RELEASE NOTE
JN517x IEEE802.15.4 SDK

JN-SW-4263

Build 1613

NXP Semiconductors

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

www.nxp.com

CONTENTS

1. Software Components

The JN517x IEEE802.15.4 Software Developer's Kit (JN-SW-4263) contains software resources needed to develop IEEE802.15.4 applications for the NXP JN5179, JN5178 and JN5174 wireless microcontrollers. The SDK must be installed on an existing installation of the LPCXpresso Integrated Development Environment (see Section 4).

This SDK release is version 1546 and provides:

- Full MAC layer (802.15.4-2003 / 2006) plus MicroMAC
- 802.15.4 Stack API
- AES (Advanced Encryption Standard) API
- Debug library
- JN517x Integrated Peripherals API
- Production Test API
- Chip and board support for JN517x devices
- Radio Recalibration library
- Jennic packet sniffer binary images for use with Ubiqua packet analyser
- 'New project' wizard for LPCXpresso
- IEEE 802.15.4 Application Template (JN-AN-1211)

Note: The IEEE 802.15.4 Application Template for JN517x (JN-AN-1211) provides a starting point for a new project and can be accessed via the 'New project' wizard in LPCXpresso, as described in the *JN517x LPCXpresso Installation and User Guide (JN-UG-3109)*.

2. Supported Hardware Products

This software release supports the following hardware products:

Chips	Modules	Hardware Kits	Dongles
JN5179-001	JN5179-001-M10 JN5179-001-M13 JN5179-001-M16	JN517x-DK005 JN517x-XK030 JN517x-XK040	OM15021
JN5178-001			
JN5174-001			

3. File Integrity

The following table indicates the MD5 checksum of files included in the release.

File	MD5
JN-SW-4263-IEEE-802.15.4-SDK-v1546.zip	c2b73080ff16be15c80ad0db47a1bcfb

4. Installation

This SDK (JN-SW-4263) must be installed on top of the Eclipse-based LPCXpresso. The SDK is distributed as an Eclipse plug-in archive for LPCXpresso.

4.1 LPCXpresso Installation

LPCXpresso can be obtained from the following NXP web page:

www.nxp.com/lpcxpresso

In order to develop JN517x applications without limitation, we recommend that you purchase the Pro edition of LPCXpresso.

The required version of LPCXpresso for this SDK release is:

LPCXpresso IDE v7.9.2 build 493 (LPCXpresso_7.9.2_493)

Important: *This is the version with which the libraries within the SDK were compiled and verified. Other versions of LPCXpresso may not be compatible with the contents of the SDK and cannot be guaranteed to work or be supported with the JN51xx devices.*

To obtain LPCXpresso and install it on your development machine:

1. If you do not already have a web account with NXP, navigate to www.nxp.com and create an account.
2. Sign in to your NXP web account.
3. Navigate to the page www.nxp.com/lpcxpresso.
4. Select the **Downloads** tab and then click the **Download** button.
5. Check whether the displayed version is the recommended version indicated above:
   - If it is the recommended version, download it.
   - If it is not the recommended version, click **Previous** and then select the recommended version and download it.
6. Launch the LPCXpresso installer and follow the on-screen instructions. On Linux platforms, this MUST be done as root. Full installation details are provided in the *LPCXpresso IDE Installation and Licensing Guide*, available on the **Documentation** tab of the above web page.

4.2 SDK Plug-in Installation

Once LPCXpresso is installed, the SDK plug-in archive can be installed as follows:

1. Start LPCXpresso (on Linux platforms, this MUST be done as root).
2. Navigate to the drop-down menu option **Help**, which can be found at the top of the IDE options.
3. In the drop-down menu, select **Help > Install New Software**.
4. In the pop-up dialog box, click the **Add** button.
5. In the resulting **Add Repository** pop-up dialog box, within the **Name** field enter “NXP IEEE 802.15.4 SDK”. Then click the **Archive** button.

6. In the resulting **Repository Archive** pop-up window, navigate to the location where the **JN-SW-4263-IEEE-802.15.4-SDK-v1613.zip** is present, then click the **OK** button.
7. This will populate the plug-ins list as follows:

   NXP RFCS JN-SW-4263 Software Development Kit

   NXP RFCS JN51xx Flash Programmer

   NXP RFCS Terminal
8. Select all the above by clicking the **Select All** button. Once all the options are selected, click the **Finish** button.
9. Follow all the subsequent instructions, and accept the terms and conditions and disclaimers.

5. Release Details

5.1 New Features and Changes

The following new features are included in this release:

Internal ID	Description
Eclipse Flash Programmer Plug-In	
-	Added identification of minor JN517x chip ID revision

5.2 Bug Fixes

None

5.3 Modifications Required

None

5.4 Known Issues

5.4.1 Link Time Optimisation Problem

In some instances when using Link Time Optimisation (LTO) to reduce code size, it has been observed that optimisation fails when a private variable (i.e. local to a file) exists with the same name as a global assembler symbol in another file. The result is that the symbol defined in the assembler file is not linked with a corresponding extern in a C file because it clashes with the name of the static variable. A linker error occurs, reporting an undefined reference in the file containing the extern reference.

The workaround is to disable LTO for the file containing the extern by specifying the flags as part of the build instructions, e.g. to switch off LTO for file “problem_file.o”, add the following to the makefile:

```
problem_file.o:     CFLAGS += –fno–lto
```

Alternatively, one or other symbol can be renamed so that they no longer clash.

To switch off LTO for ALL files, the flag DISABLE_LTO=1 can be used, but this will result in larger overall binary sizes. The section in the makefile is shown below:

```
# Default (unless debugging) is to compile & link using link time
optimisation, but allow it to be disabled by setting DISABLE_LTO=1
ifneq ($(DISABLE_LTO), 1)
CFLAGS += -flto -ffat-lto-objects
LDFLAGS += -flto
else
CFLAGS += -fno-lto
LDFLAGS += -fno-lto
Endif
```


6. Migrating from JN516x Family of Devices

This section provides guidance on migrating software applications from the NXP JN516x family to the JN517x family. This migration is mainly concerned with understanding the hardware changes that have been made between the two device families. The software development environment remains the same, as it is based on the Eclipse IDE, although the IDE is the NXP LPCXpresso package for JN517x and the 'BeyondStudio for NXP' package for JN516x. Since the two environments are the same, there should be no need to change the application support apart from recompiling the source to create binaries for the new device.

6.1 Increased On-chip Flash Size

The internal Flash memory size has been increased to 512 KBytes on the JN5179. The internal Flash memory sizes for the JN5178 and JN5174 are 256 KBytes and 160 KBytes respectively.

6.2 Decreased Boot Flash Size

The decrease in bootloader Flash memory on the JN517x is unlikely to be an issue for the normal user, but those who require changes to the bootloader should be aware that only 8 KBytes of Flash are available for the boot sector. This restriction is likely to make it very difficult to build a bootloader which contains a meaningful amount of debug for development purposes. For reference, the supplied bootloader occupies around 6.6 KBytes.

6.3 Interrupt Priority Levels

There are 15 levels of interrupt priority available on the JN517x. These levels should map directly to those that are available on the JN516x chips, i.e. priority increases from 0 to 15. However, be aware that critical sections can be over-ridden by high-priority interrupts and therefore care needs to be exercised when setting interrupt levels. The threshold for high-priority interrupts which will override critical sections on the JN517x is 12:

- Time-critical non-stack functions which require rapid interrupt responses should use priority levels above 12.
- Stack-based and user functions with looser timing requirements should use priority levels below 12.

6.4 DIO Pin Configuration

The JN517x platform has a more flexible scheme for multiplexing different functionality onto DIO lines than the JN516x series. In order to cope with the increased flexibility, a new Integrated Peripherals API function **vAHI_SetDIOpinMultiplexValue()** is used to configure the functionality of the individual DIO pins, while the JN516x allowed only one alternative functional role on a DIO pin. As a consequence, a number of API functions that were used to enable DIO pins for these alternative roles are no longer relevant. However, new API functions have been introduced that perform the same operations as their JN516x counterparts apart from configuring the DIO lines. The expected behaviour for JN517x is a call to **vAHI_SetDIOpinMultiplexValue()** to route the

peripheral signals to the appropriate DIO pin, followed by a call to the appropriate configure function to set up the peripheral.

The JN516x functions that are no longer relevant contain the word "**…SetLocation**" in the function name. The JN517x equivalents that configure the peripheral but do not route the peripheral signals to DIO pins are indicated by "**…NoneDIO**" in their names.

Selectable pull-downs have been added to the DIO pins on the JN517x, in addition to the selectable pull-ups found on the JN516x. The functions **vAHI_DioSetPullupDirection()** and **u32AHI_DioReadPullupDirection()** have been added to select a pull-up or pull-down when the functionality is enabled using **vAHI_DIOSetPullup()**.

There are a restricted number of DIO pins available on the JN517x compared to the JN516x. These are numbered as DIOs 0-15, 17 and 18.

6.5 Timers

6.5.1 17-bit 32MHz Timer Operation

The Timers on the JN5179 have been modified to work using a 32-MHz source clock and their counters have been extended from 16 bits to 17 bits (with the exception of the IR timer on Timer4). The Timer start functions for all modes have had their period variables promoted from uint16 to uint32 (this will not cause compatibility issues). A new API function **bAHI_Switch32MHzClockForPWM()** has been added to allow the timers to switch to the 32MHz clocking mode and new API functions have been introduced to read the timers in this mode – these functions are indicated by "**Timer17bit**" in the name.

6.5.2 Additional Timers

There are 8 timers on the JN517x that are capable of PWM output, increased from 4 timers on the JN516x. The additional timers are PWM Timers 3-6, also known as Timers 4-7.

PWM Timer 4 is slightly different in that it incorporates an Infra-Red Remote Control function. This functionality has been moved from PWM Timer 2 on the JN5169 device.

In addition, a further dedicated timer has been included for use with the Analogue Peripherals. This timer is named the Analogue Peripheral Timer (APT), and is also known as Timer 8. As a consequence of these additions, further API functions to register interrupt callbacks have been defined for the extra timers (5-8).

6.6 New I²C Block

A new I²C block has been used in the JN517x. As a consequence, the SI interface functions for the JN516x have been superseded by new functions beginning "**I2C**". These functions are detailed in the *JN517x Integrated Peripherals API User Guide (JN-UG-3118)*.

6.7 Tick Timer

The Tick Timer on the JN517x uses the ARM core tick timer, which works differently from the tick timer of the JN516x. The JN517x timer is 24 bits wide and counts down from a reload value which is stored in a reload register. The counter value can be read and also cleared by writing to the counter, but it cannot be explicitly set to a value. The timer can be configured to generate an interrupt and reload the counter with the value stored in the reload register on reaching 0. The Tick Timer can be clocked from a non-dozed version of the processor clock, which can be of frequency 32, 16, 8, 4, 2 or 1 MHz, or a 32kHz clock which means that it can operate much more slowly than the tick timer on the JN516x. All Tick Timer functions have been modified to reflect the new behaviour.

6.8 Endian Switch

With the move to the ARM Cortex M3 processor, the architecture of the processor has moved from Big-Endian to Little-Endian. While the libraries supplied with the JN517x have been ported and tested on the new architecture, user-supplied code will also need to go through this process. Particular care should be taken where code is accessing 8-bit and 16-bit quantities via pointers, since the byte-order within larger structures or variables will have changed. In addition, the values previously returned by the code may no longer be correct. Similarly, pointer manipulations to step through memory and access structure fields may also fail, as the ordering of fields within memory may also have changed.

6.9 Additional Analogue Inputs

An additional two analogue inputs are supported by the ADC of the JN517x, bringing the total number of inputs available for monitoring external signals to six, in addition to internal inputs connected to the on-chip temperature sensor and battery voltage monitor.

7. Related Documentation

The following user documentation supports this software release:

- JN-UG-3024: IEEE 802.15.4 Stack User Guide
- JN-UG-3109: JN517x LPCXpresso Installation and User Guide
- JN-UG-3118: JN517x Integrated Peripherals API User Guide
- JN-AN-1211: IEEE 802.15.4 Application Template for JN517x

Note: The JN-AN-1211 application template is supplied in the SDK and is available through the ‘New project’ wizard in LPCXpresso, as described in the *JN517x LPCXpresso Installation and User Guide (JN-UG-3109)*.

RELEASE HISTORY (Build 1546)

8. Release Details

8.1 New Features and Changes

The following new features are included in this release:

Internal ID	Description
Build Configuration	
-	When linking, now uses arm-none-eabi-gcc-ar instead of arm-none-eabi-ar. This is necessary for builds with LTO (Link-Time Optimisation) enabled and is backwards compatible with non-LTO builds.
-	Build options: • Warning for “unused local typedefs” disabled • For non-LTO builds, now includes flags to explicitly disable LTO
JN517x Integrated Peripherals API	
-	Added new function vAHI_ModuleConfigure() for JN517x module configuration.
lpsw7446	Added new I2C Master functions boAHI_I2CIsMasterTransferComplete() and u32AHI_I2CTransferHasFailed().
lpsw7726	Improved the I2C bus error handling when detecting a NACK to ensure that the boAHI_I2CIsMasterTransferComplete() function correctly completes when a NACK is received.
Production Test API	
-	Production Test API updated to latest version.
Recalibration	
-	Now clears the PHY interrupt on completion of calibration, which is of benefit to systems that do implement non-standard PHY interrupt handling.
Platform	
-	Support files added for ‘DK5’ type development boards kit (supplied in JN517x hardware kits).

8.2 Bug Fixes

The following issues have been fixed in this release:

Internal ID	Description
JN517x Integrated Peripherals API	
-	Wake timer: Fixed bug in u32AHI_WakeTimerCalibrateEnhanced where any interrupt source would terminate the calibration, not just the wake timer interrupt
-	Infrared driver: Fixed vAHI_InfraredDisable so that it disables the IR timer

8.3 Modifications Required

None

8.4 Known Issues

8.4.1 Link Time Optimisation Problem

In some instances when using Link Time Optimisation (LTO) to reduce code size, it has been observed that optimisation fails when a private variable (i.e. local to a file) exists with the same name as a global assembler symbol in another file. The result is that the symbol defined in the assembler file is not linked with a corresponding extern in a C file because it clashes with the name of the static variable. A linker error occurs, reporting an undefined reference in the file containing the extern reference.

The workaround is to disable LTO for the file containing the extern by specifying the flags as part of the build instructions, e.g. to switch off LTO for file “problem_file.o”, add the following to the makefile:

```
problem_file.o:     CFLAGS += -fno-lto
```

Alternatively, one or other symbol can be renamed so that they no longer clash.

To switch off LTO for ALL files, the flag DISABLE_LTO=1 can be used, but this will result in larger overall binary sizes. The section in the makefile is shown below:

```
# Default (unless debugging) is to compile & link using link time
optimisation, but allow it to be disabled by setting DISABLE_LTO=1
ifneq ($(DISABLE_LTO), 1)
CFLAGS += -flto -ffat-lto-objects
LDFLAGS += -flto
else
CFLAGS += -fno-lto
LDFLAGS += -fno-lto
Endif
```


RELEASE HISTORY (Build 1381)

9. Release Details

9.1 New Features and Changes

This release of JN-SW-4263 is the first public release of the IEEE 802.15.4 SDK in support of the JN517x family of devices. As such there are no new features or changes from previous versions.

9.2 Bug Fixes

None

9.3 Modifications Required

None

9.4 Known Issues

Link Time Optimisation problem.

In some instances when using Link Time Optimisation (LTO) to reduce code size it has been observed that optimisation fails when a private variable (ie local to a file) exists with the same name as a global assembler symbol in another file. The result is that the symbol defined in the assembler file is not linked with a corresponding extern in a C file because it clashes with the name of the static variable. A linker error occurs, reporting an undefined reference in the file containing the extern reference.

The workaround is to disable LTO for the file containing the extern by specifying the flags as part of the build instructions, e.g. to turn off LTO for file “problem_file.o” add the following to the makefile:

```
problem_file.o:     CFLAGS += -fno-lto
```

or by renaming one or other symbol so they no longer clash

To turn off LTO for ALL files, the flag DISABLE_LTO=1 can be used, but this will result in larger overall binary sizes. The section in the makefile is shown below:

```
# Default (unless debugging) is to compile & link using link time
# optimisation, but allow it to be disabled by setting
DISABLE_LTO=1
ifneq ($(DISABLE_LTO), 1)
CFLAGS += -flto -ffat-lto-objects
LDFLAGS += -flto
endif
```