Application Note: JN-AN-1202

BeyondStudio Migration Guidelines

This Application Note provides an introduction to the new features included in the 'BeyondStudio for NXP' Integrated Development Environment (IDE) which can be used to develop applications for NXP's JN516x family of wireless microcontrollers. Guidance is also provided on migrating existing JN516x projects to BeyondStudio for NXP.

Introduction

Software applications for the JN516x wireless microcontrollers were previously developed in the Eclipse IDE supplied in NXP's JN51xx SDK Toolchain (JN-SW-4041). This toolchain is being replaced by 'BeyondStudio for NXP', which is an NXP adaptation of the BeyondStudio IDE v1.1.9 (from Beyond Semiconductor) specifically for use with the JN516x wireless microcontrollers. BeyondStudio is itself based on the open-source Eclipse IDE.

BeyondStudio for NXP is supplied in the software package with part number JN-SW-4141. Installation instructions and basic operational instructions are provided in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

Note: Before installing BeyondStudio for NXP, check that it is supported for your proposed software stack. To do this, refer to the relevant tab of the NXP Wireless Connectivity TechZone.

This document describes the features that have been introduced into the NXP version of BeyondStudio. It also provides guidance on migrating JN516x projects to BeyondStudio for NXP that were previously developed using the former Eclipse IDE.

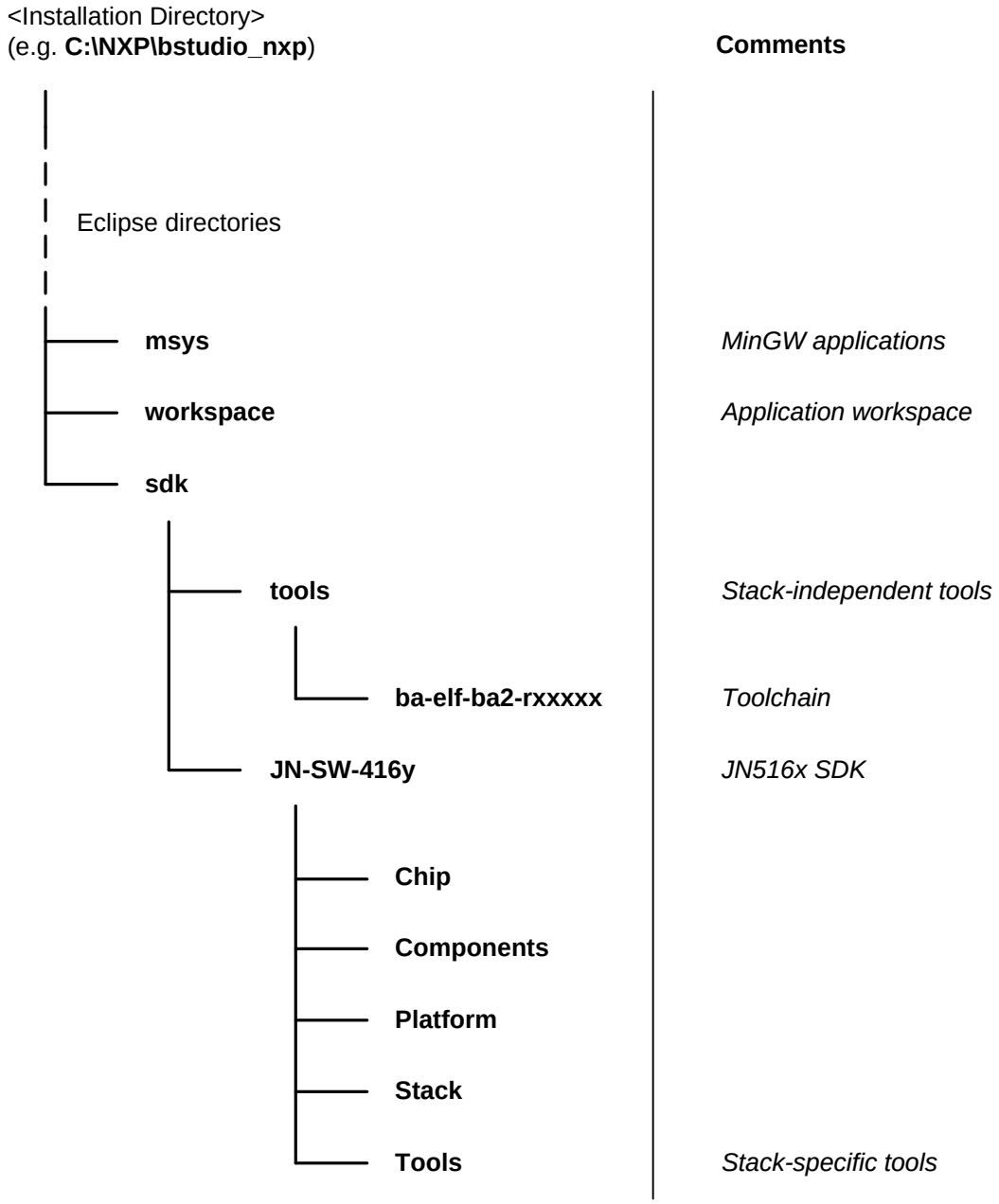
Compatibility

The software descriptions in this Application Note relate to the following NXP evaluation kits and software:

Product Type	Part Number	Version
Evaluation Kit	JN516x-EK001	-
BeyondStudio for NXP	JN-SW-4141	v1111

Directory Structure

The directory structure of the installation has changed slightly from the previous Eclipse version. This is to allow multiple stacks to be installed and used concurrently. All applications are created within the Eclipse workspace folder. The new directory structure is illustrated below.



Application Changes

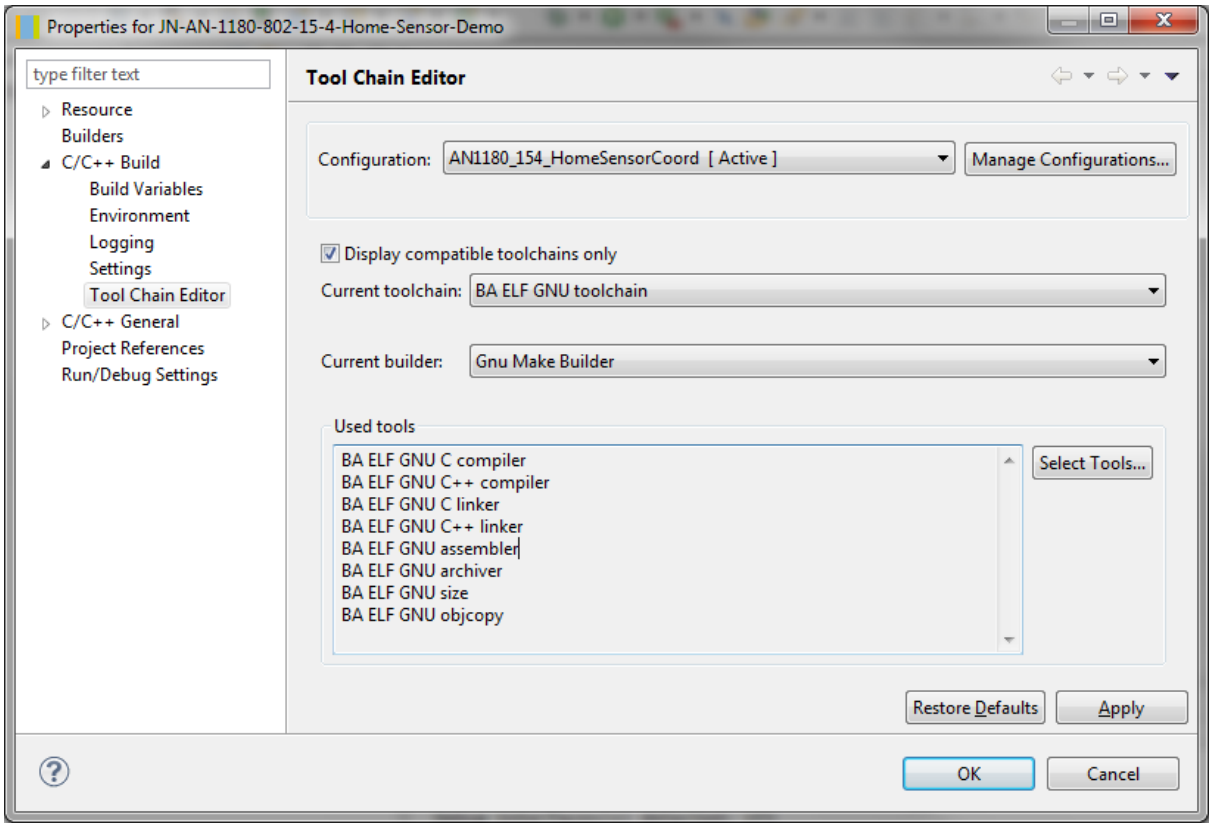
The following changes are necessary to migrate a project to BeyondStudio for NXP. These guidelines assume that you have launched BeyondStudio for NXP and imported the project, as described in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

Project Changes

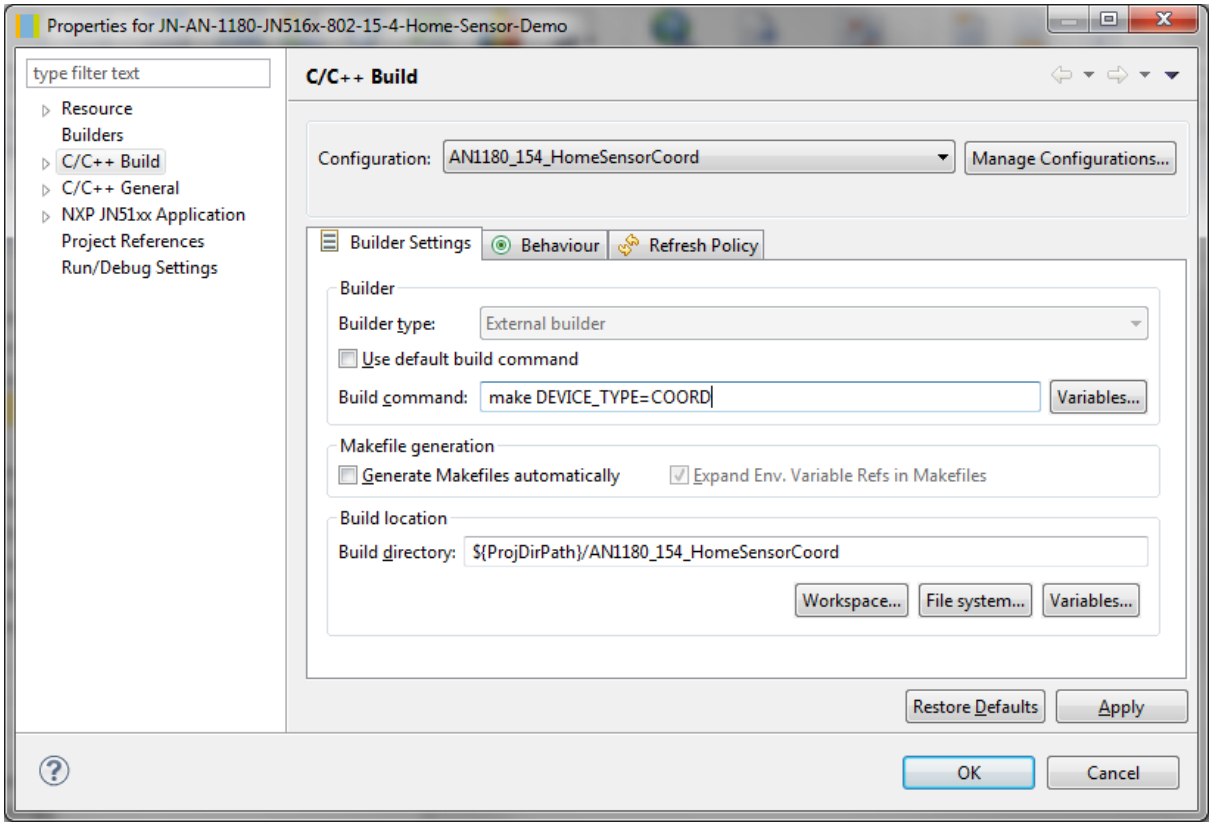
Launch the **Properties** dialogue box for your project by right-clicking on the project in the Project Explorer pane (left) and selecting **Properties** from the menu.

Under **C/C++ Build**, check whether the "**Use default build command**" box is ticked. If not, make a note of the text in the **Build command** field, as this information may be lost.

The project must now be configured to use the BA ELF GNU toolchain. Follow the path **C/C++ Build > Tool Chain Editor** and in the **Current toolchain** field, select "BA ELF GNU toolchain". It may be necessary to uncheck the "**Display compatible toolchains only**" box in order to present the option. The toolchain selection should appear as shown below:



Under **C/C++ Build**, ensure that the "**Generate Makefiles automatically**" box is unticked. If there was previously a custom **make** command, re-enter this now. The build options should appear as shown below - in this example, there is a custom **make** command that supplies "DEVICE_TYPE=COORD" to make. Also ensure that the **Build directory** field is set correctly – for some applications, there is a **Build** sub-directory (where the Makefile is kept).



Repeat the above instructions for every build configuration.

Makefile Changes

Each application must now select the appropriate stack SDK. The following lines should be included in the Makefile (this example selects JN-SW-4163, the IEEE802.15.4 SDK):

```
##############################################################################
# Default SDK is the IEEE802.15.4 SDK

JENNIC_SDK ?= JN-SW-4163

##############################################################################
# Path definitions

SDK_BASE_DIR                ?= $(abspath ../../../../sdk/$(JENNIC_SDK))
```

Note that SDK_BASE_DIR must be modified, as shown above, to take the SDK value.

Available toolchains are installed in **<Installation Directory>/sdk/Tools/<Toolchain name>.** Each stack SDK selects the appropriate toolchain from this directory – no Makefile changes are required for this, as this selection is automatic.

System headers in dependency files generated during compilation may contain Windows-style absolute paths which are not compatible with MinGW make. This can be avoided by

modifying the application Makefile, replacing the flags –M with –MM and –MD with –MMD when generating the dependencies. These flags are passed to the GCC compiler by the Makefile. Building with these flags includes only user headers in the dependencies, removing the problematic system header paths.

The build process no longer supplies include paths for every available component. The application Makefile must add include paths as well as library names for the components it uses - for example, to add the DBG library:

```
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/DBG/Include
APPLIBS += DBG
```

Changes to the Debug Library

The SDK Libraries compatible with this new toolchain include several new features in the Debug (DBG) library.

Flags

The DBG library now has a flags field that may be used to fine-tune the library's operation. The flags are stored in the global variable `DBG_u32Flags`. The flags are as follows:

Flag	Meaning
DBG_FLAG_NONE	None of the following flags are set.
DBG_FLAG_OUTGOING_NL_CRNL	If set, every \n character in the outgoing string is sent as \r\n. This is for compatibility with certain terminal programs.
DBG_FLAG_AUTO_FLUSH	If set, **DBG_vFlush()** is called at the end of each **DBG_vPrintf()** invocation. The application may instead choose to flush the outgoing characters in idle time rather than at the end of each outputted string.
DBG_FLAG_FLUSH_WHEN_FULL	If the DBG back-end buffers outgoing characters then it will be automatically flushed when it is full, if this flag is set. Otherwise, characters that do not fit in the buffer may be lost.

DBG_FLAG_OUTGOING_NL_CRNL and DBG_FLAG_AUTO_FLUSH are set by default.

DBG Terminal IO

The DBG library has also been extended to allow full duplex interaction with a terminal, by allowing the reception of characters in addition to printing them. An application may poll for characters by calling:

```
DBG_iGetChar();
```

This function will return a character if one has been received or -1 if no character is available.

JTAG DBG_vPrintf

The SDK Libraries compatible with this new toolchain include a new back-end to the DBG library, allowing the user to **printf()** to the debug terminal within BeyondStudio for NXP.

Instead of initialising the DBG library with the UART back-end, when using JTAG debugging the JTAG back-end can be enabled as follows:

```
#if defined HWDEBUG
      DBG_vJtagInit();
#else
      DBG_vUartInit(DBG_UART, DBG_BAUD_RATE);
#endif /* HWDEBUG */
```


The debug **printf()** implementation may then be used as normal - for example:

```
DBG_vPrintf(TRUE, "Hello world!\n");
```

When the project is compiled for hardware debugging, this string will be printed to the BeyondStudio target debug console. Otherwise, the string will be printed to the UART and may be received in the BeyondStudio serial terminal.

When using the JTAG back-end to the DBG library, calling **DBG_vFlush()** (this is done automatically within **DBG_vPrintf()** if the DBG_FLAG_AUTO_FLUSH flag is set) will trigger a syscall exception in the CPU. This is caught by the debugger. When the syscall exception occurs, the address of the outgoing string buffer and its length are stored in CPU registers 4 and 5 respectively. The address of the incoming string buffer and its length are stored in CPU registers 6 and 7 respectively. The debugger reads the outgoing buffer and prints it. The debugger may also write data to the target in the incoming buffer. The debugger then resumes the CPU automatically.

Writing the incoming buffer is not supported by the debugger in this release. Calling **DBG_iGetChar()** when using the JTAG back-end will always return -1.

Standardised Back-end Functions

In previous SDKs, it was necessary to call certain back-end functions directly, such as to flush the UART. In this release, there are additional functions common to all back-ends that will call the appropriate function of the initialised back-end. This means that the back-end can be replaced (for example, by swapping from UART to JTAG) without changing any source code and with minimal conditional compilation.

The new functions are:

- **DBG_vFlush()** – Flushes outgoing characters:
  - For the UART back-end, this blocks until the transmit shift register is empty
  - For the JTAG back-end, it performs the system call to transfer the output buffer to the debugger
- **DBG_iGetChar()** – Reads incoming characters from the terminal:
  - For the UART back-end, this checks for a received character in the incoming buffer
  - For the JTAG back-end, it will return data that has been written into a buffer by the debugger (not currently supported by GDB)

Toolchain Features

MinGW

The new toolchain does not include Cygwin. The toolchain is compiled natively for Windows using MinGW. Users should not use the Cygwin shell when working with the new toolchain and should instead switch to the included MinGW shell. This is because tools compiled using MinGW expect Windows-style paths (e.g. **c:\test** or **/c/test/)** whereas tools compiled with Cygwin expect Cygwin-style paths (e.g. **/cygdrive/c/test**). The Cygwin shell will expand Windows-style paths to Cygwin-style paths, which MinGW applications cannot understand. Likewise, running Cygwin applications from a MinGW shell will not work because Cygwin applications do not understand Windows-style paths.

A shortcut to the MinGW shell is created in the Windows **Start** menu when BeyondStudio for NXP is installed. This shell may be used to run any installed MinGW applications.

Some common MinGW applications are installed to **<Installation directory>/msys/**. These include, for example, **GNU Make** for managing the building of applications.

Link-time Optimisation

The new toolchain supports link-time optimisation, which is enabled by default. Link-time optimisation makes it possible to perform many additional optimisations, due to the fact that the linker has visibility of the entire program. It can therefore do things such as inlining functions that are used in only one place, rearranging program sections in memory to reduce jump sizes, removing unused function arguments and variables, etc.

Link-time optimisation can be disabled by adding the following line to the application Makefile:

```
DISABLE_LTO=1
```

Of course, this will have a detrimental effect on the size of the generated program and can mean the binary file grows by 5-10%.

GCC Warnings

This release of the toolchain supports and enables GCC warnings as documented here:

http://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html

A description of these warnings and potential issues follows:

-Wpacked

This warning is given if the packed attribute is applied to a structure but has no effect on the layout or size of the structure. However, the packed attribute is regularly used to ensure that structures which encode protocol definitions are unpadded. In this case, the secondary property of the packed attribute which allows structures to be misaligned is useful, as byte buffers are frequently cast to/from these structures. If the structure was declared without the packed attribute then misalignment exceptions may occur. Therefore it is recommended that structures defining an ‘on-wire’ protocol be wrapped in the following:

```
#pragma GCC diagnostic ignored "-Wpacked"
#pragma GCC diagnostic ignored "-Wattributes"

/** Structure definitions */

#pragma GCC diagnostic pop
```

In all other cases, ignoring this warning will result in inefficient code being generated to access structure members. Use of the packed attribute is therefore discouraged. The programmer should take steps to ensure that structure members are, as far as practical, laid out to allow the compiler to pack them naturally. This means that structure members should be declared in order of descending alignment requirement, i.e. pointers first, 32-bit values next, 16-bit values next, with 8-bit values and bit-fields last.

-Wcast-align

This warning is given whenever a pointer is cast such that the required alignment of the target is increased. For example, casting an 8-bit byte pointer to a 32-bit word pointer will give this warning. If the source pointer is not word-aligned (which the compiler cannot guarantee in this case) then this will generate an alignment exception at run-time. It is better

to be warned about this at compile-time, so that the code can be fixed, than to get alignment exceptions at run-time.

If data must be accessed as words then it should be copied from the source pointer into a word-aligned variable.

Possible Issues

Const Arrays with Function Scope

A const array declared locally in a function may be copied onto the stack rather than referenced in place, leading to sub-optimal memory usage and unnecessary CPU cycles being consumed. This can be fixed by declaring the array as a static const within the function.

Link-time Optimisation Removing Arguments

If a function does not make use of some arguments then GCC may choose to remove them from the calling code. Normally this is not a problem, since if they are genuinely not used then it is wasteful to generate the code to pass them. Problems may occur, however, if the function relies on those arguments being present - for example, in inline assembly. In this case, assembly constraints should be used to inform GCC that the arguments are used by the assembly code. If the inline assembly implicitly uses arguments that are being optimised out then it may be necessary to disable LTO when compiling the C file in order to prevent this optimisation.

Internal Compiler Error in do_SUBST

The following error message has been observed when compiling certain code constructs:

```
internal compiler error: in do_SUBST
```

This appears to be a bug in the GCC compiler. If this error is observed, the workaround is to disable optimisation of the function in which the compiler error occurs. To do this, add the following attribute to the function:

```
__attribute__((optimize("O0")))
```

So the function definition appears in full as:

```
void __attribute__((optimize("O0"))) function_name (void)
{
}
```


Revision History

Version	Notes
1.0	First release
1.1	Updated branding

Important Notice

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

All trademarks are the property of their respective owners.

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com