# Getting Started with MCUXpresso SDK for LPC5410x

## 1 Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC Microcontrollers. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains FreeRTOS and various other middleware to support rapid development.

For supported toolchain versions, see the MCUXpresso SDK Release Notes Supporting LPCXpresso54102 (document MCUXSDKLPC5410XRN).

For more details about MCUXpresso SDK, see the MCUXpresso SDK homepage MCUXpresso-SDK: Software Development Kit for MCUXpresso.

### Contents

1	Overview	1
2	MCUXpresso SDK board support folders	2
3	Run a demo application using IAR	4
4	Run a demo using Keil® MDK/µVision	8
5	Run a demo using MCUXpresso IDE	14
A	How to determine COM port	33
B	Updating debugger firmware	



#### MCUXpresso SDK board support folders



Figure 1. MCUXpresso SDK layers

## 2 MCUXpresso SDK board support folders

MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm<sup>®</sup> Cortex<sup>®</sup>-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board\_name> folder, there are various sub-folders to classify the type of examples they contain. These include (but are not limited to):

- cmsis\_driver\_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- demo\_apps: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver\_examples: Simple applications intended to concisely illustrate how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- rtos\_examples: Basic FreeRTOS<sup>TM</sup> OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers

## 2.1 Example application structure

This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see the *MCUXpresso SDK API Reference Manual* document (MCUXSDKAPIRM).

Each <board\_name> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello\_world example (part of the demo\_apps folder), the same general rules apply to any type of example in the <board\_name> folder.

In the hello\_world application folder you see the following contents:



Figure 2. Application folder structure

All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

## 2.2 Locating example application source files

When opening an example application in any of the supported IDEs, a variety of source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- devices/<device\_name>: The device's CMSIS header file, MCUXpresso SDK feature file and a few other things.
- devices/<device\_name>/cmsis\_drivers: All the CMSIS drivers for your specific MCU.
- devices/<device\_name>/drivers: All of the peripheral drivers for your specific MCU.
- devices/<device\_name>/<tool\_name>: Toolchain-specific startup code. Vector table definitions are here.
- devices/<device\_name>/utilities: Items such as the debug console that are used by many of the example applications.

#### Run a demo application using IAR

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

## 3 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello\_world demo application targeted for the LPCXpresso54102 hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

## 3.1 Build an example application

The following steps helps you build the hello\_world example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

<install\_dir>/boards/<board\_name>/<example\_type>/<application\_name>/iar

Using the LPCXpresso54102 hardware platform as an example, the hello\_world workspace is located in

<install\_dir>/boards/lpcxpresso54102/demo\_apps/hello\_world/iar/hello\_world.eww

Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu. For this example, select the "hello\_world - debug" target.



## Figure 3. Demo build target selection

3. To build the demo application, click the "Make" button highlighted in red below.





4. The build completes without errors.

Run a demo application using IAR

## 3.2 Run an example application

To download and run the application, perform these steps:

- 1. Download and install LPCScrypt or the Windows<sup>®</sup> operating systems driver for LPCXpresso boards from www.nxp.com/lpcutilities. This installs required drivers for the board.
- 2. Connect the development platform to your PC via USB cable between the Link2 USB connector (named Link for some boards) and the PC USB connector. If connecting for the first time, allow about 30 seconds for the devices to enumerate.
- 3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
  - a. 115200 (reference BOARD\_DEBUG\_UART\_BAUDRATE variable in board.h file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit

ategory:					
Session	Basic options for you	ur PuTTY session			
	Specify the destination you want to connect to				
	Serial li <u>n</u> e	Speed			
Bell	COM16	115200			
Features	Connection type:	ogin © <u>S</u> SH 💿 Serjal			
Appearance Behaviour	Load, save or delete a stored Saved Sessions	session			
Selection	Debug				
Colours	Default Settings Debug	Load			
Data		Sa <u>v</u> e			
Telnet		Delete			
⊕ SSH Serial	Close window on exit: Always Never	<ul> <li>Only on clean exit</li> </ul>			

## Figure 5. Terminal (PuTTY) configuration

4. In IAR, click the "Download and Debug" button to download the application to the target.



### Figure 6. Download and Debug button

5. The application is then downloaded to the target and automatically runs to the main() function.

i 🗅 🗅 🖻 🚇 🗎 🕹 🖒 🗅 🖸	C	C	- < Q > ≒ ⊭ < ♀ > ≧ ▷ <b>▷ ◎ ●</b> ∈ Θ ⊂ <mark>♀</mark> <b>○</b> → ∀ + + ● ● ± - ₽ = ₩ ₩ ₽
Workspace	•	φ×	hello_world.c X
Debug		-	main()
Files	•	•	$\begin{array}{c} 41\\ 42 \end{array} \rightarrow \\                                 $
E ■ hello_world - Debug - 10 ■ board - 10 ■ doc	•		43 * Prototypes 44
			46 ⊟ /***********************************
i utilities L⊕ ∎ Output			49 □ /*! 50   * @brief Main function 51   */
			52         int main (void)           53         (           54         char ch;
			<pre>55 56 /* Init board hardware. */ 57 /* attach 12 MHz clock to FLEXCOMM0 (debug console) */ 58 CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);</pre>
			<pre>59 60 BOARD_InitPins(); 61 BOARD_BootClockFROHF48M(); 62 BOARD_InitDebugConsole();</pre>

### Figure 7. Stop at main() when running debugging

6. Run the code by clicking the "Go" button to start the application.



### Figure 8. Go button

7. The hello\_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



Figure 9. Text display of the hello\_world demo

## 3.3 Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

<install\_dir>/boards/<board\_name>/multicore\_examples/<application\_name>/<core\_type>/iar

Begin with a simple dual-core version of the Hello World application. The multicore Hello World IAR workspaces are located in this folder:

<install\_dir>/boards/lpcxpresso54102/multicore\_examples/hello\_world/cm0plus/iar/hello\_world\_cm0plus.eww <install\_dir>/boards/lpcxpresso54102/multicore\_examples/hello\_world/cm4/iar/hello\_world\_cm4.eww

Build both applications separately by clicking the "Make" button. It is requested to build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

## 3.4 Run a multicore example application

The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 - 4 as described in *Run an example application*. These steps are common for both single core and dual-core applications in IAR.

After clicking the "Download and Debug" button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary image are loaded into the device flash memory and the primary core application is executed. It stops at the default C language entry point in the *main()* function.

Run both cores by clicking the "Start all cores" button to start the multicore application.



### Figure 10. Start all cores button

During the primary core code execution, the auxiliary core is released from the reset. The hello\_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check the terminal settings and connections.



### Figure 11. Hello World from primary core message

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. When both cores are running, use the "Stop all cores" and "Start all cores" control buttons to stop or run both cores simultaneously.



Figure 12. "Stop all cores" and "Start all cores" control buttons

4 Run a demo using Keil<sup>®</sup> MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello\_world demo application targeted for the LPCXpreeso54102 hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

## 4.1 Install CMSIS device pack

After the MDK tools are installed, Cortex<sup>®</sup> Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions, and flash programming algorithms. Follow these steps to install the appropriate CMSIS pack.

1. Open the MDK IDE, which is called µVision. In the IDE, select the "Pack Installer" icon.

1	File	Edit	View	Project	Flash	Debug	Peripherals	Tools	SVCS	Window
	n	<b>2</b>	9	8 45	8	50	(m = ) 1/2	12.1	1 12	律律》
ALL ALL ALL	1			Lond Lond				S   1	1 2	* *

### Figure 13. Launch the Pack Installer

2. After the installation finishes, close the Pack Installer window and return to the  $\mu$ Vision IDE.

## 4.2 Build an example application

• Open the desired example application workspace in: <install\_dir>/boards/<board\_name>/<*example\_type>/* <application\_name>/mdk

The workspace file is named <demo\_name>.uvmpw, so for this specific example, the actual path is:

<install\_dir>/boards/lpcxpresso54102/demo\_apps/hello\_world/mdk/hello\_world.uvmpw

• To build the demo project, select the "Rebuild" button, highlighted in red.



Figure 14. Build the demo

• The build completes without errors.

## 4.3 Run an example application

To download and run the application, perform these steps:

- 1. Download and install LPCScrypt or the Windows driver for LPCXpresso boards from www.nxp.com/lpcutilities . This installs required drivers for the board.
- 2. Connect the development platform to your PC via USB cable between the Link2 USB connector and the PC USB connector. If connecting for the first time, allow about 30 seconds for the devices to enumerate See Section 8.2 to update the debug probe flash using LPCScrypt.

- 3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
  - a. 115200 or 9600 baud rate, depending on your board (reference BOARD\_DEBUG\_UART\_BAUDRATE variable in board.h file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit

- Session	Basic options for your PuTT	Ysession
Logging	Specify the destination you want to co	onnect to
Teminal	Serial line	Speed
Keyboard	COM16	115200
- Features ⊒- Window	Connection type:	SSH Serial
Appearance Behaviour	Load, save or delete a stored session Saved Sessions	
- Selection	Debug	
Colours	Default Settings	Load
- Connection	Debug	Save
Proxy		Jave
Telnet Rlogin		Delete
⊡-SSH	C	
Serial	Close window on exit: Always Never Only	on clean exit

### Figure 15. Terminal (PuTTY) configurations

4. In μVision, after the application is properly built, click the "Download" button to download the application to the target.

🛛 🔗 🖾 📾 🤐 🔛	hello_world Debug	- 🔊
Project	<b>д 💌</b>	
WorkSpace		
🗄 🔧 Project: hello_world	ł	

### Figure 16. Download button

5. After clicking the "Download" button, the application downloads to the target and should be running. To debug the application, click the "Start/Stop Debug Session" button, highlighted in red.

🗋 😂 🖬 🖉	□ 🧉 🖬 🗿 🕺 🐴 🖏 🕫 🔄 🕐 🔅 🎘 🎘 🎼 淳 /// /// /// /// //////////////////						
詩   🗉 📀   そ	ት 🖓 🖓 🖓 👘						
Registers	<b>4</b>	) Disassembly					
Register	Value	0x00001D7E 4000 DCW 0x4000					
Core		58: CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);					
<mark>R0</mark>	0x00001D81						
R1	0x0200002C	(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx					
R2	0x0000000	60: BOARD InitFins():					
R3	0x00001A2D						
R4	0x00001E94	J.C.					
R6	0x00001E34	startup_LPC54102_cm4.s hello_world.c					
B7	0x00000000						
	0x00000000	43 * Prototypes					
	0x00000000	44 ************************************					
R10	0x00000000	45					
R11	0x0000000	46 = /***********************************					
R12	0x0000000	47 * Code					
R13 (SP)	0x02010000	48 ************************************					
R14 (LR)	0x000003BD	49 🗇 /*!					
R15 (PC)	0x00001D80	50 * @brief Main function					
	0x61000000	51 L */					
Banked		52 int main (void)					
- Jotemal							
Mode	Thread	54 Char Ch;					
Privilege	Privileged	55 /* Thit heard hardware */					
Stack	MSP	57 /* attach 12 MHz clock to USARTO (debug console) */					
States	200	D 58 CLOCK AttachClk (BOARD DEBUG UART CLK ATTACH):					
Sec	0.00002000	59					
		60 BOARD InitPins();					
		61 BOARD_BootClockPLL96M(); /* Rev B device can only support max core frequency to 96Mhz.					
		62 Rev C device can support 100Mhz,use BOARD_BootClockPLL100M() to boot core to 100Mhz.					
		63 DEVICE_ID1 register in SYSCON shows the device version.					
		64 - More details please refer to user manual and errata. */					
		65 BOARD_InitDebugConsole();					
		o/ PRINIF("nelio world.\r\n");					
		60 while (1)					
1							

### Figure 17. Stop at main() when run debugging

6. Run the code by clicking the "Run" button to start the application.



Figure 18. Go button

Getting Started with MCUXpresso SDK for LPC5410x, Rev. 0, 08/2019

The hello\_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



Figure 19. Text display of the hello\_world demo

## 4.4 Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

<install\_dir>/boards/<board\_name>/multicore\_examples/<application\_name>/<core\_type>/mdk

Begin with a simple dual-core version of the Hello World application. The multicore Hello World Keil MSDK/µVision<sup>®</sup> workspaces are located in this folder:

<install\_dir>/boards/pcxpresso54102/multicore\_examples/hello\_world/cm0plus/mdk/hello\_world\_cm0plus.uvmpw <install\_dir>/boards/lpcxpresso54102/multicore\_examples/hello\_world/cm4/mdk/hello\_world\_cm4.uvmpw

Build both applications separately by clicking the "Rebuild" button. Build the application for the auxiliary core (cm0plus) first because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

## 4.5 Run a multicore example application

The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 - 5 as described in *Run an example application*. These steps are common for both single core and dual-core applications in  $\mu$ Vision.

Both the primary and the auxiliary image is loaded into the device flash memory. After clicking the "Run" button, the primary core application is executed. During the primary core code execution, the auxiliary core is released from the reset. The hello\_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check your terminal settings and connections.



Figure 20. Hello World from primary core message

An LED controlled by the auxiliary core starts flashing indicating that the auxiliary core has been released from the reset and is running correctly.

Attach the running application of the auxiliary core by opening the auxiliary core project in the second  $\mu$ Vision instance and clicking the "Start/Stop Debug Session" button. After this, the second debug session is opened and the auxiliary core application can be debugged.

The Ear New Hojet Hash bedag respiretais toos sites finitably help	
□ 😂 🖬 🕼   ※ 🛍   ≫ ┍   ← →   🥐 🎘 🎘 🤁   導 導 /// /// // // // //   🐼 🕟	
Registers <b>4</b> 🖸 Disassembly	
Register Value 51: for (i = 0; i < 1000000; ++i)	
52: {	
R0 0x0000A7D4 0x20010B5A 9000 STR r0,[sp,#0x00]	
R1 0x000F4240 0x20010B5C E003 B 0x20010B66	
R2 0x20000000 53:asm("NOP"); /* delay */	
R3 0x0000000 54: }	
R4 0x4008E300 0x20010B5E BF00 NOP	
R5 $0x0000001$ (i = 0 · i < 1000000 · ++i)	
R6 0x20010C0C	
R7 0xFFFFFFF hello_world_core1.c	
R8 0xFFFFFFF	
R9 0xFFFFFFF 30 * Prototymes	
R10 0xFFFFFFF 40 therefore the transference to	******
R11 0xFFFFFFF 41	
R12 0xFFFFFFF 42 / /*********************************	******
R13 (SP) 0x200267F0 43 * Code	
R14 (LR) 0x20010B9F	******
R15 (PC) 0x20010B68 45 🖂 / * 1	
E xPSR 0x01000000 46 * Obrief Function to create delay for Led blink.	
The Banked	
System 48 void delay (void)	
EInternal 49 E {	
Mode Thread 50 volatile uint32 t i = 0;	
Privilege Privileged 51 for (i = 0; i < 1000000; ++i)	
Stack MSP 52 - {	
53 asm("NOP"); /* delay */	
54 - }	
55 }	
56 -	



#### NOTE

Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK Package.

This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello\_world demo application targeted for the LPCXpresso54102 platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

## 5.1 Select the workspace location

Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be outside of the MCUXpresso SDK tree.

## 5.2 Build an example application

To build an example application, follow these steps.

1. Drag and drop the SDK zip file into the "Installed SDKs" view to install an SDK. In the window that appears, click the "OK" button and wait until the import has finished.



## Figure 22. Install an SDK

2. On the Quickstart Panel, click "Import SDK example(s)...".

U Quickstart Panel 🔤 Global Variables 🔅 Variables 🌯 Breakpoints 🗄 Outline	
MCUXpresso IDE - Quickstart Panel No project selected	
<ul> <li>Create or import a project</li> </ul>	
New project  New project  Import SDK example(s)  Import project(s) from file system	
* Build your project	
Build Clean	
* Debug your project	🗙 🕶 🔛 🕶 🛃 🕶
Terminate, Build and Debug	
* Miscellaneous	
Edit project settings	
Quick Settings>>	
Export project(s) to archive (zip)	
P Export project(s) and references to archive (zip)	
Build all projects []	

### Figure 23. Import an SDK example

3. In the window that appears, expand the "LPC5410x" folder and select "LPC54102J512". Then, select "lpcxpresso54102" and click the "Next" button.



## Figure 24. Select LPCXpresso54102 board

4. Expand the "demo\_apps" folder and select "hello\_world". Then, click the "Next" button.

SDK Import Wizard	
A The source from the SDK will be copied into the workspace. If you want to use linked files, please unzip the 'SDK_2.x_LPCXpresso54102' SDK.	
Import projects	
Project name prefix: Ipcxpresso54102_ 2 Project name suffix:	<u>a</u>
✓ Use default location	
Location: C:\Users\B58254\Documents\MCUXpressoIDE_10.0.0_344\workspace\lpcxpresso54102_	Browse
Project Type	Project Options
C Project C++ Project C Static Library C++ Static Library     C++ Static Library	<ul> <li>✓ Enable semihost</li> <li>✓ Copy sources</li> </ul>
Examples	🔤 🖉 🐼 🙀 🕀 🕞
type to filter	
Name	Version
cmsis_driver_examples demo_apps i hello_world i power_manager_lpc i shell i utick_wakeup i driver_examples i multicore_examples i multicore_examples i rtos_examples	
? Aack	Next > Finish Cancel

### Figure 25. Select "hello\_world"

5. Ensure the option "Redlib: Use floating point version of printf" is selected if the cases print floating point numbers on the terminal (for demo applications such as dac32\_adc12, dac\_adc, dac\_cadc, ecompass, sai, coremark, mbedtls\_benchmark, wolfssl\_benchmark, and for mmcau\_examples such as mmcau\_api). Otherwise, there is no need to select it. Click the "Finish" button.

Run	а	demo	using	MCUX	presso	IDE
-----	---	------	-------	------	--------	-----

SDK Import	Wizard					
					NX	
X Advar	nced Settings					
▼ C/C++ Lik	brary Settings					
Set library ty	ype (and hosting variant)	Redlib (s	emihost-nf)			•
Redlib: U	Jse floating point version of printf	Newli	bNano: Use floating point	version of printf		
Redirect	SDK "PRINTF" to C library "printf"	Newli	bivano: Use noating point	version of scani		
Redirect	printf/scanf to ITM					
Redirect						
Memory det	tails					
Туре	Name	Alias	Location	Size	Driver	
Flash	PROGRAM_FLASH	Flash	0x0	0x80000	LPC5410x_512K.cfx	
RAM	SRAM0	RAM	0x2000000	0x10000		[ clit
RAM	SRAM1	RAM2	0x2010000	0x8000		Eart
RAM	SRAM2	RAM3	0x3400000	0x2000		
<ul> <li>Hardware</li> <li>Set Floating</li> <li>MCU C Co</li> <li>Language st</li> </ul>	e settings g Point type FPv4 (HardABI) ompiler tandard Compiler default					
<ul> <li>Multicore</li> <li>Optionally a</li> </ul>	slave projects settings allow an existing slave project to be associa	ated with th	nis project.			
Slave project	t for MOSLAVE		// Br	owse Link Section RAM2		-
By defau the slave	It, the slave images will be placed in the R e project was built.	AM2 block	of the master project's me	emory map. The slave memo	bry setting in the master project sho	ould match how
?				< Back	Next > Finish	Cancel

Figure 26. Select "User floating print version of printf"

## 5.3 Run an example application

For more information on debug probe support in the MCUXpresso IDE v11.0.1, visit community.nxp.com.

To download and run the application, perform these steps:

1. On the Quickstart Panel, click on "Debug 'lpcxpresso54102\_demo\_apps\_hello\_world' [Debug]".

🕑 Quickstart P 🔀 🗱 Global Varia (x)= Variables 💁 Breakpoints 📑 Outline	
MCUXpresso IDE (Free Edition)	<u>^</u>
▼ Start here	
X New project	
🔀 Import SDK example(s)	
Import project(s) from file system	=
🔏 Build 'lpcxpresso54102_demo_apps_hello_world' [Debug]	
🧹 Clean 'lpcxpresso54102_demo_apps_hello_world' [Debug]	
🎋 Debug 'lpcxpresso54102_demo_apps_hello_world' [Debug]	
🎋 Terminate, Build and Debug 'lpcxpresso54102_demo_apps_hello_world' [Debug]	
Edit 'lpcxpresso54102_demo_apps_hello_world' project settings	
🖄 Quick Settings>>	
Export project(s) to archive (zip)	-

### Figure 27. Debug "hello\_world" case

2. The first time you debug a project, the Debug Emulator Selection Dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click the "OK" button. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)

X Probes discovered				- • •
Connect to target: LPC54102J512				
1 probe found. Select the probe to us	e:			
Available attached probes				
Name	Serial number/ID	Туре	Manuf	IDE Debug Mode
LPC-LINK2 CMSIS-DAP V5.134	A00000002	LinkServe	NXP Sem	Non-Stop
Supported Probes (tick/untick to ena	ble/disable)			
MCUXpresso IDE LinkServer (inc.	CMSIS-DAP) probes			
P&E Micro probes				
SEGGER J-Link probes				
Probe search options				
Search again				
Remember my selection (for this La	unch configuration)			
2			OK	Cancel
			UK	Cancer

### Figure 28. Attached Probes: debug emulator selection

3. The application is downloaded to the target and automatically runs to main():



### Figure 29. Stop at main() when running debugging

4. Start the application by clicking the "Resume" button.

Project	Dun	Window		
<u>1</u>   🔌		10		14

Figure 30. Resume button

The hello\_world application is now running and a banner is displayed on the MCUXpresso IDE console window. If this is not the case, check your terminal settings and connections.

 Installed SDKs
 Properties
 Console &
 Problems
 Memory Instruction Trace
 SWO Trace Config
 Power Measurement Tool

 IMCUXpresso Semihosting Telnet console for Ipcxpresso54102\_demo\_apps\_hello\_world started on port 2333]
 ImcUXpresso Semihosting Telnet console for Ipcxpresso54102\_demo\_apps\_hello\_world started on port 2333]

SEGGER J-Link GDB Server V6.16a - Terminal output channel hello world.

.

### Figure 31. Text display of the hello\_world demo

## 5.4 Build a multicore example application

This section describes the steps required to configure MCUXpresso IDE v11.0.1 to build, run, and debug multicore example applications. The dual-core version of hello\_world example application targeted for the LPCXpresso54102 hardware platform is used as an example, though these steps can be applied to any multicore example application in the MCUXpresso SDK

1. Multicore examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for LPCXpresso54102 is installed and available in the "Installed SDKs" view, click "Import SDK example(s) ..." on the Quickstart Panel. In the window that appears, expand the "LPCxx" folder and select "LPC54102J512". Then, select "lpcxpresso54102" and click the "Next" button.



### Figure 32. Select the LPCXpresso54102 board

2. Expand the "multicore\_examples/hello\_world" folder and select "cm4". Because multicore examples are linked together, the cm0plus counterpart project is automatically imported with the cm4 project, and there is no need to select it explicitly. Click the "Finish" button.

SDK Import Wizard		
A The source from the SDK will be copied into the workspace. If you want to use linked files, please unzip the 'SDK_2.x_LPCXpresso54102' SDK. The	<b>JP</b>	G
Import projects		
Project name prefix: Project name suffix:		
Use default location		
Location: C:\Users\B58254\Documents\MCUXpressoIDE_10.0.0_344\workspace\lpcxpresso54102_		
Project Type	Project Options	
C Project C++ Project C Static Library C++ Static Library	Enable semihost Copy sources	E
Examples	ès   4	2 🗹 🐐
type to filter		
Name	Version	
▶ 🔲 🚪 cmsis_driver_examples		
Image:		
driver_examples		
Image: A multicore_examples		
▲ I i i hello_world		
cm0plus sm1 The following linked examples will be automatically imported, cm0plus		
Image: Second and the following initial examples will be automatically imported, chopids, so the second		
Image: Strange in the pingpong		
Image: The second se		
Image:		-
٠́		P.
Rext >	Finish	Cancel

### Figure 33. Select the hello\_world multicore example

3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the lpcxpresso54102\_multicore\_examples\_hello\_world\_cm4 project (multicore master project) in the Project Explorer, then choose the appropriate build target, "Debug" or "Release", by clicking the downward facing arrow next to the hammer icon, as shown below. For this example, select the "Debug" target.



Figure 34. Selection of the build target in MCUXpresso IDE

The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) causes the referenced auxiliary core application (cm0plus) to build as well.

## 5.5 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in Section 5.3, "Run an example application". These steps are common for both single core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. However, there is one additional dialogue that is specific to multicore examples, and requires selecting the target core. See the following figures as reference.





Figure 35. Debug "lpcxpresso54102\_multicore\_examples\_hello\_world\_cm4" case

X Probes discovered				
Connect to target: LPC54102 A The following probes have b P&E Micro probes SEGGER J-1	2 <b>J512</b> een disabled in the Link probes	preferences:		
Available attached pro	bes			
Name	Serial number/ID	Туре	Man	IDE Debug Mode
LPC-LINK2 CMSIS-DAP V5.	A00000002	LinkServer	NXP Se	Non-Stop
Supported Probes (tick/untick t	o enable/disable) r (inc. CMSIS-DAP) i	probes		
P&F Micro probes	(incl childs bray)	probes		
SEGGER J-Link probes				
Probe search options Search again				
Remember my selection (for t	his Launch configu	ration)		
?			OK	Cancel

Figure 36. Attached Probes: debug emulator selection



### Figure 37. Stop the primary core application at main() when running debugging

After clicking the "Resume All Debug sessions" button, the hello\_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.



### Figure 38. Hello World from the primary core message

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54102\_multicore\_examples\_hello\_world\_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click "Debug 'lpcxpresso54102\_multicore\_examples\_hello\_world\_cm0plus' [Debug]" to launch the second debug session.

<mark>ပ</mark> ံ Qi	uicksta 🛛	(×)= Global V	(x)= Variables	◎ Breakpoi	a Outline	
X	MCUXp	resso IDE (F	ree Edition	)		
IDE	, .			-		
▼ St	tart here					
X	New project					
X	Import SDK	example(s)				
•	Import proje	ect(s) from file sy	stem			
R	Build 'lpcxpr	resso54102_multi	core_examples_	hello_world_cm	0plus' [Debug	]
1	Clean 'lpcxp	resso54102_mult	icore_examples	_hello_world_cn	n0plus' [Debug	9]
*	Debug 'lpcx	presso54102_mul	ticore_example	s_hello_world_c	m0plus' [Debu	ig]
*	Terminate, E	Build and Debug	'lpcxpresso5410	2_multicore_exa	mples_hello_v	vorld_cm0
89 89	Edit 'lpcxpre	sso54102_multic	ore_examples_h	iello_world_cm0	plus' project s	ettings
2	Quick Settin	gs>>				
	Export proje	ct(s) to archive (z	tip)			
Ų,	Export proje	ct(s) and referen	ces to archive (a	cip)		
010	Build all proj	jects [Debug]				
•			(11			P

Figure 39. Debug "lpcxpresso54102\_multicore\_examples\_hello\_world\_cm0plus" case

1	X				×
	SWD Co	nfiguration			
	1 availat Target '	ble SWD Device detected Cortex-M0+' has been se	l. elected.		
	Device	Name	TAP Id	Details	
1	<b>V</b> 1	Cortex-M0+	0x2ba01477	APID:24770011	
	?			ОК	Cancel

Figure 40. Target core selection dialog

Xw	vorkspace - Develop - Ipcxpresso54102_multicore_examples_hello_world_cm4/source/hello_world_core0.c - MCUXpresso IDE	×
File	Edit Source Refactor Navigate Search Project Run FreeRTOS Window Help	
1	- 🗐 🐚   🗞 - 🗞 - 🐘   🔪   🕨 🗉 🔳 🖉 R. 👁 🗷   🗮 🛒 🍈 👘 🖷 R. 🗞 🚸 🕹 🔗 📕 🎋 🚸 - 🔾 - 🏰	-
: 🖉		
	🎋 Debug 🔀 🙀 🖬 🗸 🗖	
B	▲ X Ipcxpresso54102 multicore examples hello world cm4 Debug [C/C++ (NXP Semiconductors) MCU Application]	
<u> </u>	Ipcxpresso54102_multicore_examples_hello_world_cm4.axf [LPC54102J512 (cortex-m4)]	
29	Thread #1 1 (Stopped) (Suspended : Breakpoint)	
0101	main() at hello_world_core0.c:86 0x98e	
£	arm-none-eabi-gdb (7.12.0.20161204)	
****	Ipcxpresso54102_multicore_examples_hello_world_cm0plus Debug [C/C++ (NXP Semiconductors) MCU Application]	
8	Ipcxpresso34102_multicore_examples_helio_world_cmuplus.axt [LPC34102J312 (cortex-muplus)]	
U	arm-none-eabi-adb (7 12 0 20161204)	
(×)=		
(x)=		
•	Section Secti	
8	<pre>74 core1_image_size = (uint32_t)section_end("sec_core") - (uint32_t)&amp;core1_image_start;</pre>	*
	75 #endif 76 return corel image size:	
	77 }	
	78 #endif	
	79⊖ /*! 80 * @brief Main function	
	81 */	
	820 int main(void)	_
	83 { 84 #if !defined(K32W042S1M2 M4 SERIES)	
	85 /* Define the init structure for the switches*/	= -
	<pre> 86 gpio_pin_config_t sw_config = {kGPIO_DigitalInput, 0}; 87 markif</pre>	
	87 <b>#end1</b> †	
	89 /* Init board hardware.*/	
	90 /* attach 12 MHz clock to USART0 (debug console) */	
	91 CLOCK_ATTACHCIK(BOARD_DEBUG_DART_CLK_ATTACH); 92	
	93 BOARD_InitPins_Core0();	
	940 BOARD_BootClockPLL96M(); /* Rev B device can only support max core frequency to 96Mhz.	
	95 Revice can support 100Mnz,use BUARD_BOOTCLOCKPLL100M() to boot co 96 DEVICE ID1 register in SYSCON shows the device version.	
	97 More details please refer to user manual and errata. */	
	98 BOARD_InitDebugConsole();	-
	• • • • • • • • • • • • • • • • • • •	

### Figure 41. Two opened debug sessions

Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected and clicking the "Resume" button. The hello\_world multicore application then starts running. The primary core application starts the auxiliary core application during runtime, and the auxiliary core application stops at the beginning of the main() function. The debug session of the auxiliary core application is highlighted. After clicking the "Resume" button, it is applied to the auxiliary core debug session. Therefore, the auxiliary core application continues its execution.



### Figure 42. Auxiliary core application stops at the main function

At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both cores. This is done either by selecting both opened debug sessions (multiple selection) and clicking the "Suspend" / "Resume" control button, or just using the "Suspend All Debug sessions" and the "Resume All Debug sessions" buttons.

Xw	kspace - Develop - Ipcxpresso54102_multicore_examples_hello_world_cm0plus/source/hello_world_core1.c - MCUXpresso IDE	×
File	Edit Source Refactor Navigate Search Project Run FreeRTOS Window Help	
1		
8	†¢ Debug ⊠	
	Ipcxpresso54102_multicore_examples_hello_world_cm4 Debug [C/C++ (NXP Semiconductors) MCU Application]	
E.	Ipcxpresso54102_multicore_examples_hello_world_cm4.axf [LPC54102J512 (cortex-m4)]	
1010	per Thread #1 1 (Stopped) (Running)	
0101	🚽 arm-none-eabi-gdb (7.12.0.20161204)	
	Ipcxpresso54102_multicore_examples_hello_world_cm0plus Debug [C/C++ (NXP Semiconductors) MCU Application]	
****	a 😸 [pcxpresso54102_multicore_examples_hello_world_cm0plus.axf [LPC54102J512 (cortex-m0plus)]	
8	Thread #11 (Stopped) (Running)	
U	🚽 arm-none-eabi-gdb (7.12.0.20161204)	
(×)=		
(x)=		$ \rightarrow$
•	Welcome	
말해	63@ /*!	
<u> </u>	64 * @brief Main function	
	65 */	
	68 uint32 t startupData, i;	
	69	
	70 /* Define the init structure for the output LED pin*/	
	71 gpio_pin_config_t led_config = {	
	72 RGPI0_Digitaloutput, 0,	
	75 /* Initialize MCMGR before calling its API */	
	76 MCMGR_Init();	
	78 /* Get the startup data */	Ξ
	numok_detstartupbata(kntrigk_toret, astartupbata);	
	/* Make a noticable delay after the reset */	
	82 /* Use startup parameter from the master core */	
	83 for (i = 0; i < startupData; i++)	
	84 delay();	
	85 /* Thit board bandware */	
	87 /* enable clock for GPTO */	
	88 CLOCK EnableClock(kCLOCK Gpio0):	Ŧ
	٠ ( ا	

Figure 43. Synchronous suspension/resumption of both cores using the multiple selection of debug sessions and "Suspend"/"Resume" controls



# Figure 44. Synchronous suspension/resumption of both cores using the "Suspend All Debug sessions" and the "Resume All Debug sessions" controls

## Appendix A How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

1. Linux: The serial port can be determined by running the following command after the USB Serial is connected to the host:

\$ dmesg | grep "ttyUSB" [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0 [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1

There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. Windows: To determine the COM port open Device Manager in the Windows operating system. Click on the Start menu and type "Device Manager" in the search bar.

Control Panel (3)
🚔 Device Manager
View devices and printer
Update device View and update your hardware's settings and drivers
Pictures (9)
Companies.inc
hut.inc
PTPStillImageTables.inc
VIDs_PIDs.TXT
SCSI_CDB_RcvCpyRslts.inc
SCSI_CDB_SPC.inc
hci_command_table.inc
RNDIS_OID.inc
CDCRequests.inc
Files (1)
ialog_settings.xml
₽ See more results
Device Manager × Shut down

### Figure A-1. Device Manager

- 3. In the Device Manager, expand the "Ports (COM & LPT)" section to view the available ports. The COM port names will be different for all the NXP boards.
  - a. LPC-Link2



### Figure A-2. LPC-Link2

## Appendix B Updating debugger firmware

## **B.1 Updating LPCXpresso board firmware**

The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScrypt. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

### NOTE

If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking the "Debug" button). Using DFU mode ensures most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the LPCScrypt utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScrypt user guide (www.nxp.com/lpcutilities, select LPCScrypt, then select documentation tab).

- 1. Install the LPCScript utility.
- 2. Unplug the board's USB cable.
- 3. Make the DFU link (install the jumper labelled DFUlink).
- 4. Connect the probe to the host via USB (use Link USB connector).
- 5. Open a command shell and call the appropriate script located in the LPCScrypt installation directory (<LPCScrypt install dir>).
  - a. To program CMSIS-DAP debug firmware: <LPCScrypt install dir>/scripts/program\_CMSIS
  - b. To program J-Link debug firmware: <LPCScrypt install dir>/scripts/program\_JLINK
- 6. Remove DFU link (remove the jumper installed in step 3).
- 7. Re-power the board by removing the USB cable and plugging it again.

#### How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP. the NXP logo. NXP SECURE CONNECTIONS FOR A SMARTER WORLD. COOLFLUX. EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.

Document Number MCUXSDKLPC5410XGSUG Revision 0, 08/2019



