



Application Note: JN-AN-1178

IEEE 802.15.4 Wireless UART for JN516x

This Application Note describes how to create a wireless radio link between two UART devices. The link is based on the IEEE 802.15.4 protocol and is demonstrated using two NXP evaluation kit boards featuring the JN516x wireless microcontroller.

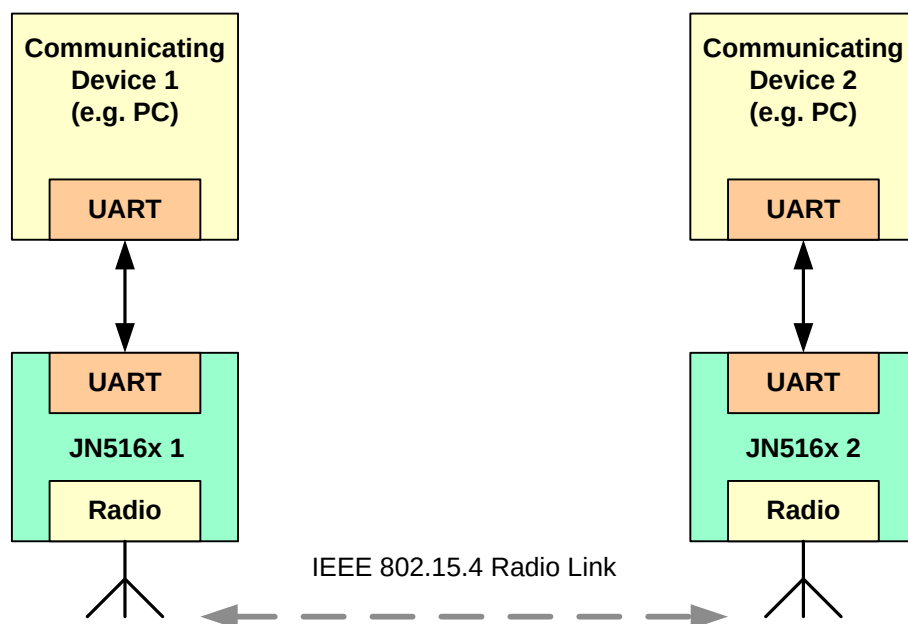
The benefits of using this solution include:

- Replacement of serial cables with wireless connectivity
- Quick addition of low-cost wireless connectivity to existing products that use a UART link for communications
- Hardware flow control using the RTS/CTS lines 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

The software required to implement the system was developed using NXP's IEEE 802.15.4 Stack API and JN516x Integrated Peripherals API, which are both supplied in the JN516x IEEE 802.15.4 Software Developer's Kit (JN-SW-4063).

1 Application Overview

The application uses two boards from the NXP JN516x-EK001 Evaluation Kit to form a two-node wireless network. Data received by a UART on the JN516x device of one board is transmitted via an IEEE 802.15.4 wireless radio link to the second board, where it is output by the second board's JN516x 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.



1.1 Pre-built Applications

Two different application binary files are provided with this Application Note, one for each board (containing a JN5168 wireless microcontroller):

- **AN1178_154_WUART_Coord_JN5168_NO_FLOW_CTRL.bin**: This application is for the Co-ordinator node which initially creates the network and then operates in the same way as the other node.
- **AN1178_154_WUART_EndD_JN5168_NO_FLOW_CTRL.bin**: This application is for the End Device node which initially joins the network (created by the Co-ordinator) and then operates in the same way as the other node.

Flow control is not implemented in the above pre-built applications but build options are provided to re-build the applications with flow control (see Section 3 and Section 5).

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

1.2 Loading the Applications

The application binaries can be loaded into any of the JN516x-EK001 Evaluation Kit boards that feature a JN5168 device – that is, a DR1174 Carrier Board fitted with one of the expansion boards. UART0 on the JN516x device is used for the communication link, by default, and will work with any of the expansion boards that are included in the evaluation kit. It is, however, recommended that the DR1199 Generic Expansion Boards are used, since their LEDs will indicate the various modes of operation (as described below).

To load the application binaries into the boards from a PC, use the JN51xx Flash Programmer tool described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*.

1.3 Starting the Applications

Connect each board to a PC using the USB-to-mini USB cables provided with the evaluation kit. First start the application on the Co-ordinator, then start the application on the End Device. To start an application, reset the relevant board.

The LEDs on a Generic Expansion Board are used to indicate the various modes of operation as follows:

- When a board is first powered on, LED0 (marked as D1 on the board) flashes periodically to show that the device is active.
- Once the End Device has associated with the Co-ordinator, LED1 (D2) is illuminated.

1.4 Using the Applications



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, such as Tera Term, can be used on the PCs to send data between the boards. The serial connection between a PC and a board should be configured as follows: 115200 bps, 8 data bits, no parity, 1 stop bit. The flow control mode is determined by which binaries have been programmed into the devices (i.e. none or hardware) – when using the pre-built binaries, there is no flow control.

Data entered into one board's terminal emulator window 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 second board's UART to the second terminal emulator window.

Whenever data is transmitted, LED0 on the Generic Expansion Board is extinguished on the transmitting node while the node waits for an acknowledgement from its paired node.

2 Compatibility

The software provided with this Application Note has been tested with the following evaluation kits and SDK (Software Developer's Kit) versions:

Product Type	Part Number	Version
Evaluation Kit	JN516x-EK001	-
SDK Libraries	JN-SW-4063	983
SDK Toolchain	JN-SW-4041	1.1

3 Hardware Flow Control and Build Configurations

This Application Note includes three different 'flow control' build configurations for each of the Co-ordinator and End Device node types. The selected build option is added to the end of the filename of the resulting binary file. For build instructions, refer to Section 5.

3.1 No Flow Control (NO_FLOW_CONTROL)

In this configuration with the NO_FLOW_CONTROL build option, only the UART's Tx and Rx signals are used, and no hardware flow control is implemented.

3.2 Manual Hardware Flow Control (HW_SW_FLOW_CONTROL)

In this configuration with the HW_SW_FLOW_CONTROL build option, manual hardware flow control is enabled with UART_AUTOFLOW defined as FALSE (see **wuart.h**). In this mode, the RTS line is lowered by software as the receive queue begins to fill and is raised as the queue begins to empty. An interrupt is programmed to occur when the CTS line is changed. The state of the CTS line is then read and the transmission of data is allowed or denied dependent upon the state of the CTS line.

3.3 Automatic Hardware Flow Control (HW_FLOW_CONTROL)

In this configuration with the HW_FLOW_CONTROL build option, automatic hardware flow control is enabled with the UART_AUTOFLOW defined as TRUE (see **wuart.h**). In this mode, when the 16-byte receive FIFO begins to receive the 16th byte, the UART's RTS line is lowered and is then raised once space is available in the FIFO again. When the UART's CTS line is low, the UART will not transmit any data from the transmit FIFO.



Caution: The FTDI Serial-to-USB convertor supplied on the boards in the JN516x evaluation kit does not react quickly enough to the RTS signal being lowered, resulting in an extra character being sent to the UART which is then lost. When using two UARTs with automatic flow control, they should react quickly enough to prevent such data loss. Manual Hardware Flow Control is therefore recommended when using the evaluation kit and FTDI USB convertor.

4 Software Design

This Application Note was created using the C APIs (Application Programming Interfaces) of the JN516x IEEE 802.15.4 SDK (JN-SW-4063), which contains software libraries for developing IEEE 802.15.4 wireless networking applications to run on the NXP JN516x wireless microcontrollers. The main APIs used were as follows:

- IEEE 802.15.4 API, which provides functions for controlling the wireless network and is described in the *IEEE 802.15.4 User Guide (JN-UG-3024)*
- JN516x Integrated Peripherals API, which provides functions for controlling the on-chip hardware peripherals of the JN516x devices and is described in the *JN516x Integrated Peripherals API User Guide (JN-UG-3087)*

The above resources and other JN516x support collateral can be found on NXP's [Wireless Connectivity TechZone](#).

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

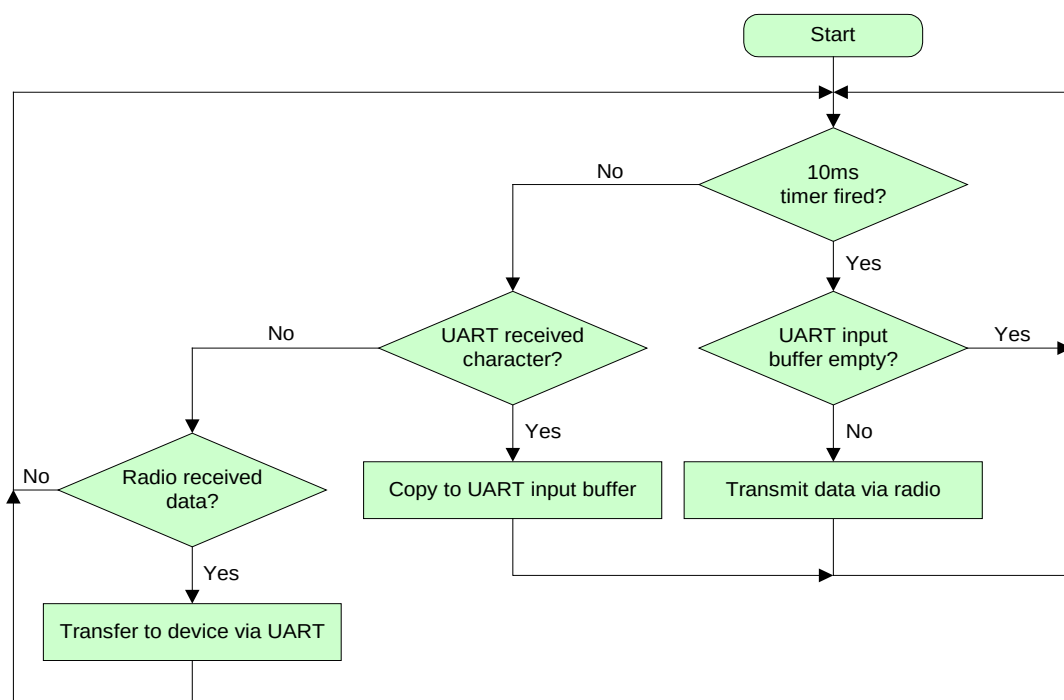


Figure 1: Software Flowchart

4.1 Module Overview

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

wuart.h

This file contains definitions which determine 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.

AN1178_154_WUART_Coord.c

This is the source file for the Co-ordinator node, which is responsible for creating the network and allowing other nodes to join the network as children of the Co-ordinator. The file contains the standard set of functions for starting and running an IEEE 802.15.4 Co-ordinator node.

AN1178_154_WUART_EndD.c

This is the source file for the End Device node, which can join the network created by the Co-ordinator node. The file contains the standard set of functions for starting and running an IEEE 802.15.4 End Device node.

protocol.c/h

This module provides the main functionality of this wireless UART application. Functions are provided to initialise the stack and application, maintain a timer for the application and transmit data via the network.

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.

Two circular queues (implemented by **serialq.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 **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.

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 (defined in **serialq.h**) 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.

serialq.c/h

This module provides code to manage the circular queues and includes 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

radio.c/h

This module contains common code for IEEE 802.15.4 radio operation including the receiving and transmitting of data packets.

5 Building and Loading the Application

This section describes how to build the applications provided in this Application Note and load the resulting binary files into NXP evaluation kit boards. The software can be built for the JN5168, JN5164 or JN5161 wireless microcontroller – the target device type appears in the filename of the resulting binary file.

The application can be built using the Eclipse IDE or makefiles. These two build methods are described separately in the sub-sections below.

5.1 Using Eclipse

This section describes how to build the application using the Eclipse IDE.

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

1. Ensure that the project directory is located in

<SDK_ROOT>\Application

where **<SDK_ROOT>** is the path into which the relevant NXP SDK was installed.

2. Start the Eclipse platform and import the relevant project files (**.project** and **.cproject**) as follows:
 - a) In Eclipse, 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**, browse to the **Application** directory and click **OK**.

- d) In the **Projects** box, select the project to be imported and click **Finish**.
3. Build the application. To do this, ensure that the project is highlighted in the left panel of Eclipse and use the drop-down list associated with the hammer icon  in the Eclipse toolbar to select the required build configuration (see Section 3) – 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, the resulting filenames indicating the chip type (e.g. **JN5168**) for which they were built.
4. Load the resulting binary files into the boards. You can do this using the JN51xx Flash Programmer, which can be launched from within Eclipse or used directly (and is described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*).

5.2 Using Makefiles

This section describes how to build the application using the supplied makefiles. The application for each node type (e.g. Co-ordinator, End Device) has its own **Build** directory, which contains the makefile for the application.

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

1. Ensure that the project directory is located in

<SDK_ROOT>\Application

where **<SDK_ROOT>** is the path into which the relevant NXP SDK was installed.

2. Navigate to the **Build** directory for the application to be built.
3. Run the makefile, specifying the required build configuration on the command line (see Section 3). Do this as illustrated below for your chip type (in these examples, the build configuration for manual flow control is specified):

For JN5168:

```
make FLOW_CONTROL=HW_SW_FLOW_CONTROL clean all
```

Note that for the JN5168, you can alternatively enter the above command from the top level of the project directory, which will build the binaries for both the applications.

For JN5164 or JN5161:

```
make JENNIC_CHIP=JN5164 FLOW_CONTROL=HW_SW_FLOW_CONTROL clean all
```

or

```
make JENNIC_CHIP=JN5161 FLOW_CONTROL=HW_SW_FLOW_CONTROL 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. To do this, use the JN51xx Flash Programmer, described in the *JN51xx Flash Programmer User Guide (JN-UG-3007)*.

Revision History

Version	Notes
1.0	First internal release
1.1	First external release

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 Laboratories UK Ltd
(Formerly Jennic Ltd)
Furnival Street
Sheffield
S1 4QT
United Kingdom

Tel: +44 (0)114 281 2655
Fax: +44 (0)114 281 2951

www.nxp.com