UM11153

NTAG SmartSensor getting started: A guide to start developing using an NHS31xx

Rev. 2.1 — 6 February 2020 User manual

Document information

Information	Content
Keywords	NHS31xx, Starter kit, SDK, quick start guide
Abstract	A concise guide describing the NHS31xx SW SDK setup process using the LPCXpresso development environment and its features when using the NXP provided NTAG SmartSensor development boards.

Revision history

Revision history		
Rev	**Date**	**Description**
2.1	20200106	Update for SDK 12.4
Modifications:	• Minor refresh of contents	
2.0	20190329	Update for SDK 12
Modifications:	• Major format update and refresh of contents	
1.7	20180615	Update for SDK 11.2
1.6	20180205	Update for SDK 11.1
1.5	20171011	Update for SDK 11 and LPCXpresso 8.2.2
1.4	20170220	Update for SDK 10
1.3	20170113	Update for SDK 9
1.2	20160905	Update for SDK 8.1
1.1	20160628	Update for SDK 8 and LPCXpresso 8.1.4
1.0	20160122	Update for SDK 6 and LPCXpresso 8.0.0
0.1	20150616	Initial version

1 Introduction

1.1 Document scope

The NHS31xx series is a family of ICs targeting vertical markets such as, but not limited to, smart logistics, industry 4.0, personal healthcare, and therapy compliance. The chips combine low power and flexibility with an ARM Cortex-M0+. Communication is typically done using the built-in NFC interface or wired using I^{2}C or SPI. All chips have been designed with the lowest system cost in mind and provide simple interfaces to sensors such as the built-in accurate temperature sensor.

Each NHS product has a specific value proposition and comes with its own use cases and reference design. This document describes how to evaluate our offering: how to set up, compile and flash the firmware contained in the SDK using the released boards. In this way, a firmware engineer can start developing using an NHS31xx.

Figure 1. Backside of an NTAG SmartSensor board.

Note: *This document is applicable to any setup featuring an NHS31xx, regardless of the demo PCB the IC is mounted on.*

1.2 Supported environments

- The LPCXpresso IDE is the only supported environment for firmware development. The MCUXpresso IDE is the successor of the LPCXpresso IDE, but is currently not supported.
- The NHS31xx SDKs are developed and tested under Windows 10 and macOS Catalina.
  The NHS31xx SDKs are known to be working on Linux distributions (such as Arch and Debian/Ubuntu), but no support is currently given.

1.3 Contact

- Business: nhs-info@nxp.com
- Technical: nhs-support@nxp.com

2 Tooling

2.1 LPCXpresso IDE

Before installation of the LPCXpresso IDE, ensure that the development host computer meets the requirements as listed in:

https://www.nxp.com/docs/en/user-guide/LPCXpresso_IDE_Installation_Guide.pdf, *Section 1 Host Computer Requirements*

2.1.1 Installation

- Download the LPCXpresso IDE v8.2.2. A direct download link can be found in the tools folder of the SDK contents.
- Install.

Note: *Use the default location to install the LPCXpresso IDE to.*

More information

https://www.nxp.com/docs/en/user-guide/LPCXpresso_IDE_Installation_Guide.pdf, Section 2 Installation.

2.1.2 Registration

Initially the LPCXpresso IDE runs with a default Unregistered license. The current license is listed on the Welcome page that is opened by default after each restart and can be checked via Help > Display License Type.

Activate with a Free Edition or Pro Edition license. The Free Edition is sufficient for all NHS31xx development. All features of the LPCXpresso IDE are available and the debug download limit of 256 kB is not hit, because any NHS31xx IC has 32 kB Flash.

More information

https://www.nxp.com/docs/en/user-guide/LPCXpresso_IDE_Installation_Guide.pdf:

- Section 3 Activation
- Section 2.5 Activating your product (LPCXpresso Free Edition)
- Section 2.6 Activating your product (LPCXpresso Pro Edition)

Note: *The automatic display of the Welcome page at start-up can be suppressed by clearing the checkbox at Window > Preferences > LPCXpresso > General > Show welcome view. This option depends on the chosen workspace.*

2.2 NHS31xx plugin

As a prerequisite, the NHS31xx plugin must be installed before the SDK can be used.

The LPCXpresso IDE supports all commercially available Cortex-M/ARM7/ARM9 LPC MCUs. However, to support the NHS31xx chips, a dedicated plugin must be installed on top of an existing installation of the LPCXpresso IDE.

With the plugin installed, the code can be compiled for the correct CPU core and corresponding memories. And while debugging, the peripheral registers and the different fields are available for inspection and modification. In addition, the New Project wizard is expanded with support for the NHS31xx SW framework and corresponding chips.

Warning:

Install the LPCXpresso IDE in a path where you are sure that you have sufficient rights or launch the LPCXpresso executable with administrative rights (Windows) or root/ superuser access (macOS) before installing this plugin. When you do not have enough write rights the installation fails. However, the installation fail is not reported. The failure only surfaces when you are trying to establish a debug session with an NHS31xx IC, with a confusing error description.

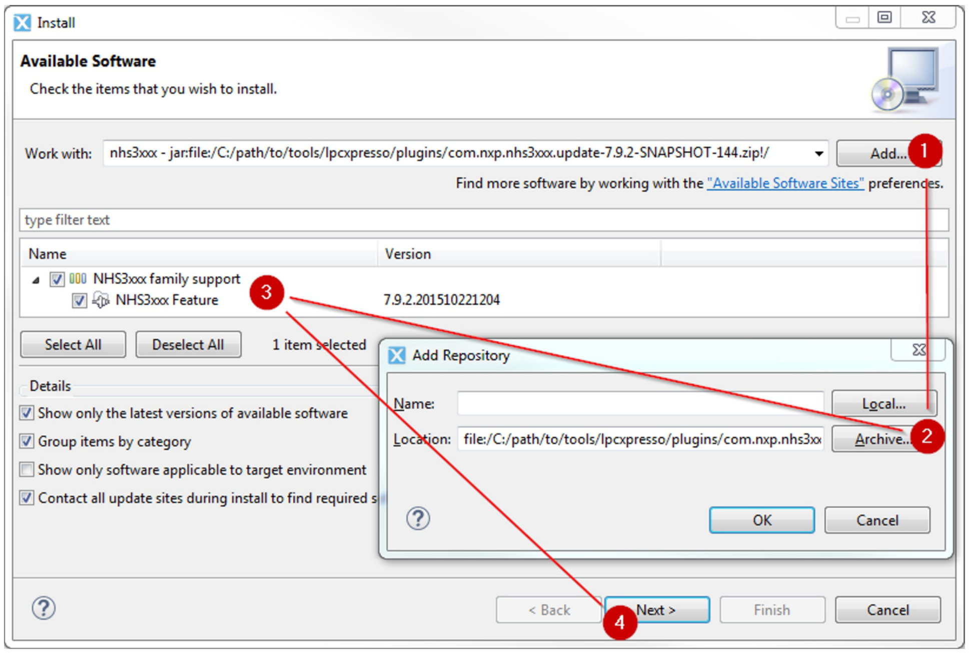
On Windows:

1. Open Windows Explorer and navigate to
   `<LPCXPresso install path>\lpcxpresso\lpcxpresso.exe`
2. Right-click on the executable and choose Run as administrator.

On macOS:

1. Open the Terminal app and run
   `sudo /Applications/lpcxpresso_8.2.2_650/lpcxpresso/ lpcxpresso.app/Contents/MacOS/lpcxpresso`

Installation steps:



Work with: nhs3xxx - jar:file:/C:/path/to/tools/lpcxpresso/plugins/com.nxp.nhs3xxx.update-7.9.2-SNAPSHOT-144.zip!/

Name	Version
☑ NHS3xxx family support	
☑ NHS3xxx Feature	7.9.2.201510221204

☑ Show only the latest versions of available software
☑ Group items by category
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required s

Name:
Location: file:/C:/path/to/tools/lpcxpresso/plugins/com.nxp.nhs3xx

aaa-031847

Figure 2. NHS31xx plugin installation

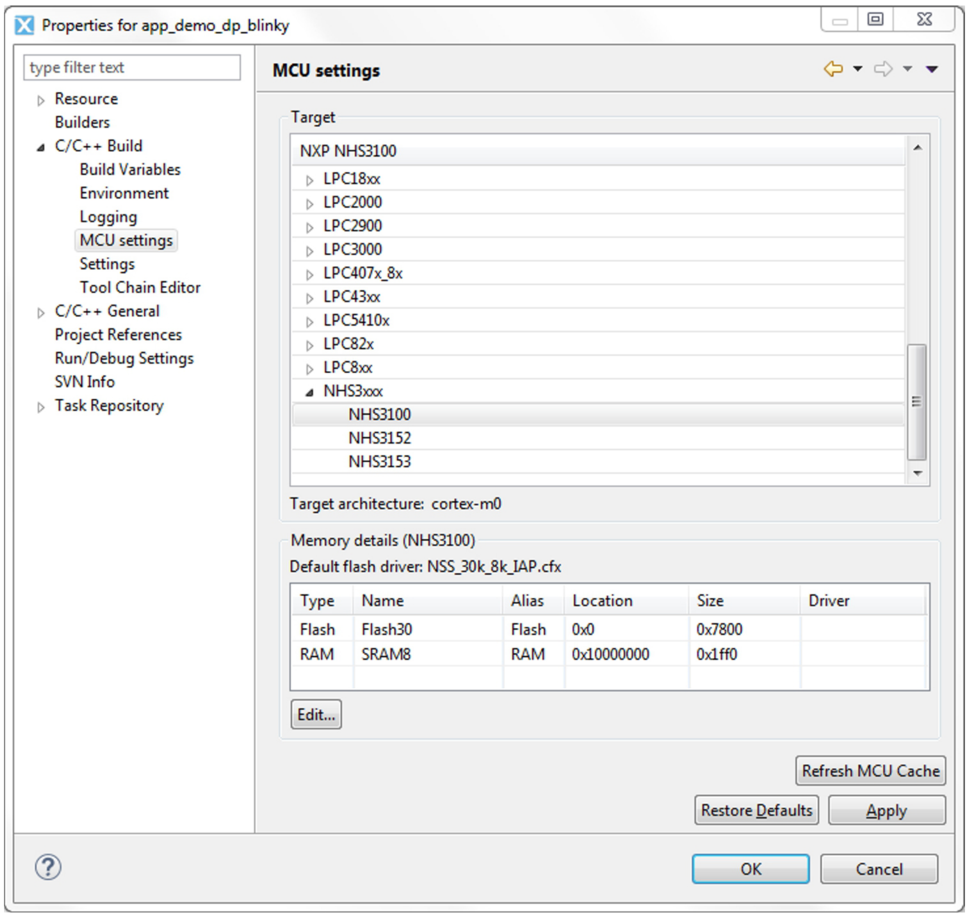
1. Help > Install New Software…
2. Locate the plugin *com.nxp.nhs3xxx.update-xxxxx.zip* in the SDK release under the *<SDK>/tools/lpcxpresso/plugins* folder by following these steps also outlined in Figure 2 above:
   - (1) Add…
   - (2) Archive… as an Available Software Site
   - (3) NHS3xxx Feature
     (4) Click Next

3. Finish the installation by accepting the license agreement. A warning dialog and a request to restart the LPCXpresso IDE pops up. Click OK and Yes where appropriate; when the LPCXpresso IDE starts up again, full NHS31xx support is available in your installation.

Note: *You can speed up the installation process by unchecking the option "Contact all update sites during install to find required software". The plugin does not depend on other plugins that must be installed up front.*

The plugin expands the LPCXpresso IDE with several NHS-specific features. A short overview is given in the sections below.

2.2.1 Chip support



NXP NHS3100
LPC18xx
LPC2000
LPC2900
LPC3000
LPC407x_8x
LPC43xx
LPC5410x
LPC82x
LPC8xx
NHS3xxx
NHS3100
NHS3152
NHS3153

Target architecture: cortex-m0

Memory details (NHS3100)
Default flash driver: NSS_30k_8k_IAP.cfx

Type	Name	Alias	Location	Size	Driver
Flash	Flash30	Flash	0x0	0x7800	
RAM	SRAM8	RAM	0x10000000	0x1ff0	

aaa-031855

Figure 3. After installing the NHS31xx plugin, support for several NHS ICs become available

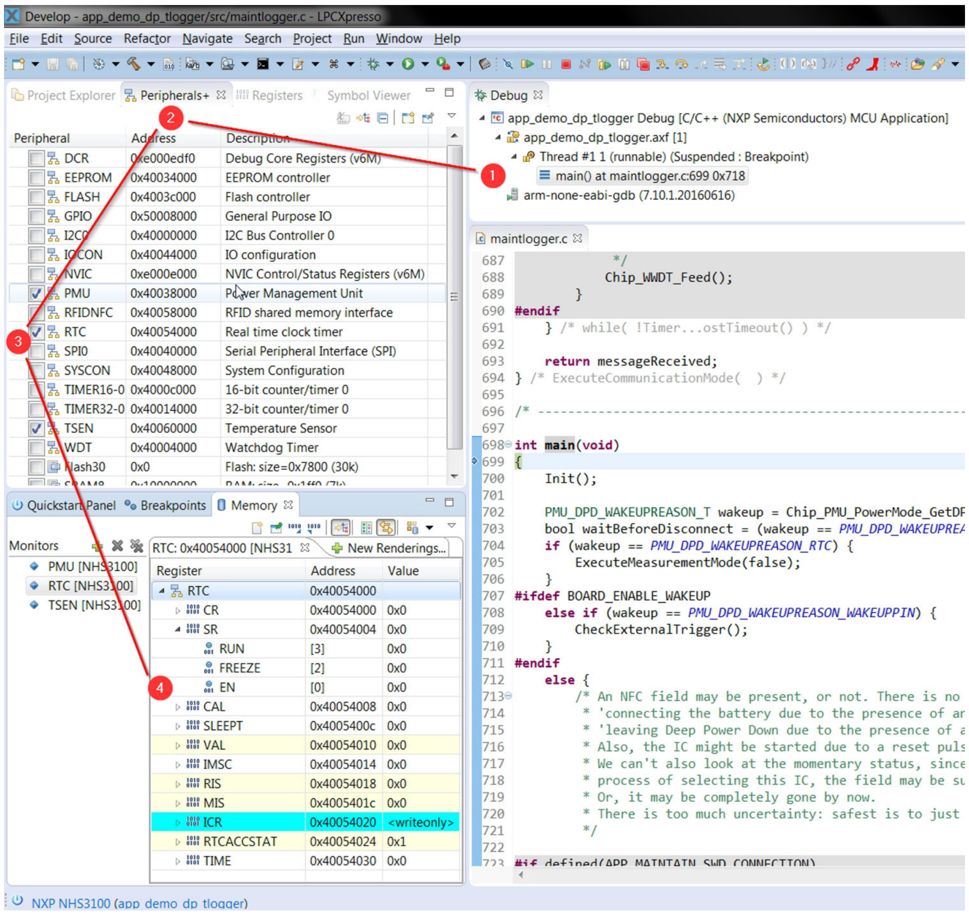
The chip for which a project is built can be checked or modified. To check or modify it, go to Project > Properties > C/C++ Build > MCU Settings (see Figure 3). With the plugin installed, the targets NHS3100 and NHS3152 have now become available.

Note: *After updating your installation with a newer version of the NHS31xx plugin, explicitly refresh the list of supported chips for the LPCXpresso. The list can be refreshed via the context menu of the project Tools > Refresh MCU cache.*

2.2.2 Register support

The NHS plugin provides direct access to all peripheral registers together with the fields and their descriptions.

Figure 4 shows the steps:



Peripheral	Address	Description
☐ DCR	0xe000edf0	Debug Core Registers (v6M)
☐ EEPROM	0x40034000	EEPROM controller
☐ FLASH	0x4003c000	Flash controller
☐ GPIO	0x50008000	General Purpose IO
☐ I2C0	0x40000000	I2C Bus Controller 0
☐ IOCON	0x40044000	IO configuration
☐ NVIC	0xe000e000	NVIC Control/Status Registers (v6M)
✓ PMU	0x40038000	Power Management Unit
☐ RFIDNFC	0x40058000	RFID shared memory interface
✓ RTC	0x40054000	Real time clock timer
☐ SPI0	0x40040000	Serial Peripheral Interface (SPI)
☐ SYSCON	0x40048000	System Configuration
☐ TIMER16-0	0x4000c000	16-bit counter/timer 0
☐ TIMER32-0	0x40014000	32-bit counter/timer 0
✓ TSEN	0x40060000	Temperature Sensor
☐ WDT	0x40004000	Watchdog Timer
☐ Flash30	0x0	Flash: size=0x7800 (30k)

```
app_demo_dp_tlogger Debug [C/C++ (NXP Semiconductors) MCU Application]
  app_demo_dp_tlogger.axf [1]
    Thread #1 1 (runnable) (Suspended : Breakpoint)
      main() at maintlogger.c:699 0x718
  arm-none-eabi-gdb (7.10.1.20160616)
```

```
                */
            Chip_WWDT_Feed();
        }
#endif
    } /* while( !Timer...ostTimeout() ) */

    return messageReceived;
} /* ExecuteCommunicationMode(  ) */

/* ------------------------------------------------------------

int main(void)
{
    Init();

    PMU_DPD_WAKEUPREASON_T wakeup = Chip_PMU_PowerMode_GetDP
    bool waitBeforeDisconnect = (wakeup == PMU_DPD_WAKEUPREA
    if (wakeup == PMU_DPD_WAKEUPREASON_RTC) {
        ExecuteMeasurementMode(false);
    }
#ifdef BOARD_ENABLE_WAKEUP
    else if (wakeup == PMU_DPD_WAKEUPREASON_WAKEUPPIN) {
        CheckExternalTrigger();
    }
#endif
    else {
        /* An NFC field may be present, or not. There is no
         * 'connecting the battery due to the presence of ar
         * 'leaving Deep Power Down due to the presence of a
         * Also, the IC might be started due to a reset puls
         * We can't also look at the momentary status, since
         * process of selecting this IC, the field may be su
         * Or, it may be completely gone by now.
         * There is too much uncertainty: safest is to just
         */

#if defined(APP_MAINTAIN_SWD_CONNECTION)
```

Register	Address	Value
RTC	0x40054000	
CR	0x40054000	0x0
SR	0x40054004	0x0
RUN	[3]	0x0
FREEZE	[2]	0x0
EN	[0]	0x0
CAL	0x40054008	0x0
SLEEPT	0x4005400c	0x0
VAL	0x40054010	0x0
IMSC	0x40054014	0x0
RIS	0x40054018	0x0
MIS	0x4005401c	0x0
ICR	0x40054020	<writeonly>
RTCACCSTAT	0x40054024	0x1
TIME	0x40054030	0x0

Figure 4. The NHS31xx plugin adds support for all peripheral registers

1. Start a debug session.
2. If not already open, open the Peripheral view: Window > Show View > Other… > Debug> *Peripherals+*
3. Select the HW block(s) you want register access to.
4. Expand the registers of interest, and hover the mouse to get the description of the field as a tooltip. Writeable fields can be updated via the Memory view as well.

***Note:** The Peripherals and Memory views are only populated when a debug session is running.*

Warning:

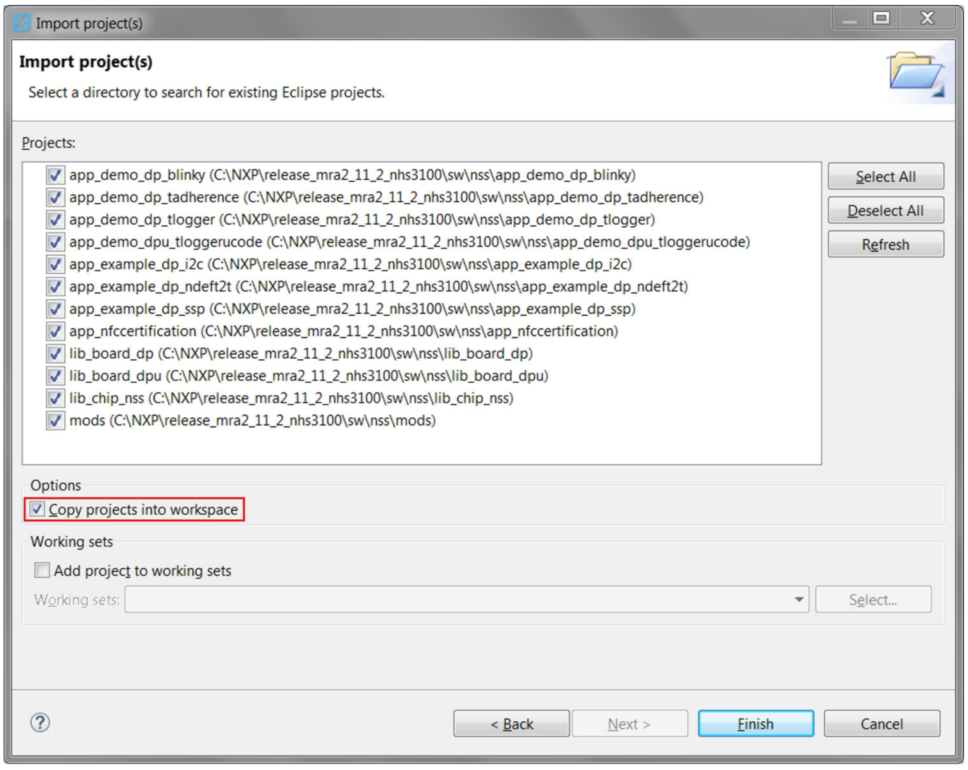
Viewing the registers may influence the behavior of the program. For example, the data register DR in the SSP HW block SPI0 is read each time the view is updated, which pops a value from the RX FIFO buffer. This value is then displayed within the LPCXpresso IDE but lost to the program under debug.

2.3 SDK

With the plugin in place, the SDK can be used by importing all the provided projects into a workspace. The LPCXpresso IDE-specific Quickstart panel can be used to import everything easily. For a short description of Quickstart panel, see the Section 4.1.1.

Create a new workspace or reuse your existing workspace and import the projects contained in the SDK release.

1. In the Quickstart Panel view, click Import project(s).
2. Fill in the path (including the filename) to the compressed SDK contents next to Archive or fill in the path to the decompressed SDK contents next to Root directory and click Next.
3. Select the projects found. Best is to import all projects found, including the non-application projects: each application project depends on the chip library, on one board library, and on multiple modules grouped in the mods container project. To be able to build successfully, all sources must be available in the same workspace.



Import project(s)

Import project(s)

Select a directory to search for existing Eclipse projects.

Projects:

- ✓ app_demo_dp_blinky (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_demo_dp_blinky)
- ✓ app_demo_dp_tadherence (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_demo_dp_tadherence)
- ✓ app_demo_dp_tlogger (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_demo_dp_tlogger)
- ✓ app_demo_dpu_tloggerucode (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_demo_dpu_tloggerucode)
- ✓ app_example_dp_i2c (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_example_dp_i2c)
- ✓ app_example_dp_ndeft2t (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_example_dp_ndeft2t)
- ✓ app_example_dp_ssp (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_example_dp_ssp)
- ✓ app_nfccertification (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\app_nfccertification)
- ✓ lib_board_dp (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\lib_board_dp)
- ✓ lib_board_dpu (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\lib_board_dpu)
- ✓ lib_chip_nss (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\lib_chip_nss)
- ✓ mods (C:\NXP\release_mra2_11_2_nhs3100\sw\nss\mods)

Options

✓ Copy projects into workspace

Working sets

☐ Add project to working sets

Working sets:

aaa-031857

Figure 5. Importing existing projects

Note:

Pay special attention to the option Copy projects into workspace. When importing from a Project archive (zip), it is grayed out as only a copy can be performed. However, when importing from a Project directory (unpacked), you can choose.

- *If checked, the files are copied into your LPCXpresso workspace and the original files are left untouched.*
- *If unchecked, the workspace only contains a reference to the location you provided and all changes are done in-place.*

The option is selected by default, but both options yield equal results. Choose whatever best suits your style. Be careful not to mix styles. Application projects using modules from the mods project use relative references. If the application is imported by copying while the mods project is imported by referencing or vice versa, building fails with errors, like `chip.h:35:29: fatal error: startup/startup.h: No such file or directory.`

Note:

The current workspace can be retrieved by hovering over the hyperlinked text in the bottom status bar or by copying the default value from the dialog that pops up after File > Switch Workspace > Other... > Workspace.

After importing the projects, all firmware content from the SDK becomes available from within the LPCXpresso IDE.

Note: *A description and guide regarding the host tools (mobile and PC-based) and their development environment is outside the scope of this document.*

2.4 Flash Magic

Flash Magic is a third-party PC tool for programming FLASH-based microcontrollers from NXP Semiconductors using Intel HEX files via a serial protocol. It can be freely used during development or for programming small batches. Using Flash Magic on a production line is also possible, but requires a purchase.

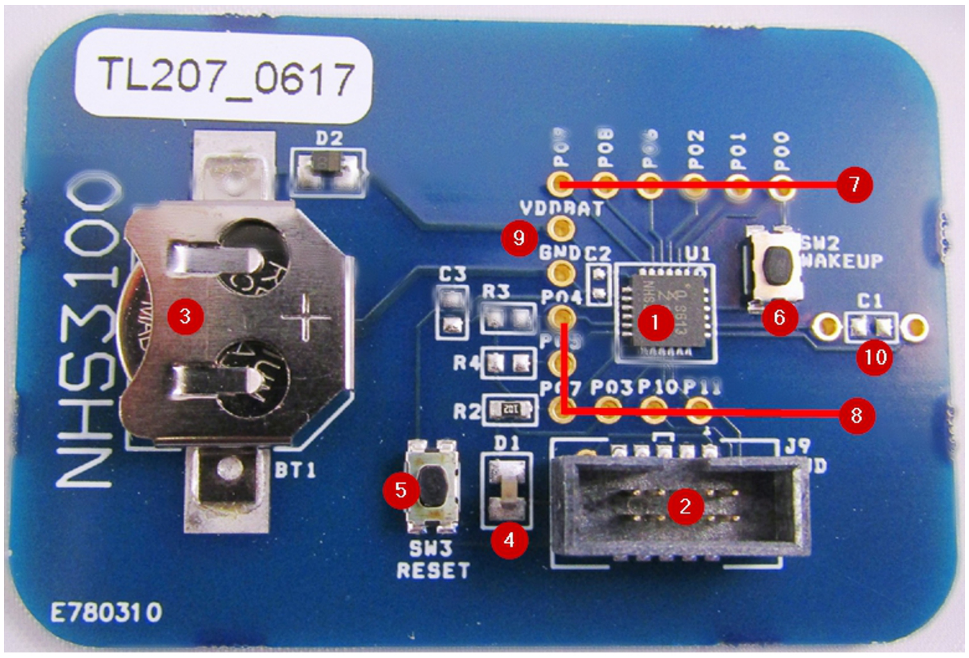
The use of this tool is not enforced, but it helps to program ICs quickly using prebuilt firmware images.

Further information can be found in the application note "Overview of supported methods for firmware flashing on NHS31xx ICs" (AN12328), which is part of the SDK.

3 Boards

3.1 Demo PCB

For each IC, several boards have been developed and released. These boards can be used for demonstration and SW development purposes. Most boards feature the same characteristics. As an example, consider the NHS3100 temperature logger demo PCB (Figure 6).



aaa-031858

Figure 6. NHS3100 temperature logger demo PCB

Components:

- (1) An NHS3100 IC in an HVQFN24 package (U1)
- (2) An SWD connector (J9)
- (3) A coin cell holder for standalone operations (BT1)
- (4) One SW controllable LED (D1)
- (5) A tactile switch (SW3) connected to the RESETN pin
- (6) A tactile switch (SW2) connected to the WAKEUP pin
- Through-holes for easy access to:
  - (7) and (8) All PIOs of the IC (P0x)
  - (9) GND and VDDBAT
  - (10) Antenna coil connections LA and LB, connected with the NFC antenna on the back (not visible)

The demo PCB is ready for use upon arrival.

***Note:** After powering the demo PCB by inserting a battery or by connecting it to a prepared LPC-Link2 board (see Section 3.2), the NHS IC does not become active. Only after briefly providing an NFC field near the antenna or when the RESET button is pushed, the IC wakes up and starts executing the ARM program stored in Flash.*

Note: *The suggested coin cell battery type to use is CR1225. It can be ordered internationally at e.g. https://www.newark.com/w/c/batteries-chargers/prl/results?st=CR1225.*

Note: *The Demo PCB can be powered via the JTAG connector or a nearby NFC field.*

Note: *The diode D2 protects the circuits against reverse battery polarity and charging the battery via the SWD reference. However, there is a chance that the battery itself is shorted, depleting it. Ensure that the battery is inserted with the positive side facing up.*

3.2 LPC-Link2

LPC-Link2 is a standalone debug adapter that can be configured to support various development tools and IDEs with downloadable firmware. It is compatible with the LPCXpresso IDE and the NHS31xx SDK using the CMSIS-DAP debugging protocol.

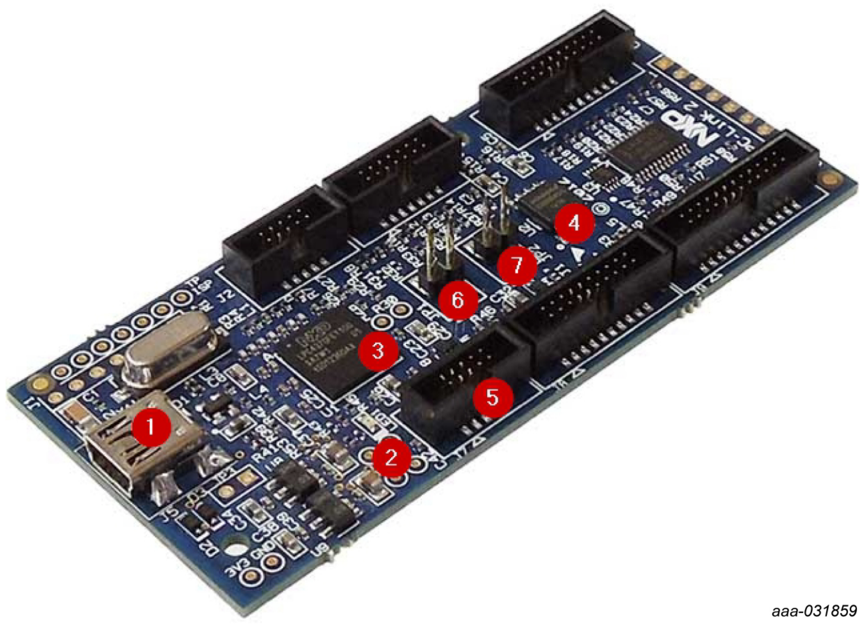


Figure 7. LPC-Link2 board without jumpers over JP1 and JP2

A few components of interest:

- (1) A mini USB connector
- (2) One LED signifying connection and communication with the PC
- (3) An LPC4370 to execute the debugging protocol firmware
- (4) An SPIFI flash which may be used to store the debugging protocol firmware
- (5) A JTAG connector compatible with the NHS demo PCB
- (6) JP1 to control the use of the SPIFI flash.
  When mounted, it selects the SWD firmware in the onboard SPIFI flash. When missing, the LPCXpresso IDE soft loads the SWD software via DFU.
- (7) JP2 to control whether the Device Under Test is powered via the JTAG connector.
  When mounted, the LPC-Link2 provides 3.3 V supply voltage on the VTREF pin of the SWD and powers the NHS31xx.

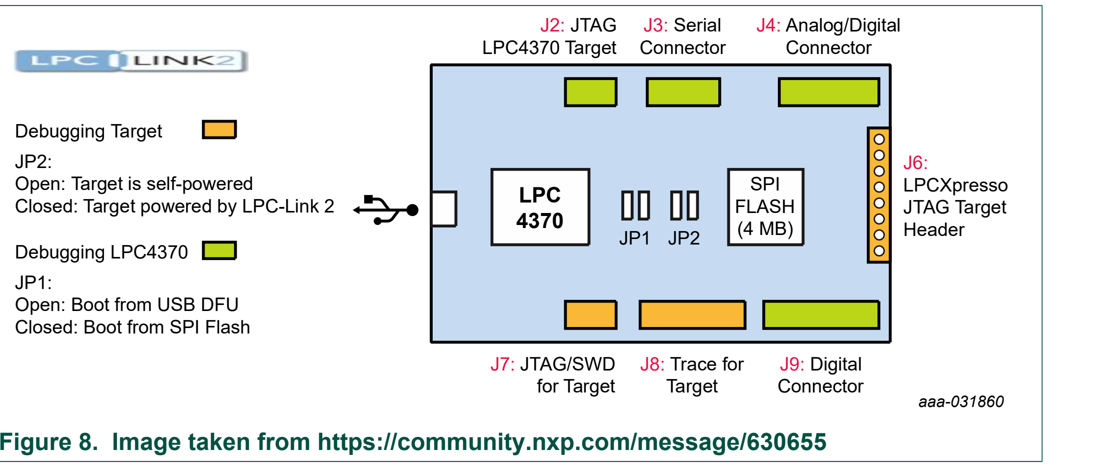


Figure 8. Image taken from https://community.nxp.com/message/630655

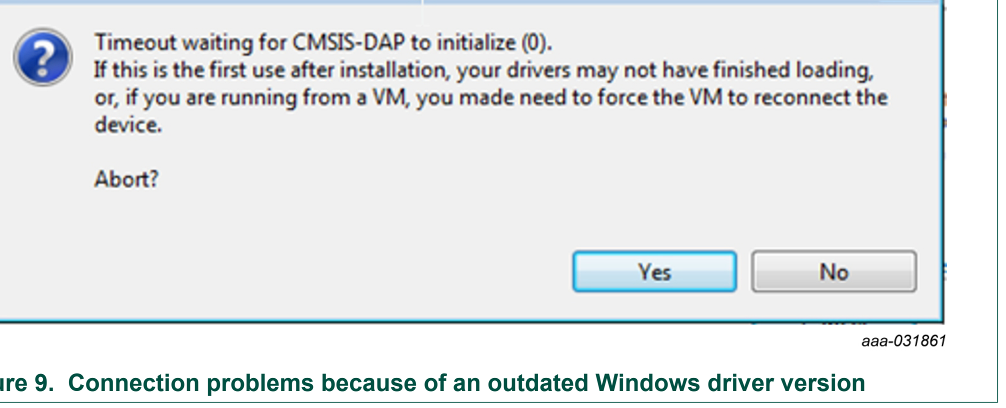
More information

http://www.nxp.com/demoboard/OM13054.html

The LPC-Link2 board is ready for use upon arrival.

Warning

On some Windows PCs, an old version of the LPC-Link2 driver (1.0.0.0) is automatically selected, even though the installation procedure was followed correctly. The selection of an old version of the LPC-Link2 driver goes unnoticed until a debug session is attempted. No LPC-Link2 board can be put to use. To fix this issue, uninstall and remove the old driver via the Windows Device Manager. After installing the new device drivers, driver version 2.0.0.0 is reported.


CMSIS-DAP

Timeout waiting for CMSIS-DAP to initialize (0).
If this is the first use after installation, your drivers may not have finished loading, or, if you are running from a VM, you made need to force the VM to reconnect the device.

Abort?

Figure 9. Connection problems because of an outdated Windows driver version

More information

https://community.nxp.com/message/630660

3.2.1 JP1

By default, no jumper is placed over JP1.

- JP1 unmounted
  The PC host downloads the CMSIS-DAP debugging protocol firmware[1] to the RAM of the LPC-Link2 board each time it is power cycled.
- JP1 mounted
  Optionally, if the protocol firmware is stored in the SPIFI flash on the LPC-Link2 board, a jumper can be placed over JP1. With it, the LPC-Link2 board is immediately ready for use after powering up. The result is a modest speed increase, which may become important when there is only a very small time window in which the SWD lines are active.

To program the SPIFI flash on the LPC-Link2 board with the CMSIS-DAP debugging protocol firmware, download and install LPCScrypt. Ensure that the jumper is not placed over JP1 and launch the Program LPC-Link2 with CMSIS-DAP script (under Windows directly available from the Start Menu). Afterward, fit JP1 and power cycle the LPC-Link2 board.

Note: *The use of LPCScrypt is not required. Using it, a somewhat faster start-up cycle is gained while establishing a debug connection. When continually switching between debugging and using Flash Magic, remember to remove and refit the jumper over JP1.*

Non-bridged variant

By default, the debugging protocol includes extra bridge channels to provide extra functionality such as SWO Trace capture and UART VCOM port. This functionality is either not available through the LPC-Link2 board or not exposed by the NHS31xx IC.

To use the non-bridged variant with JP1 unmounted, select CMSIS-DAP (Non-bridged - Debug only) as LPC-Link 2 boot type under Window > Preferences > LPCXpresso > Redlink/LinkServer. This option depends on the workspace.

To use the non-bridged variant with JP2 mounted, use the CMSIS-DAP script from LPCScrypt with the `NB` argument: `<LPCScrypt install path>\scripts\program_CMSIS NB`

Download

A direct download link can be found in the tools folder of the SDK contents.

More information

LPC-Link2 support: https://community.nxp.com/message/630655

LPC-Link2 debug probe firmware programming:
https://community.nxp.com/message/630661

LPC-Link2 debug probe script options: Section 7.2 in
https://www.nxp.com/docs/en/supporting-information/Debug_Probe_Firmware_Programming.pdf
https://www.nxp.com/pages/:LPC-TIPS-LPCSCRYPT-LPC-LINK2

Configuring which LPC-Link2 firmware image to soft load:
https://community.nxp.com/thread/388968

1 CMSIS-DAP provides a standardized way to access the debug port of an ARM Cortex microcontroller via USB. In combination with a NHS31xx IC, it provides a connection from a development board to a debugger running on a host computer using serial wire debug (SWD) to the target device.

3.2.2 JP2

By default, no jumper is placed over JP2.

- JP2 unmounted
  The debugging target, the NHS31xx demo PCB, must be self-powered.
- JP2 mounted
  The NHS31xx demo PCB is powered by the LPC-Link2 board.

Note: *The battery and the jumper over JP2 can be fitted simultaneously.*

Warning:

Regardless whether JP2 is mounted, when testing the power-off state of the NHS31xx (VDDBAT switched open), the SWD connector must be removed as the SWD pins are driven high by the LPC-Link2 probe, interfering with the PMU of the NHS31xx.

3.3 Debug setup

Through the LPC-Link2 board, you have full SW debug capabilities on the NHS31xx demo PCB.

- Connect a flat cable with the JTAG connector on the LPC-Link2 board: J7 (1) and the Demo PCB: J9 (2).
- Connect a mini USB cable to the LPC-Link2 board and your PC (3).

Your setup should now look similar to Figure 10.

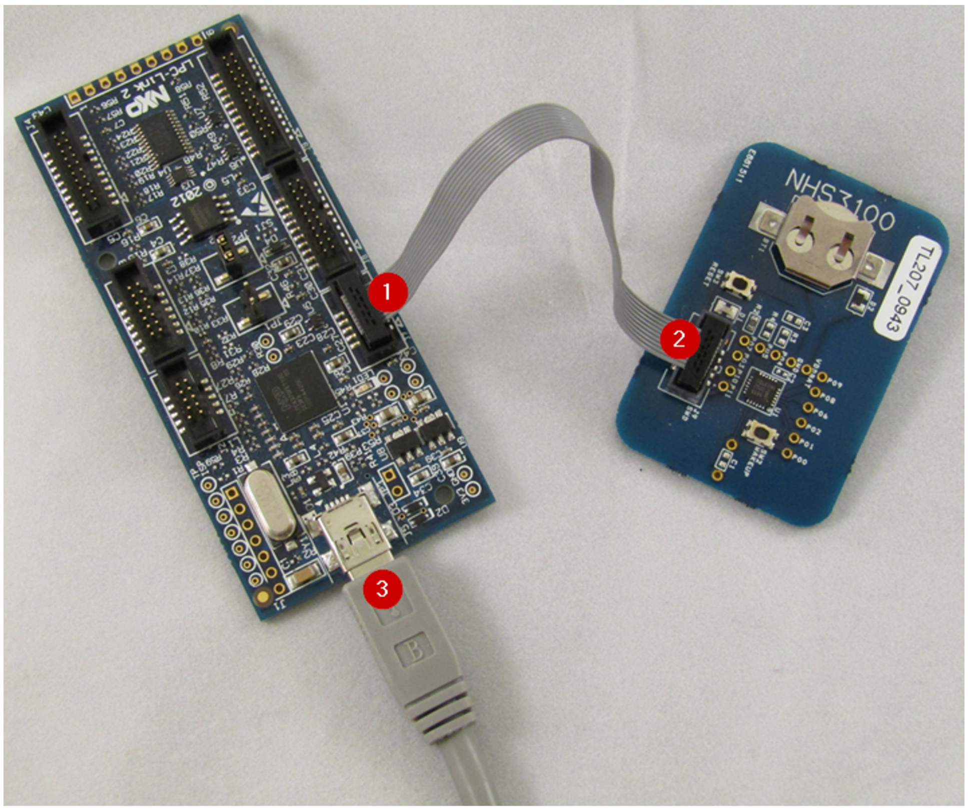


Figure 10. Demo PCB ready for debug using an LPC-Link2

4 Using LPCXpresso

Documentation about using the LPCXpresso environment can be found under Help > Help Contents. If you are new to Eclipse or the LPCXpresso IDE, read this documentation.

More information

https://www.nxp.com/docs/en/brochure/LPCXpresso_IDE_Flyer.pdf

https://www.nxp.com/docs/en/user-guide/LPCXpresso_IDE_User_Guide.pdf, section 2 LPCXpresso IDE Overview

4.1 Project Setup

4.1.1 Quickstart panel

The LPCXpresso IDE is built upon Eclipse and adds a Quickstart Panel view. In that view, the panel Start here provides quick links to existing functionality that is scattered throughout the various Eclipse menus.

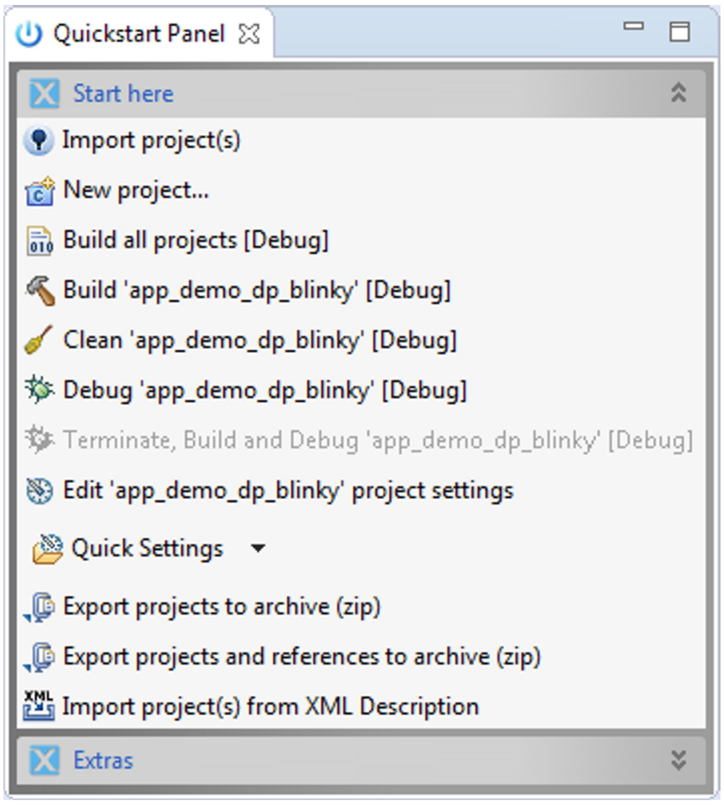

Quickstart Panel
Start here
Import project(s)
New project...
Build all projects [Debug]
Build 'app_demo_dp_blinky' [Debug]
Clean 'app_demo_dp_blinky' [Debug]
Debug 'app_demo_dp_blinky' [Debug]
Terminate, Build and Debug 'app_demo_dp_blinky' [Debug]
Edit 'app_demo_dp_blinky' project settings
Quick Settings
Export projects to archive (zip)
Export projects and references to archive (zip)
Import project(s) from XML Description
Extras
aaa-031863

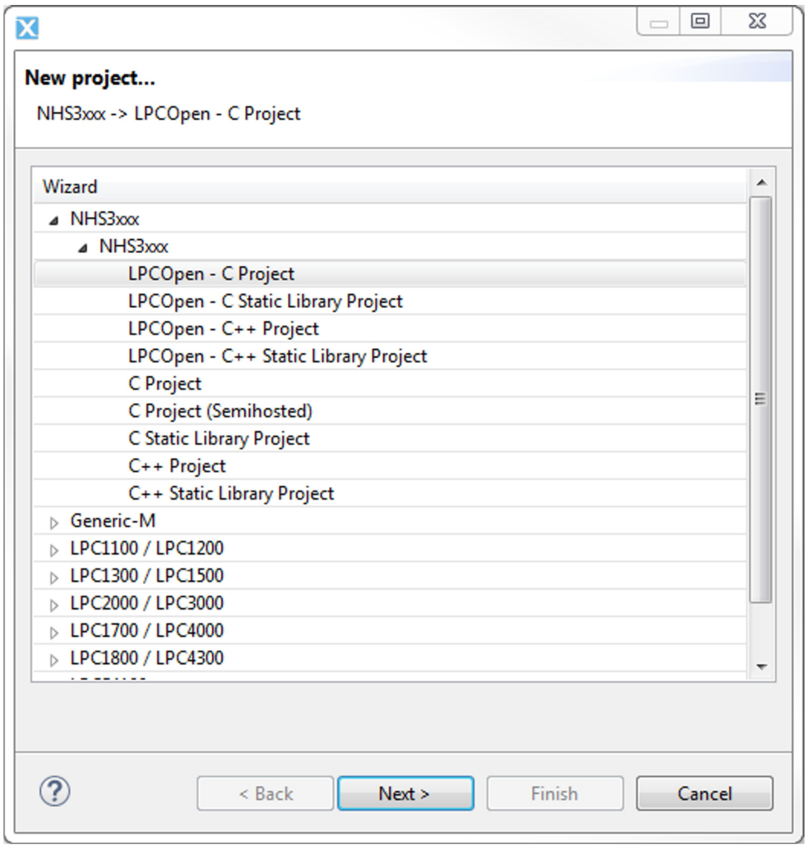
Figure 11. Quickstart panel groups often used commands for the selected project

If you feel more comfortable with the Eclipse way of offering, you can close this view, saving up some screen space for other views. The view can be found again via Windows > Show View > Other… > Quickstart > Quickstart Panel.

4.1.2 New project wizard

The New Project Wizard is the preferred way to create new projects quickly.

1. In the Quickstart Panel view, click New project…
2. Select NHS31xx > NHS31xx > LPCOpen – C Project (see Figure 12).



New project...

NHS3xxx -> LPCOpen - C Project

Wizard

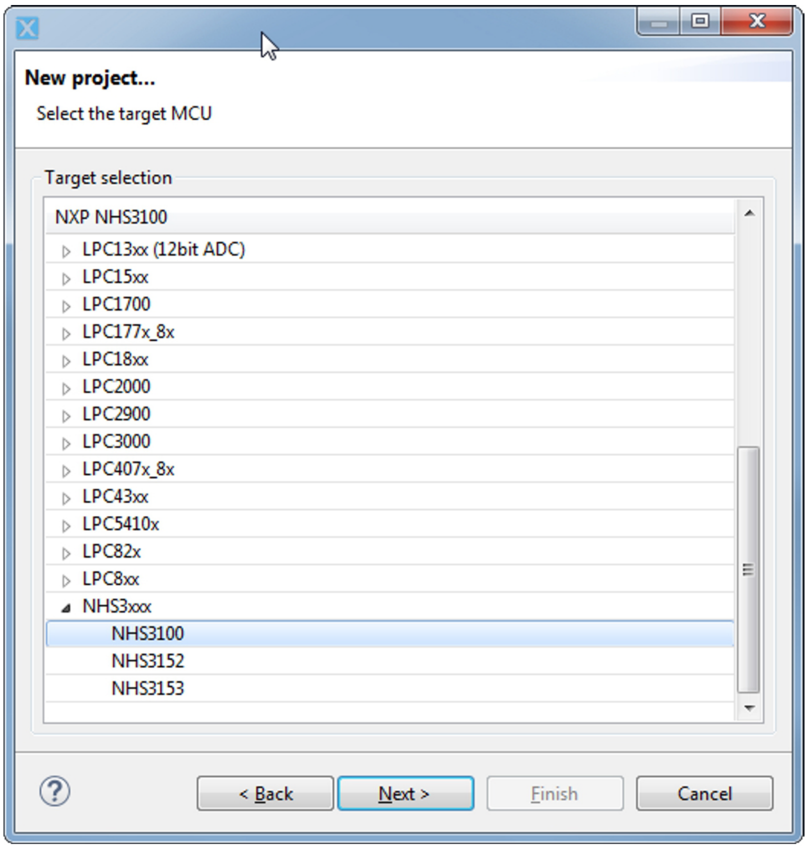
- NHS3xxx
  - NHS3xxx
    - LPCOpen - C Project
    - LPCOpen - C Static Library Project
    - LPCOpen - C++ Project
    - LPCOpen - C++ Static Library Project
    - C Project
    - C Project (Semihosted)
    - C Static Library Project
    - C++ Project
    - C++ Static Library Project
- Generic-M
- LPC1100 / LPC1200
- LPC1300 / LPC1500
- LPC2000 / LPC3000
- LPC1700 / LPC4000
- LPC1800 / LPC4300

aaa-031864

Figure 12. LPCOpen - C project is the sole supported project type

Note: For any NHS31xx IC, only the LPCOpen - C Project may be chosen. All other options presented have not been implemented.

3. Choose a project name and select the desired location.
4. Select the correct NHS31xx IC from the list (see Figure 13).



aaa-031865

Figure 13. The NHS31xx plugin expands the list of wizards to choose from

5. You can accept the default options presented in the screens that follow, except for the chip and board library. Specifically choose lib_chip_nss as the referenced LPCOpen Chip Library Project and select or type in the correct board - such as lib_board_dp - as the referenced LPCOpen Chip Board Project.

Note: *LPCXpresso IDE v8.2.2 uses the GNU Tools for ARM Embedded Processors v5.4.1, which means that the Default Compiler language dialect is gnu11.*

More information

https://gcc.gnu.org/gcc-5/

4.1.3 Import project

Any existing project can be imported. The procedure is the same as described in Section 2.3.

4.1.4 Mods

Code that does not belong in the Chip library or the Board library, but that is still usable for a multitude of projects, is grouped under the mods project. The mods project is a simple container project with no compilation settings enabled. The mods LPCXpresso project contains SW modules that can be reused across several other projects while supporting diversity (using `#define` precompilation flags). These modules may be included at any layer (chip, board, application) and the respective code is compiled by the project they are included in.

Using this approach, the SDK can offer additional reusable blocks of code while still adhering to the way of working of the LPCXpresso.

An application that requires the functionality provided by one of the reusable modules, must create a link to it (see Figure 14).

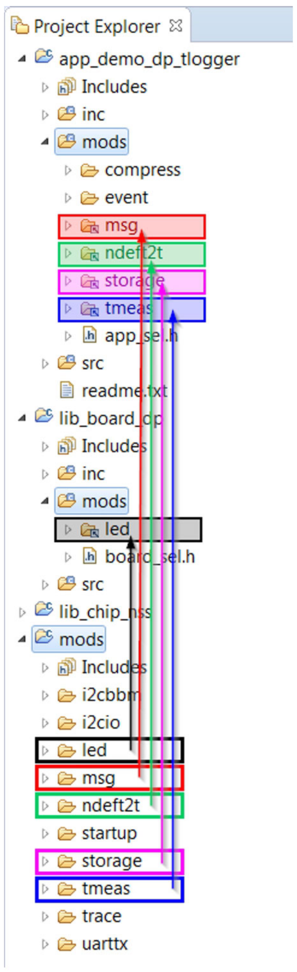
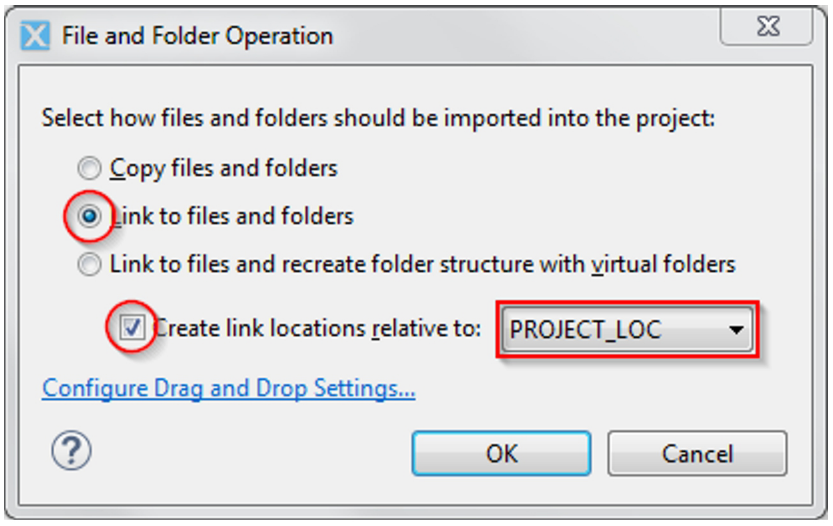


Figure 14. Code reuse by linking modules in other projects

In the LPCXpresso IDE, in the Project Explorer view:

1. Control-drag the desired reusable module, a child of the mods project, to the mods folder of the application project. Before releasing the mouse button, be sure that the `CTRL` key is down.


File and Folder Operation

Select how files and folders should be imported into the project:

( ) Copy files and folders
(●) Link to files and folders
( ) Link to files and recreate folder structure with virtual folders

[✓] Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

Figure 15. Creating a link to a reusable module

2. A window pops up. Ensure that Link to files and folders and Create link locations relative to: PROJECT_LOC are selected (see Figure 15).

The linked module is given a link icon overlay and its files are directly accessible within the application project. The module now compiles together with the rest of the application. A simple include statement `#include "modx/modx.h"` now gives you access to the API of the module.

After including one or more modules, check the documentation of each module to see which diversity settings to use. Each module provides reasonable defaults which you can override by adding `#define` precompilation flags in `app_sel.h`. By tweaking the module, you can include or exclude functionality and reduce the required code size.

***Note:** Alternatively, you can create a link via File > New > Other… > General > Folder. After clicking Next, select the mods folder of your project, click Advanced, and select Link to alternate location (Linked Folder). In the field below, you can build up the path to the module you want to reuse. Use the Browse… and Variables… buttons to build the path in a reusable way.*

4.2 Building

Building can be done in various ways:

- Using the menu Project
- Using the Quickstart Panel view
- Using the shortcut key combination `<CTRL>+B`
- Using the context menu of the project

No additional steps are necessary. After a successful import, all imported projects build without issues. First select a project, after which the Build command becomes active in the Quickstart Panel view, noting the selected project and the active build configuration between brackets.

Note:

Very rarely, running make may result in an error similar to:

```
0 [main] us 0 init_cheap: VirtualAlloc pointer is null, Win32
error 487
AllocationBase 0x0, BaseAddress 0x71110000, RegionSize
0x350000, State 0x10000
\msys\bin\make.exe: *** Couldn't reserve space for cygwin's
heap, Win32 error 0
```

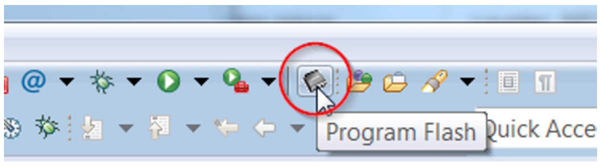
These errors are an occasional issue with the MSYS binaries that come with an LPCXpresso installation. If such an error occurs, you can change the DLL base address of the DLL `msys-1.0.dll` *which can be found under* `<install_dir>\lpcxpresso\msys\bin`*. Replace the file with the file that can be downloaded from https://community.nxp.com/message/630728. Closing or restarting the LPCXpresso IDE is not required.*

Note: *The changing of the DLL base address does not fix the problem. It only moves the DLL base address, which means that if this problem is not encountered, it may come up after replacing the DLL as described above. When you do not encounter this problem, do not apply this workaround.*

4.3 Flashing

By default, starting a debug session (see Section 4.4) also programs the flash automatically. But you can also flash any `.axf` file, `.elf`, or `.bin` file without starting a debug session:

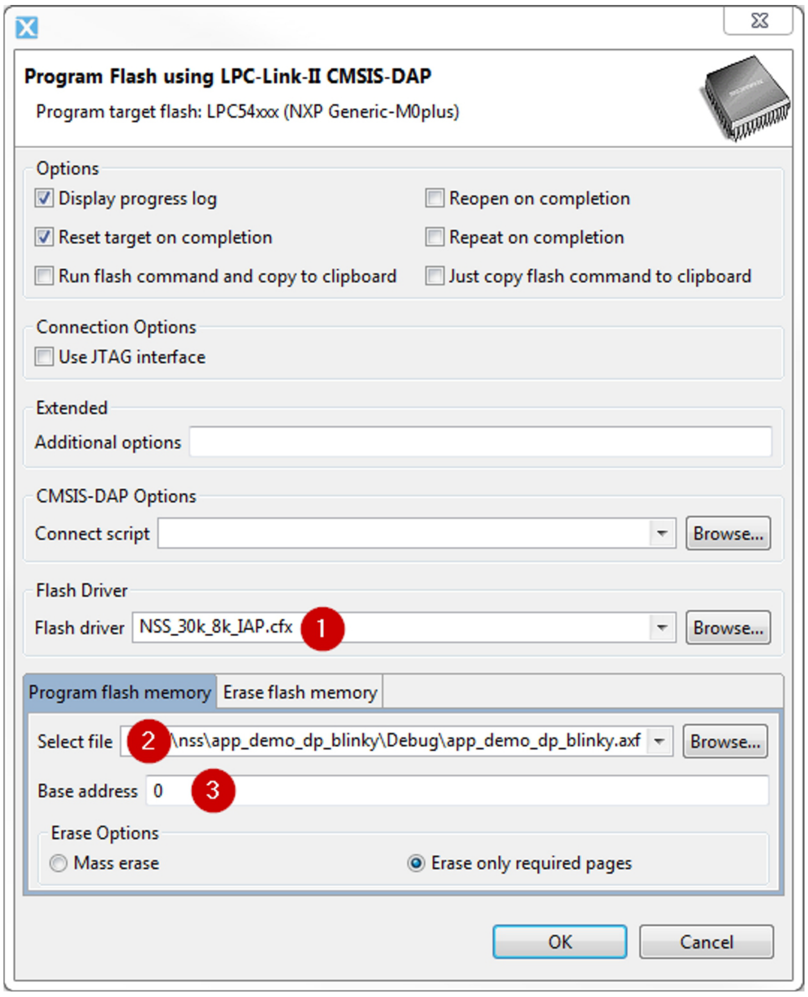
1. Make a hardware connection as outlined in Section 3.3.
2. Within the LPCXpresso IDE, select a compatible project. The Program Flash icon only becomes accessible after selecting a project, since some of the project settings are implicitly reused. Be sure to select a project that reuses the same MCU settings as the `.axf` file you want to flash.
3. Click the Program Flash icon in the toolbar (see Figure 16).



aaa-031868

Figure 16. Marking the location of the Program Flash button

4. In the dialog that pops up, verify these settings (see Figure 17).
   - (1) Flash driver: NSS_30k_8k_IAP.cfx
     This file was copied to the LPCXpresso installation directory under *<install path>/lpcxpresso/bin/Flash* during the installation of the NHS31xx plugin (see Section 2.2) and if the selected project matches your MCU, it is already correctly filled in here.
   - (2) Select file
     The `.axf` or `.bin` application file to flash. The Intel hex format - with a `.hex` extension - is not recognized.
   - (3) Base address



Program Flash using LPC-Link-II CMSIS-DAP

Program target flash: LPC54xxx (NXP Generic-M0plus)

Options

- [✓] Display progress log
- [ ] Reopen on completion
- [✓] Reset target on completion
- [ ] Repeat on completion
- [ ] Run flash command and copy to clipboard
- [ ] Just copy flash command to clipboard

Connection Options

- [ ] Use JTAG interface

Extended

Additional options:

CMSIS-DAP Options

Connect script:

Flash Driver

Flash driver: NSS_30k_8k_IAP.cfx (1)

Program flash memory | Erase flash memory

Select file: (2) \nss\app_demo_dp_blinky\Debug\app_demo_dp_blinky.axf

Base address: 0 (3)

Erase Options: Mass erase [✓ Erase only required pages]

aaa-031869

Figure 17. Program Flash dialog

After clicking OK, the flash is programmed. At the end, a dialog pops up with the log and the final result.

4.4 Debugging

You can use the full debugging features the LPCXpresso IDE offers. By default, starting a debug session also programs the flash automatically.

1. Make a proper HW connection as outlined in Section 3.
2. Within the LPCXpresso IDE, you can start a debug session in various ways:
   - Via the menu Run
   - Via the Quickstart Panel view
   - Using a shortcut key combination
   - Using the context menu of the project

Note: *The LPCXpresso IDE only allows one debugging session at a time. Even if the connection was broken, the session within the IDE must still be explicitly terminated before a new session can be successfully created.*

Warning:

When the firmware running in the NHS31xx IC decides to enter the deep power-down mode or the power-off mode, the SWD lines become inaccessible. Continuing or starting a debug session is then impossible. One possible way is to toggle the RESETN pin. It resets the IC and then starts a debug session in the LPCXpresso IDE, before the code changing the power mode has a chance to execute. Since this action is time-critical, this approach may be too impractical. Another option is to provide an NFC field near the NFC antenna. Depending on the code that has been flashed, you may then have sufficient time to launch a debugging session. A last way is to use the FlashMagic tool. Since it is much faster than the LPCXpresso IDE and triggers a RESET pulse prior to attempting to set up a debug session, it increases the chance to be able to halt the ARM core before the SWD lines become inaccessible.

To ensure that a software bug does not break the device by permanently disabling the SWD lines, add a debug code that implements a means to allow SWD access before entering the deep power-down mode or disconnecting the battery. Also ensure that the corresponding PIOs have been configured correctly for SWD functionality beforehand. Four examples of this approach have been implemented in the demo firmware app_demo_dp_tlogger:

- When using the define `APP_MAINTAIN_SWD_CONNECTION`, the deep power-down and power-off modes are entirely avoided while retaining the entire functionality.
- When the board has a provision for it (recognized using `BOARD_ENABLE_WAKEUP`), the application checks if the wake-up pin is pulled low at start-up (in `ResetISR`). It waits for as long as that pin is pulled low, giving ample time to connect via SWD and start a debugging session.
- When using the Debug build configuration, write an NDEF message on page 6 onwards, containing a MIME record with `08h 00h` as payload. It corresponds to a msg command with ID `MSG_ID_PREPAREDEBUG` and configures the SWD lines correctly, then pauses execution. The LPCXpresso IDE can then reflash the board, or connect a debug session without reflashing and continue execution while debugging. See the msg module documentation, available in the SDK under *<SDK>/docs/firmware.html*.
- If three conditions are met, the application waits for 3 seconds during deinitialization (`DeInit`):
  – The IC was not started by NFC, RTC, or the WAKE-UP pin.
  – No NDEF message exchange took place.
  – No measurement is due, i.e. the next mode is the power-off mode.

In practice, it means that to give you a reasonable time to establish an SWD session, the IC must be hard-reset.

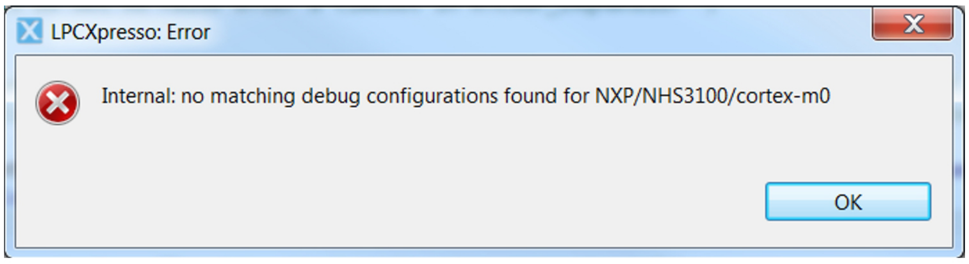
Note: *To learn more about the low-power modes, check the PMU documentation in the SDK (check for PMU_NSS CHIP: NSS Power Management Unit driver). There you can also find information about maintaining state information that persists even when restarting after leaving the deep power-down mode.*

Note: *To learn more about waking up from the deep power-down mode in a timely manner, check Example 4 in the RTC documentation in the SDK, under RTC_NSS CHIP: NSS Real-Time Clock driver.*

Note: *To learn more about the problems and possible workarounds that deep power-down mode and power-off mode can give during debugging, check SW Debug Considerations in the documentation in the SDK.*

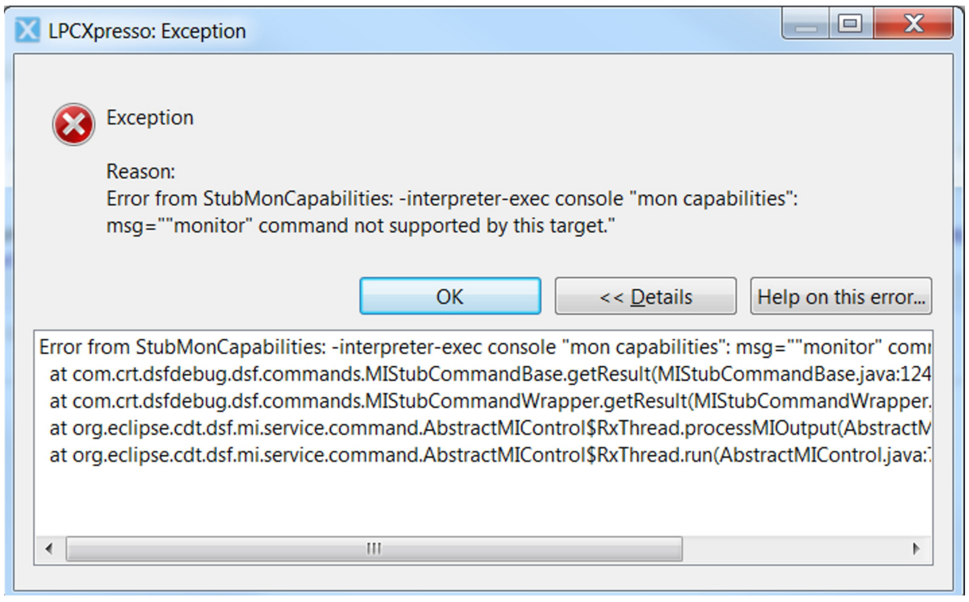
Warning:

On some Windows PCs, the errors "no matching debug configuration found for NXP/ NHS3100/cortex-m0, or ""monitor" command not supported by this target" may pop up. See the dialogs below. When the installation of the plugin was not successful, these errors occur. Reinstall the NHS31xx plugin using administrative rights. See also the warning in Section 2.2.


LPCXpresso: Error

Internal: no matching debug configurations found for NXP/NHS3100/cortex-m0

Figure 18. Debug initialization error (1)



Exception

Reason:
Error from StubMonCapabilities: -interpreter-exec console "mon capabilities": msg=""monitor" command not supported by this target."

```
Error from StubMonCapabilities: -interpreter-exec console "mon capabilities": msg=""monitor" comı
  at com.crt.dsfdebug.dsf.commands.MIStubCommandBase.getResult(MIStubCommandBase.java:124
  at com.crt.dsfdebug.dsf.commands.MIStubCommandWrapper.getResult(MIStubCommandWrapper.
  at org.eclipse.cdt.dsf.mi.service.command.AbstractMIControl$RxThread.processMIOutput(AbstractM
  at org.eclipse.cdt.dsf.mi.service.command.AbstractMIControl$RxThread.run(AbstractMIControl.java:
```

Figure 19. Debug initialization error (2)

4.5 Example

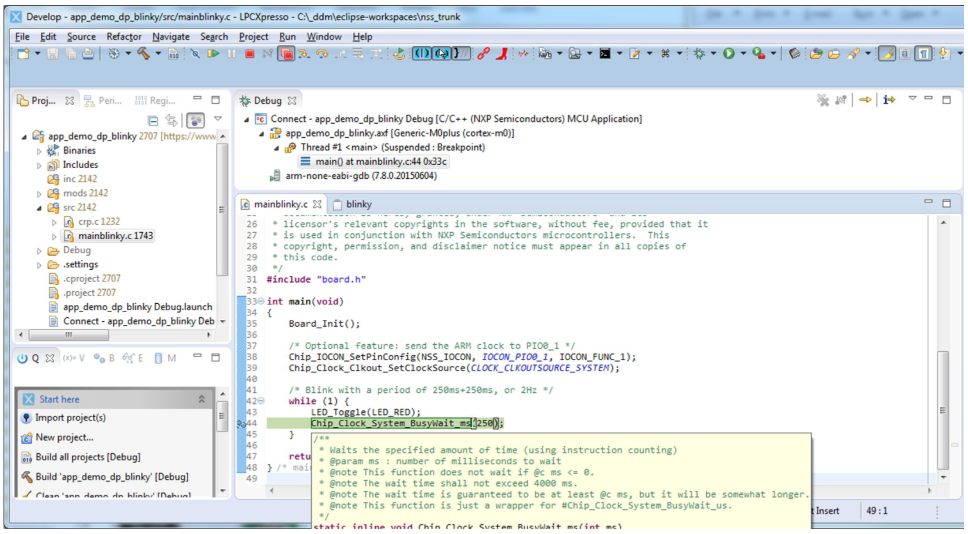
The blinky application is a very basic Hello World application. As an example, we debug the code and perform some basic debugging steps.

The full application code is:

```
#include "board.h"

int main(void)
{
    Board_Init();

    /* Optional feature: send the ARM clock to PIO0_1 */
    Chip_IOCON_SetPinConfig(NSS_IOCON, IOCON_PIO0_1,
 IOCON_FUNC_1);

 Chip_Clock_Clkout_SetClockSource(CLOCK_CLKOUTSOURCE_SYSTEM);

    /* Blink with a period of 250ms+250ms, or 2Hz */
    while (1) {
        LED_Toggle(LED_RED);
        Chip_Clock_System_BusyWait_ms(250);
    }

    return 0;
}
```

To debug the code, use the Debug ‘app_demo_dp_blinky’ [Debug] command in the Quickstart panel. It creates a default launch configuration (if necessary) and, using that configuration, flashes the correct image and starts a debugging session with the ARM core halted on the first instruction in main. You can step into and over each function, inspect and change memory contents, set and change conditional and data breakpoints, and change the Program Counter. Every debugging feature supported by ARM is available.

After resuming the application you can pause it again by, e.g., adding a breakpoint inside the `while(1)` loop. Double-click the desired line in the left gutter of the file editor view, where it halts the execution immediately (see Figure 20).



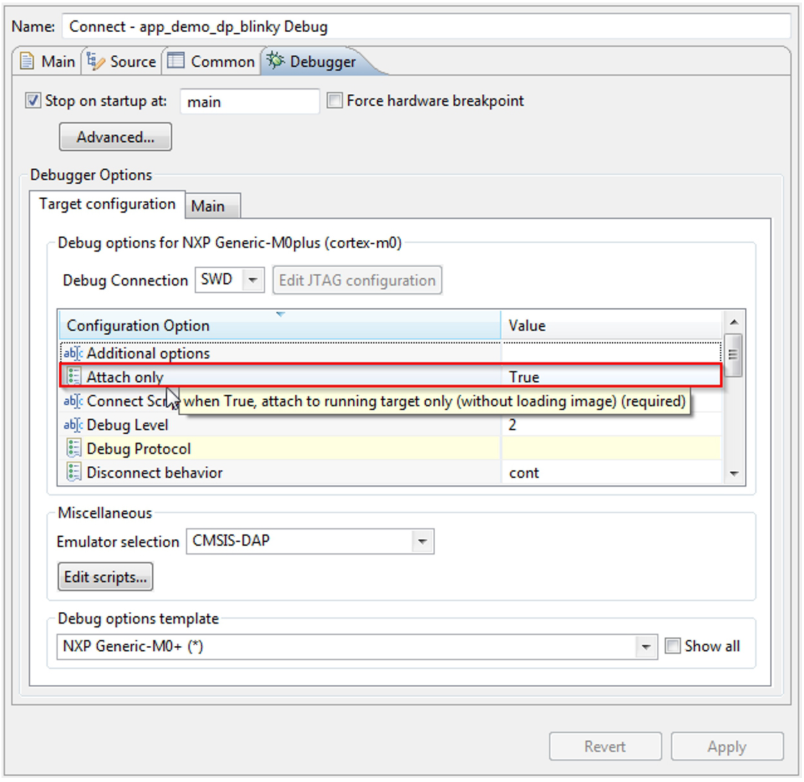
```
Connect - app_demo_dp_blinky Debug [C/C++ (NXP Semiconductors) MCU Application]
  app_demo_dp_blinky.axf [Generic-M0plus (cortex-m0)]
    Thread #1 <main> (Suspended : Breakpoint)
      main() at mainblinky.c:44 0x33c
  arm-none-eabi-gdb (7.8.0.20150604)
```

```
 * licensor's relevant copyrights in the software, without fee, provided that it
 * is used in conjunction with NXP Semiconductors microcontrollers.  This
 * copyright, permission, and disclaimer notice must appear in all copies of
 * this code.
 */
#include "board.h"

int main(void)
{
    Board_Init();

    /* Optional feature: send the ARM clock to PIO0_1 */
    Chip_IOCON_SetPinConfig(NSS_IOCON, IOCON_PIO0_1, IOCON_FUNC_1);
    Chip_Clock_Clkout_SetClockSource(CLOCK_CLKOUTSOURCE_SYSTEM);

    /* Blink with a period of 250ms+250ms, or 2Hz */
    while (1) {
        LED_Toggle(LED_RED);
        Chip_Clock_System_BusyWait_ms(250);
    }
    /**
     * Waits the specified amount of time (using instruction counting)
     * @param ms : number of milliseconds to wait
     * @note This function does not wait if @c ms <= 0.
     * @note The wait time shall not exceed 4000 ms.
     * @note The wait time is guaranteed to be at least @c ms, but it will be somewhat longer.
     * @note This function is just a wrapper for #Chip_Clock_System_BusyWait_us.
     */
    static inline void Chip_Clock_System_BusyWait_ms(int ms)
```

aaa-031872

Figure 20. Debug session with ARM halted due to a break point

You can stop a debugging session, which leaves the application running and afterward launch a new debug session by creating a launch configuration where you instruct the GDB debugger in the LPCXpresso IDE to connect only. For launching a new configuration, go to Run > Debug Configurations… Copy the existing debug launch configuration and, under the Debugger tab, change the Configuration Option Attach only to True (see Figure 21). The connect-only launch configuration prevents that a new firmware image is downloaded in the ARM and lets you investigate a running instance.



Name: Connect - app_demo_dp_blinky Debug

[✓] Stop on startup at: main
[ ] Force hardware breakpoint

Debugger Options

Debug options for NXP Generic-M0plus (cortex-m0)

Debug Connection: SWD

Configuration Option	Value
Additional options	
Attach only	True
Connect Sc…	
Debug Level	2
Debug Protocol	
Disconnect behavior	cont

when True, attach to running target only (without loading image) (required)

Miscellaneous

Emulator selection: CMSIS-DAP

Debug options template

NXP Generic-M0+ (*)

[ ] Show all

Figure 21. Creating a connect-only launch configuration

5 Conclusion

In this document, a basic overview has been given to unlock the full potential of the HW with an emphasis on the tooling and the SW.

More detailed information can be found in:

- The data sheet,
- The schematics of each respective board,
- The SW documentation – near the code itself, or as generated HTML pages.

All these documents are part of the Software Development Kit.

Be sure to check the user manuals for the demo application(s) created for each specific IC. They give a higher-level idea of the potential that can be reached with our ICs.

Further releases of the HW, the SW, and the Software Development Kit provide further improvements to HW, SW, and documentation alike. To set our priorities correctly, we highly value receiving feedback. All questions, both business and technical are greatly appreciated.

6 Legal information

Figures

Contents