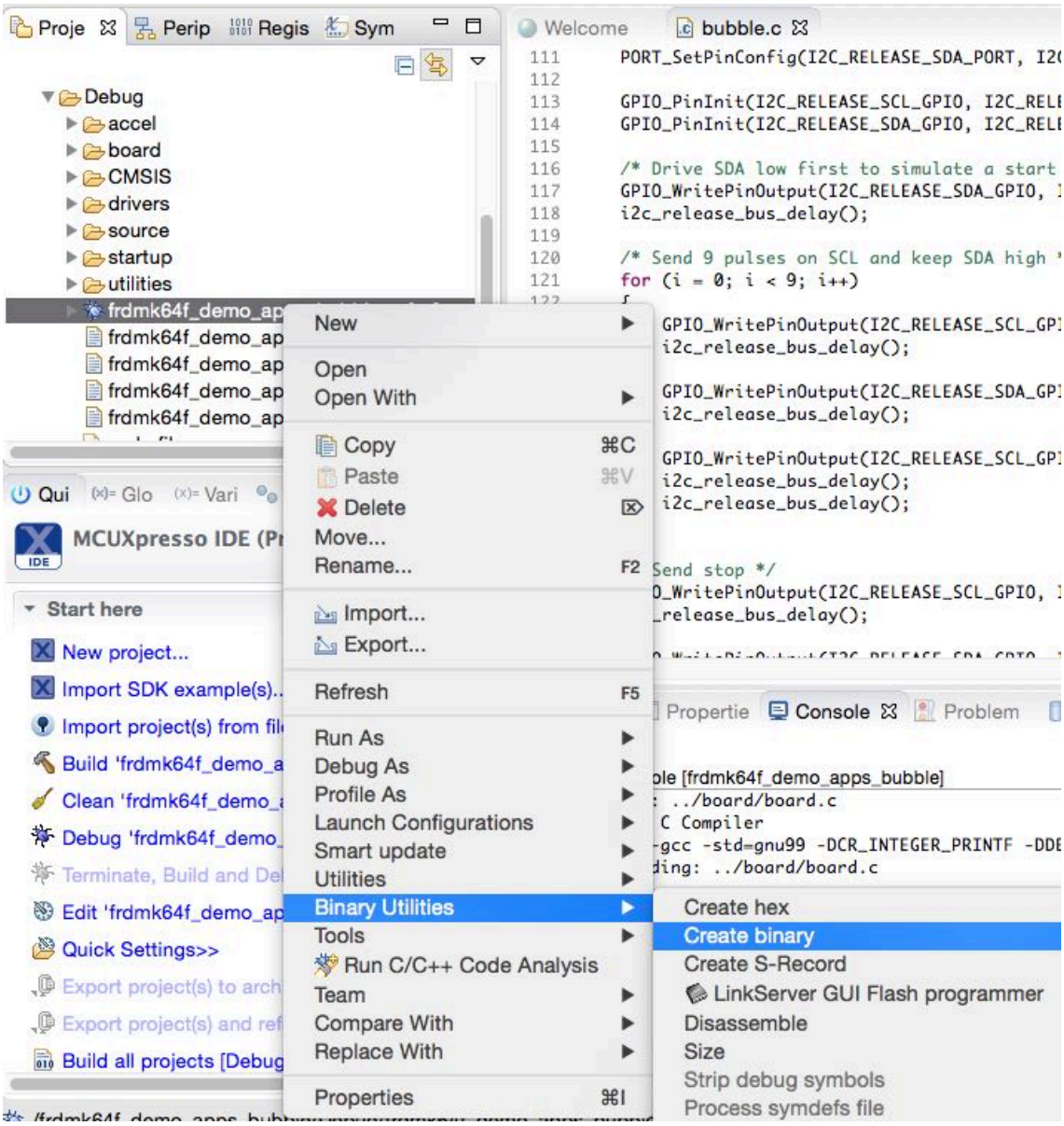
21.11 Creating bin, hex or S-Record files

When building a project, the MCUXpresso IDE tools create an ARM executable format (AXF) file – which is actually standard ELF/DWARF file. This file can be programmed directly down to your target using the MCUXpresso IDE debug functionality, but it may also be converted into a variety of formats suitable for use in other external tools.

21.11.1 Simple conversion within the IDE

The simplest way to create a one-off binary or hex file is to open up the Debug (or Release) folder in Project Explorer right click on the .axf file, and " **Binary Utilities -> Create binary**" (or Create hex, S-Record).

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



You can also change the underlying commands and options that are called by these menu entries from the " **Preferences -> MCUXpresso IDE -> Utilities**" preference page.

21.11.2 From the command line

The above “Binary Utilities” option within the IDE GUI is simply invoking the command line objcopy tool (arm-none-eabi-objcopy). Objcopy can convert into the following formats:

- srec (Motorola S record format)
- binary
- ihex (Intel hex)
- tekhex

For example, to convert example.axf into binary format, use the following command:

arm-none-eabi-objcopy -O binary example.axf example.bin

If you ctrl-click on the project name on the right hand side of the bottom bar of the IDE, this will launch a command prompt in the project directory with appropriate tool paths set up. You can also use the Project Explorer right-click “Utilities->Open command prompt here” option to do this.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

All you need to do before running the objcopy command is change into the directory of the required Build configuration.

21.11.3 Automatically converting the file during a build

Objcopy may be used to automatically convert an axf file during a build. To do this, create an appropriate Post-build step

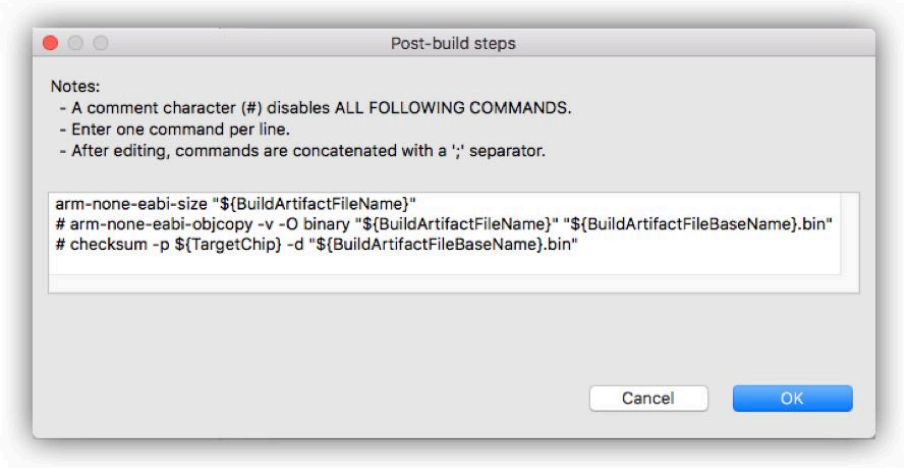
21.11.4 Binary files and checksums

When creating a binary file for most LPC MCUs, you also need to ensure that you apply a checksum to it – so that the LPC bootloader sees the image as being valid. Generally the linker script will do this if the managed linker script mechanism is used. Otherwise the “checksum” utility found in the \ide\binaries subdirectory of your MCUXpresso IDE installation can be used.

21.12 Post-build (and Pre-build) steps

It is sometimes useful to be able to automatically post-process your linked application, typically to run one or more of the GNU ‘binutils’ on the generated AXF file.

For example, any application project that you create using the Project wizard will have least one such “post-build step” - typically to display the size of your application.



Note: Additional commands may also be listed (for example to create a binary and to run a checksum command), but be commented out by use of a # character and hence not executed. Any commands following a comment #command will be ignored.

Adding additional steps is very simple. In the below example we are going to carry out three post-link steps:

- displaying the size of the application
- generate an interleaved C / assembler listing
- create a hex version of the application image

To do this:

- Open the Project properties. There are a number of ways of doing this. For example, make sure the Project is highlighted in the Project Explorer view then open the menu “Project -> Properties”.
- In the left-hand list of the Properties window, open “C/C++ Build” and select “Settings”.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

- Select the “Build steps” tab
- In the “Post-build steps - Command” field, click 'Edit...'
  - Paste in the lines below and click 'OK'

```
arm-none-eabi-size ${BuildArtifactFileName};
arm-none-eabi-objdump -S ${BuildArtifactFileName} > ${BuildArtifactFileBaseName}.lss;
arm-none-eabi-objcopy -O ihex ${BuildArtifactFileName} ${BuildArtifactFileBaseName}.hex;
```

- Click apply
- Repeat for your other Build Configurations (Debug/Release)

Next time you do a build, this set of post-build steps will now run, displaying the application size in the console, creating you an interleaved C/assembler listing file called *./lss* and a hex file called *hex*.

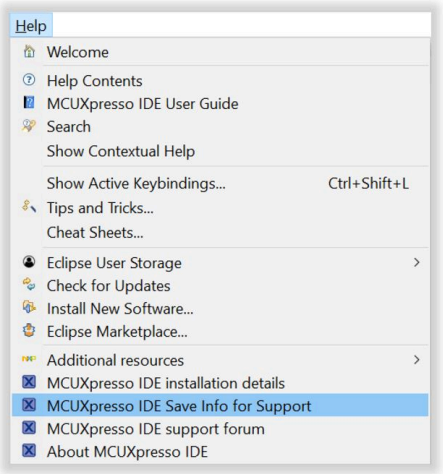
Note: Pre-build steps can be added to a project in exactly the same way if required.

21.12.1 Temporarily removing post-build steps

If you want to temporarily remove a step from your post-build process, rather than deleting it completely – move that entry to the end of the line and pre-fix it with a “#” (hash) character. This acts as a comment, causing the rest of the post-build steps to be ignored.

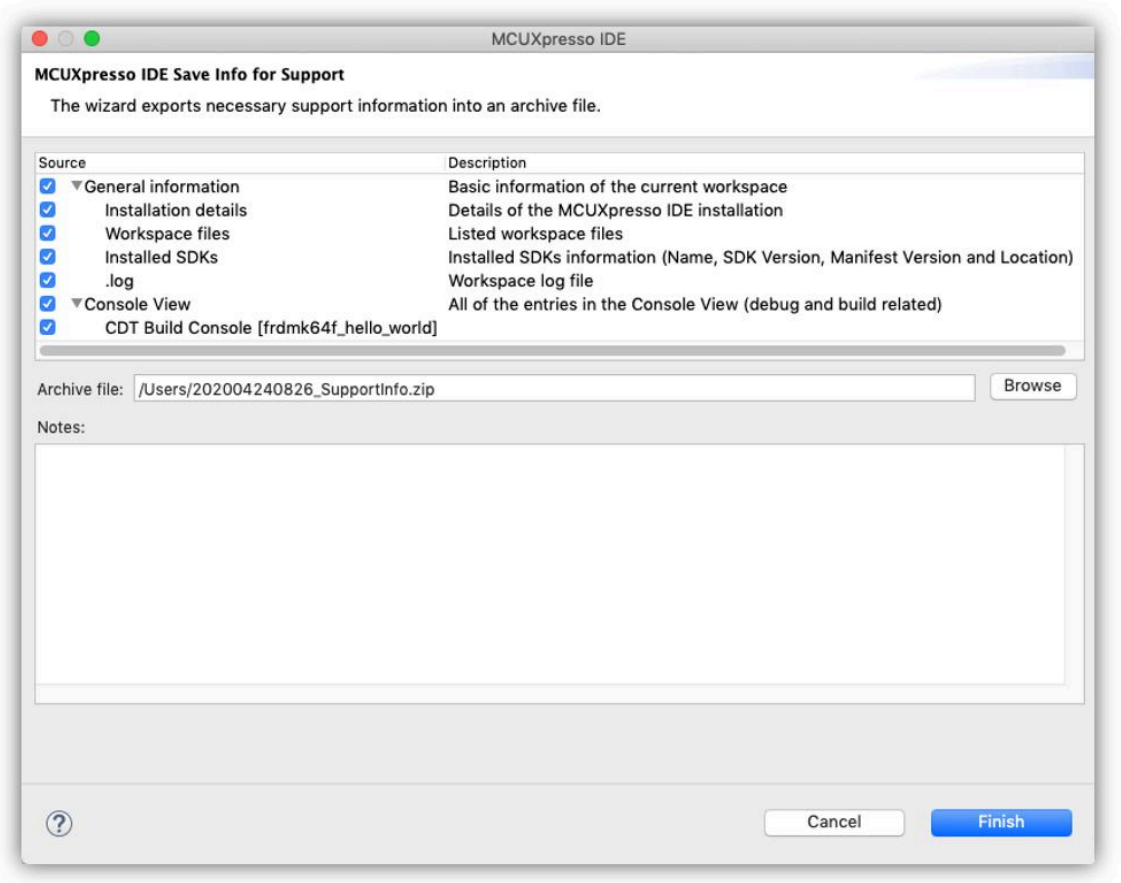
21.13 Save Info for Support

When reporting an issue, it is recommended to send to product team all necessary information to easily reproduce the error you experience in the IDE working environment. For this reason, a new option (" **Help -> MCUXpresso IDE Save Info for Support**") is introduced in Eclipse Help menu that helps you to gather and pack all information related to workspace, logs and consoles:



Once wizard is open, various types of information can be individually selected (by default, all categories are preselected):

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.



- Installation details – information about installation paths, tool version, OS and java versions, IDE plugins versions.
- Workspace files – a list with all filenames and directory structure within the workspace.
- Installed SDKs – a list with all installed SDKs generic information: name, SDK and manifest versions, location path.
- .log – can be one or more files and contain Eclipse logs generated in <workspace dir>/.metadata directory.
- Console View – all consoles (Console eclipse views) content: build log, debug and flash programming logs, standard input/output console, etc.

Note: No sensitive information like source code or file content (other than eclipse/build/debug logs) is included!

Optionally, add details in the “Notes:” edit box, located on the bottom of the “Save Info for Support” window, regarding any information you think it might be helpful: steps to reproduce, errors you observe, expected results, etc. If used, the text content will be added to the final zip file.

After the selection is done and a proper zip filename and path are chosen, press ‘Finish’ button to generate the archived support information. Use afterwards this zip as an attachment when reporting the problem.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

22. Revision History

Table 22.1. Revision history

Revision No.	Release Date	Changes
11.8.0	July 2023	Major release version update. See chapter 2 for details.

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

23. Legal information

23.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

23.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including – without limitation – lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/

All information provided in this document is subject to legal disclaimers
© 2023 NXP Semiconductors. All rights reserved.

profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications. In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and © customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

23.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

© 2023 NXP Semiconductors. All rights reserved.