



Application Note: JN-AN-1069

IEEE 802.15.4 Serial Cable Replacement

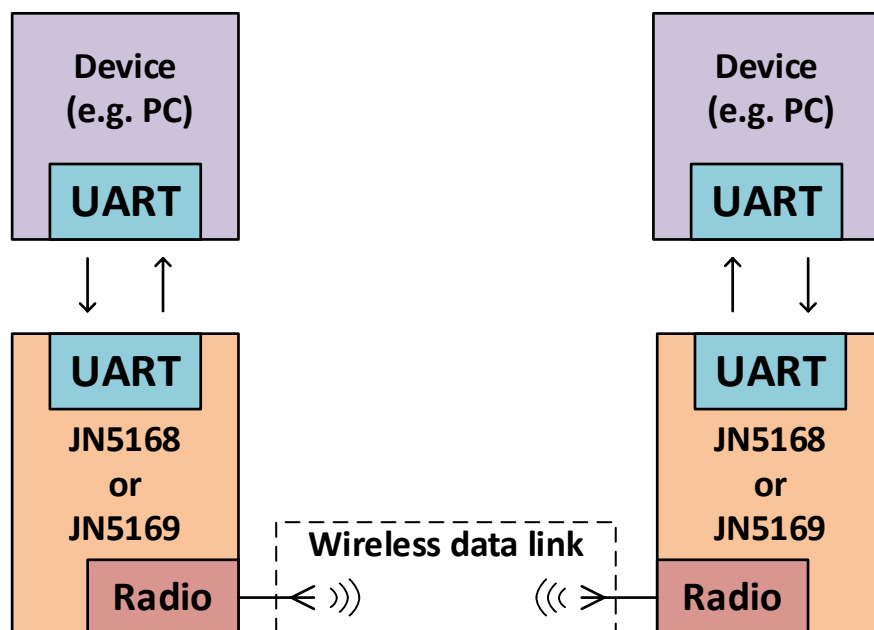
This Application Note describes how to create a wireless UART link between the UARTs on two JN5168 or JN5169 modules, using two DR1174 Carrier Boards. The features of the application include:

- Replacement of serial cables with wireless connectivity
- Quick addition of low-cost wireless connectivity to products using a UART link for communications
- UART flow control using hardware with the RTS/CTS lines or software with the XON/XOFF protocol to control the flow of data into and out of the UART
- Radio flow control using a protocol to control the flow of data over the radio

This application was developed using APIs from the NXP IEEE 802.15.4 Software Developer's Kit (SDK) [JN-SW-4163], available free-of-charge via the [NXP Wireless Connectivity TechZone](#).

1 Application Overview

The application uses a pair of Carrier Boards (DR1174) fitted with JN5168 or JN5169 modules to form a wireless network. Data received by a UART on one board is transmitted via the wireless radio link to the second board, where it is output by the second board's UART, and vice versa. This allows two devices (such as PCs) to communicate via a wireless radio link. This is illustrated in the figure below:



Two different binaries are built as part of this application:

1. The Coordinator initially creates the network, then runs in the same way as the End Device.
2. The End Device initially joins the network, then runs in the same way as the Coordinator.

It is assumed that a permanent power source is available at both ends of the wireless link, allowing the radio to always be active and ready to transmit or receive data.

1.1 Starting the Application

Connect each board to a PC using the USB-to-serial cables provided with the Evaluation Kit. UART0 on the boards is used by default.

- When the boards are first powered on, LEDs 0 and 1 (marked as D6 and D3 on the board) flash alternately while the node is creating or joining the network.
- Once the node is a member of a network, LEDs 0 and 1 flash together while the node attempts to pair with another node in the network.
- When the node has paired, LED 0 is illuminated while transmit is enabled on the UART and LED 1 is illuminated while receive is enabled on the UART.

1.2 Running the Application



Caution: Ensure the JN51xx Flash Programmer software is disconnected from the serial port on the PC, if using the same serial port for programming and running the application.

A terminal emulator can be used to send data between the boards. The serial connection is 115200 bps, 8 data bits, no parity, and 1 stop bit with hardware flow control, by default.

Data entered into one board's terminal emulator is passed to the board's UART and then transmitted over the radio to the other board. Here, it is received and passed via the board's UART to that board's terminal emulator.

Whenever data is transmitted, LED 0 is extinguished while the node waits for an acknowledgement from its paired node.

1.3 Using a Carrier Board Fitted with LCD Expansion Board

When the Coordinator or End Device binary file is loaded into a carrier board fitted with a LCD Expansion Board (DR1215), the screen will display a looping count of the data bytes transmitted and received by the various modules of the application, along with the rate of throughput (bytes per second) of the queues.

2 Software Design

This Application Note was created using the C function APIs (Application Programming Interfaces) of the JN516x SDK, which contain software libraries and tools for developing wireless networking applications to run on the NXP wireless microcontrollers. The main APIs used were as follows:

- The IEEE 802.15.4 APIs provide functions for controlling the wireless network.
- The JN516x Integrated Peripherals API provides functions for controlling the on-chip hardware peripherals of the JN516x family of devices.

A User Guide (JN-UG-3024) and other detailed information about IEEE 802.15.4 can be found on the [NXP Wireless Connectivity TechZone](#).

Apart from the creation and joining of the network, the software contained in each node is identical.

2.1 Module Overview

The source files required to build the application are described below:

2.1.1 config.h

This file contains definitions controlling the operation of the radio network, including the channels to be used to form the network and the PAN ID of the network to be formed.

2.1.2 crd_coordinator.c/h

The Coordinator node is responsible for creating the network and allowing other nodes to join the network as children of the Coordinator.

crd_coordinator.c contains the standard set of functions for starting and running a IEEE 802.15.4 Coordinator node.

2.1.3 ed_enddevice.c/h

An End Device node can join the network created by the Coordinator node.

ed_enddevice.c contains the standard set of functions for starting and running a IEEE 802.15.4 End Device node.

2.1.4 node.c/h

node.c provides networking code common to both the Coordinator and End Device nodes.

Functions are provided to initialise the stack and application, maintain a timer for the application and transmit data via the network.

The data used by this module is held in the global **NODE_sData** structure defined in **node.h**.

2.1.5 `wuart.c/h`

This module provides the main functionality of this wireless UART application.

The module runs a state machine when the node becomes a member of the network. This state machine initially acts to form a pairing with another unpaired node in the network. Once paired, the state machine manages the exchange of data between the paired nodes forming a wireless UART. Further detail on this state machine is provided in the [Wireless UART State Machine](#) section of this document.

Two circular queues (implemented by `queue.c`) are allocated:

1. The receive queue is filled with data received by the UART (in `uart.c`). This module takes the data from the receive queue and transmits it over the radio to the paired node.
2. The transmit queue is filled with data received over the radio by this module. The data in the transmit queue is then transmitted by the UART (in `uart.c`).

Flow control over the radio is maintained by this module:

- When the transmit queue becomes almost full (usually because the UART is not allowed to transmit due to flow control being asserted on the UART), the node indicates this over the radio to its paired node, which stops transmitting further data.
- When the transmit queue begins to empty again, the node indicates this over the radio to its paired node which is then permitted to transmit further data.

When exchanging data between the paired nodes, a single message is used to:

- Indicate the current status of the node (if it is able to transmit and receive, if the receive state is changed, if the node is simply checking the connection).
- Acknowledge data and status changes in the previously received message.
- Send new data to the other device.

In this way, when the nodes are sending data to each other, they will alternate their transmissions, sending both data and acknowledgements in a single message, thus maximising the throughput of data.

The data used by this module is held in the global **WUART_sData** structure defined in `wuart.h`.

This module acts as the hub for the communications between the UART and the radio transceiver, and allocates the queue storage. There are many defines in `wuart.h` to configure the UART port settings, and the sizes and fill levels of the queues.

When **WUART_STATS** is set to **TRUE** in `wuart.h`, statistics on the data being transmitted and received over the radio will be maintained.

When **WUART_LCD** is set to **TRUE** in `wuart.h`, the statistics collected by the various modules will be displayed on the LCD screen of the LCD Expansion Board (DR1215), for monitoring purposes.

2.1.6 uart.c/h

This module provides a high-level interface to the UART hardware, running the UART in an interrupt-driven mode.

The serial queues (allocated in **wuart.c**) are used by the UART code:

- When data is received by the UART, it is added to the receive queue. If the queue becomes almost full, the flow control protocol being used is asserted to prevent the attached device sending any further data. When the queue starts to empty again, the flow control protocol being used is de-asserted allowing further data to be received.
- The UART also monitors the transmit queue, outputting any data in the transmit queue to the attached device. The flow control protocol being used is also monitored - if the attached device indicates that it is not able to receive further data, this condition is detected and no further data is transmitted by the UART until the condition is cleared.

The data used by this module is held in the private **asUart** structure array defined in **uart.h**.

When **UART_STATS** is set to **TRUE** in **uart.h**, statistics on the data being transmitted and received over the UART will be maintained.

2.1.7 queue.c/h

queue.c provides code to manage circular queues.

Each queue is maintained using a **QUEUE_tsData** structure along with a data buffer. These are allocated in **wuart.c** and passed as pointers into **uart.c** during initialisation, so that they can be accessed by both the radio and UART code. The queue functions operate on pointers to a queue and provide the following features:

- Add data to and remove data from a queue.
- Monitor when a queue is almost full and is emptying, to allow flow control to be applied.
- When **QUEUE_STATS** is set to **TRUE** in **queue.h**, statistics on the amount of data and throughput of the queue are calculated.

2.1.8 lcd.c/h

This module provides a high-level interface to display text and numeric values on the LCD screen of the LCD Expansion Board (DR1215).

Code in **wuart.c** gathers the statistics from the other modules in the application and displays them using the functions in **lcd.c**, when the **WUART_LCD** define is set to **TRUE** in **wuart.h**.

2.1.9 rnd.c/h

This module provides a set of random number generating functions, using the hardware random number generator of the JN5168 and JN5169 devices.

2.1.10 dbg.c/h

This module provides functions allowing debug information to be output to a serial port.

2.2 Functional Overview

2.2.1 Message Format

All messages sent by the application over the radio have the same basic format:

```
Start SeqTx SeqRx Command [Data] End
```

where:

`Start` is a single character indicating the start of the message, always '!'.

`SeqTx` is a single character ('!' through to '~') that increments with each message transmission, and can be used to detect message received multiple times.

`SeqRx` is a single character ('!' through to '~') that indicates the `SeqTx` in the previous message which this message is responding to. Space is used when the message is not a response.

`Command` is a single character indicating the command being sent. For messages that are commands, these are all upper-case characters, whilst messages that are sent in reply to commands, they are lower-case characters.

`Data` is additional data which may be included depending on the command being sent.

`End` is a single character indicating the end of the message - always 0 (null terminator).

2.2.2 Pairing

Once a node is in the network, it must find another node with which to pair in order to exchange data.

While a node is unpaired, it will regularly broadcast an idle query message to the network. Any other unpaired node that receives such a message will send an idle response message back to the broadcasting node.

Upon receiving an idle response, an unpaired node will exchange a sequence of messages with the other unpaired node in order to form a pair. These messages reflect the steps that a standard dial-up modem follows when negotiating a connection and are illustrated in the [Wireless UART State Machine](#) section of this document.

2.2.3 Data Message Format

Once paired together, the paired nodes exchange data using only the data command message. This message may form both a query and reply in a single message. The data command is represented by the command character being 'D'.

The data portion of the message then has the following format:

```
Status Acks [DataSeq] [Data]
```

where:

`Status` is a single character ('0' through to '?') with the least significant four bits indicating the status of the node:

0x01 – Tx enabled, the node is permitted to transmit data over the radio.

0x02 – Rx enabled, the node is able to receive data over the radio.

0x04 – Rx changed, the node has changed its Rx enabled state and requires an acknowledgement.

0x08 – Ping, the node has not sent or received data recently, is checking for the presence of its paired node and requires an acknowledgement.

Acks is a single character ('0' through to '?') with the least significant four bits indicating acknowledgements of the paired node's previous message:

0x01 – Data ack, the node accepted the previously sent packet of data.

0x02 – Data nak, the node was not able to accept the previously sent packet of data.

0x04 – Rx change ack, the node is acknowledging the previously sent Rx changed status flag.

0x08 – Ping ack, the node is acknowledging the previously sent ping status flag.

DataSeq is a single character ('!' through to '~') that increments with each transmission that includes data. Can be used to ensure data that is resent over the radio is not added to the receiving node's UART output queue a second time. Only included when data is present in the message.

Data is data to be output on the receiving node's UART. Only included when data is present in the message.

If both nodes need to transmit data at the same time, a node is able to include its next data message with the acknowledgement of the other node's previous message. This results in an efficient use of the radio, as each node alternates its data transmissions.

If a node fails to get a response from its paired node after a number of attempts, it will return to an unpaired state and re-start the process of finding another node to pair with.

2.2.4 Flow Control

End-to-end flow control between two nodes A and B, where A is transmitting to B, works as follows:

1. Node B is instructed by the attached device not to transmit any more data from its UART.
2. Node B's UART transmit queue begins to fill and eventually becomes low on free space.
3. Node B indicates to node A that it is not able to receive further data over the radio.
4. Node A acknowledges that it will not transmit further data to node B.
5. Node B receives the acknowledgement and checks that node A has indicated that it will not transmit further data – if this is not the case then node A sends another status message to node B.
6. Node A is still receiving data on its UART from its attached device.
7. Node A's UART receive queue begins to fill and eventually becomes low on free space.
8. Node A instructs the attached device to stop transmitting data.

Therefore, as the queues begin to fill, the flow control works its way through the system as required. When node B is once again allowed to transmit data to its attached device, the queues will begin to empty and the flags are reversed through the system in a similar way, allowing data to flow once again.

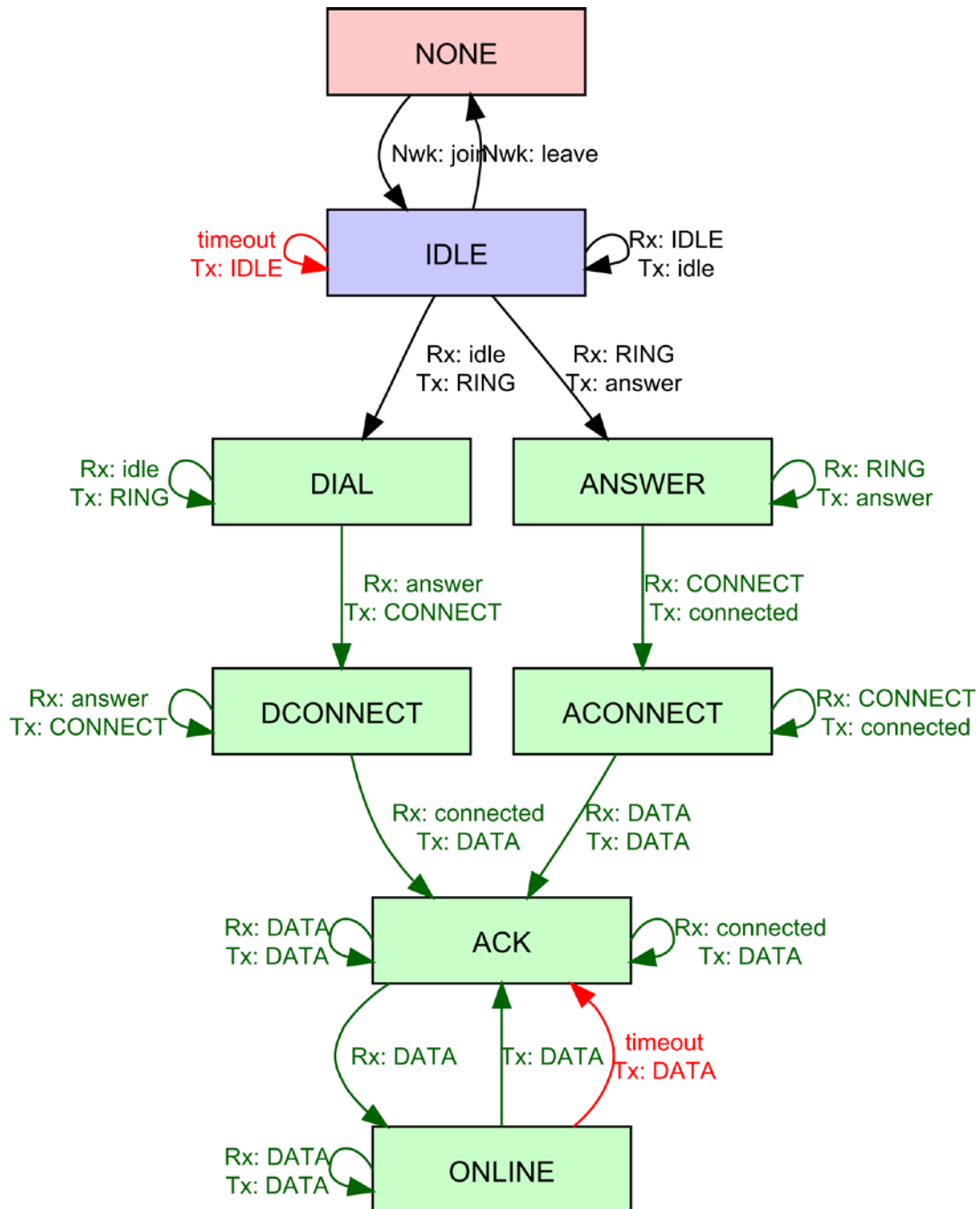
2.2.5 Wireless UART State Machine

The main transitions of the state machine implemented in **wuart.h** are shown in the figure below.

- The black state transition arrows represent transactions caused by network changes or receiving data from any node.
- The green transitions represent data received only from the paired node or the node currently being paired with.
- The red transitions represent transitions caused by the state timing out, either on a regular basis or due to not receiving the expected data.

The following error case transitions are not shown in the figure:

- All the green coloured states timeout to the IDLE state if the expected data is not received. The ACK state only times out to the IDLE state after a number of resends have failed.
- If an unexpected message is received from the paired node in any of the green coloured states, the node returns to the IDLE state responding with an ERROR message.
- If the ERROR message is received from the paired node in any of the green coloured states, the node returns to the IDLE state.
- If messages are received from nodes other than the paired node while in a green coloured state, an ERROR message is returned but the receiving node's state remains unchanged.



3 Compatibility

The software provided with this Application Note is intended to be used with the following NXP hardware and software products:

Product Type	Part Number	Version
Evaluation Kit	JN516x-EK001	-
JN516x SDK	JN-SW-4163	1307
Toolchain	JN-SW-4141	1217

4 Building and Loading the Application

4.1 Pre-requisites

It is assumed that you have installed the relevant NXP JN516x SDK on your PC – that is, the BeyondStudio for NXP toolchain (JN-SW-4141) and JN516x IEEE802.15.4 SDK (JN-SW-4163).

In order to build the application, this Application Note (JN-AN-1069) must be unzipped into the directory:

<BeyondStudio for NXP installation root>\workspace

where **<BeyondStudio for NXP Installation root>** is the path into which BeyondStudio for NXP was installed (by default, this is **C:\NXP\bstudio_nxp**). The **workspace** directory is automatically created when you start BeyondStudio for NXP.

All files should then be located in the directory:

...\workspace\JN-AN-1069- IEEE-802.15.4-Serial-Cable-Replacement

There is a sub-directory for each application, each having **Source** and **Build** sub-directories.

4.2 Build Instructions

The applications can be built from the command line using the makefiles or from BeyondStudio for NXP – makefiles and Eclipse-based project files are supplied.

- To build using makefiles, refer to Section 4.2.1.
- To build using BeyondStudio for NXP, refer to Section 4.2.2.

4.2.1 Using Makefiles

This section describes how to use the supplied makefiles to build the applications. Each application (e.g. for Coordinator or End Device) has its own **Build** directory, which contains the makefiles for the application.

To build an application and load it into a JN5168 or JN5169 board, follow the instructions below:

1. Ensure that the project directory is located in
<BeyondStudio for NXP installation root>\workspace
2. Start an MSYS shell by following the Windows Start menu path:
All Programs > NXP > MSYS Shell

3. Navigate to the **Build** directory for the application to be built and follow the instructions below:

For JN5168:

At the command prompt, enter:

```
make JENNIC_CHIP=JN5168 clean all
```

For JN5169:

At the command prompt, enter:

```
make JENNIC_CHIP=JN5169 clean all
```

You can alternatively enter the following command from the top level of the project directory, which will build the binaries for all applications:

```
make clean all
```


In all the above cases, the binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **JN5168**) for which the application was built.

4. Load the resulting binary file into the board. You can do this from the command line using the JN51xx Production Flash Programmer (described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*).

4.2.2 Using BeyondStudio for NXP

This section describes how to use BeyondStudio for NXP to build the demonstration application.

To build the application and load it into a JN516x board, follow the instructions below:

1. Ensure that the project directory is located in
<BeyondStudio for NXP installation root>\workspace
2. Start the BeyondStudio for NXP and import the relevant project as follows:
 - a) In BeyondStudio, follow the menu path **File>Import** to display the **Import** dialogue box.
 - b) In the dialogue box, expand **General**, select **Existing Projects into Workspace** and click **Next**.
 - c) Enable **Select root directory** and browse to the **workspace** directory.
 - d) In the **Projects** box, select the project to be imported and click **Finish**.
3. Build an application. To do this, ensure that the project is highlighted in the left panel of BeyondStudio and use the drop-down list associated with the hammer icon  in the toolbar to select the relevant build configuration – once selected, the application will automatically build. Repeat this to build the other applications.

The binary files will be created in the relevant **Build** directories for the applications.
4. Load the resulting binary files into the board. You can do this using the integrated Flash programmer, as described in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

5 Application Code Sizes

The applications of this Application Note have the following memory footprints on the JN5168 and JN5169 devices, when using the JN516x IEEE802.15.4 SDK (JN-SW-4163):

Application	Text Size (Bytes)	Data Size (Bytes)	BSS Size (Bytes)
Coordinator_JN5168	38032	288	26787
EndDevice_JN5168	38720	288	27795
Coordinator_JN5169	38895	280	26803
EndDevice_JN5169	39575	280	27811

Revision History

Version	Notes
1.0	First release
2.0	Re-written for improved performance. JN5148 support added.
3.0	JN5168 and JN5169 support added. JN5148 and JN5139 support dropped.

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