



# Application Note: JN-AN-1264

## Temperature-Dependent Operating Guidelines

When using the NXP wireless microcontroller in environments with significant temperature variation, periodic recalibration of the radio is recommended. This Application Note describes the software functions that perform these operations.

## 1 Introduction

The following devices

- JN5189(T)/JN5188(T)
- QN9090(T)/QN9030(T)
- K32W061/K32W041

feature an integrated radio, which is calibrated at start-up for optimum performance. In operating environments with a significant variation in temperature (e.g. greater than 20°C) due to diurnal or ambient temperature variation, it is recommended to recalibrate the radio to maintain performance.

This Application Note describes a set of software functions which measure temperature using the on-chip temperature sensor and trigger a recalibration if there has been a significant temperature change since the previous calibration.

### 1.1 Contents

<b>1 Introduction</b>	<b>1</b>
1.1 Contents	1
<b>2 Application Scenarios</b>	<b>2</b>
<b>3 Radio Driver Functional Description</b>	<b>3</b>
3.1 Requirements	3
3.1.1 Initial Calibration	3
3.1.2 Temperature Updates	3
3.1.3 Recalibration	3
3.2 Application Changes	4
<b>4 Radio Recalibration Software Functions</b>	<b>6</b>
4.1 Radio Recalibration Function	6
void vRadio_Temp_Update(int16_t s16Temp);	6
void vRadio_Recal(void);	6
<b>5 Including the Software Functions</b>	<b>7</b>
5.1 Function Header File	7
5.2 Modifying the Application	7

## 2 Application Scenarios

Due to the wide variation in applications for the devices, it is desirable for the developer to identify how rapidly the temperature may vary in the application operating environment. The causes of temperature variation may range from slow diurnal (Day/Night) changes (e.g. equipment located outdoors) to rapid changes when co-located equipment is switched on (e.g. lighting fixture). The rate of application data transmission and loading on the network may also be taken into account when determining the checking period for the calibration.

Once the temperature range and rate of change of temperature have been identified, the time-period for checking the calibration may be estimated. This time-period should be chosen to be less than the time taken for the ambient temperature of the chip to change by the recalibration range. For example, for an environment with temperature variation of 80°C over a 12-hour period, changing by 20°C in 3 hours, and so the recalibration check-time period would be set to 3 hours. In practice, a more frequent call may be made.

In lighting applications, the lamp fixtures can reach temperatures well above 100°C, and therefore recalibration will be required. The application program will know when the lamp is turned on, off or if the lamp has a dimmer function, when the light level is changed. In these circumstances, the temperature sensor should be checked more frequently than when the lamp is in a steady state.

## 3 Radio Driver Functional Description

### 3.1 Requirements

- The application is responsible for reading the temperature, as the application may also want to access to the ADC
- Radio recalibration must be performed when the MAC is not using radio
- Temperature reading and MAC being idle are not co-ordinated with one another

The basic mechanism works as follows:

#### 3.1.1 Initial Calibration

Initial calibration is performed based on input temperature  $T_c$  passed from the application during initialisation. Once calibration has completed then the temperature is stored in variable  $T_{cal}$  and clears a variable (Cal-flag) that is used to indicate if calibration is required

#### 3.1.2 Temperature Updates

The update function stores the new temperature  $T_c$  in variable  $T_c$ . If the difference between  $T_c$  and variable  $T_{cal}$  is beyond the required threshold, sets variable Cal-flag

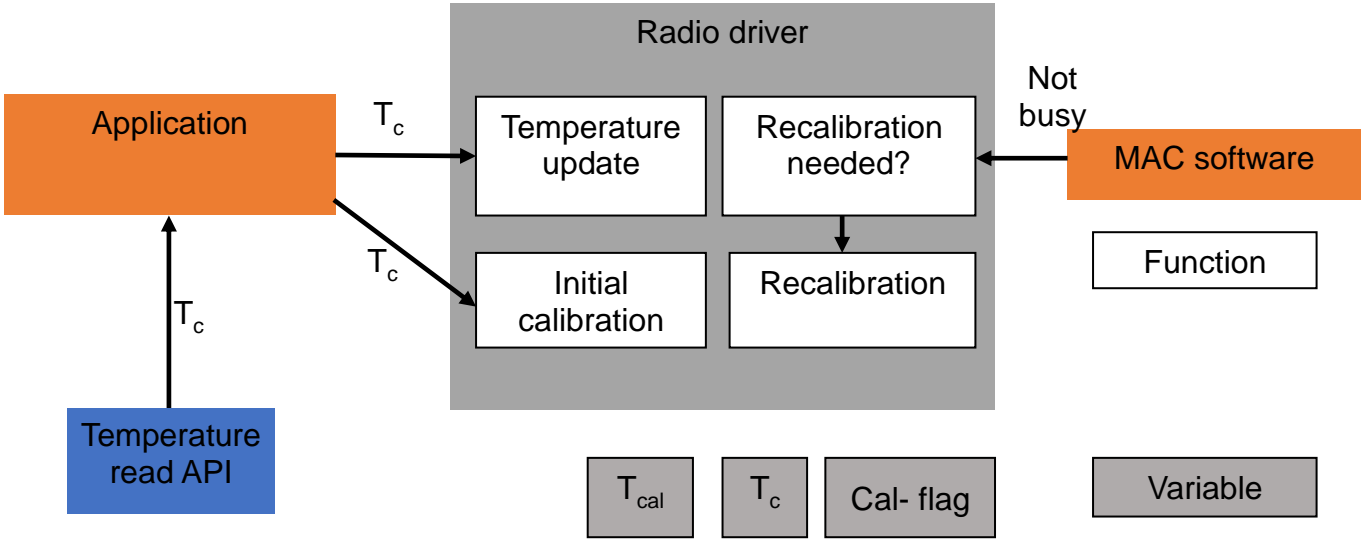
#### 3.1.3 Recalibration

When the radio changes state (between TX and RX or from idle to either RX or TX), the Cal-flag is checked. If the Cal-flag is clear, then no recalibration is performed. If the Cal-flag is set, then the recalibration function is called. This function performs the calibration based on temperature from variable  $T_c$ . Once complete, the function copies the temperature value from variable  $T_c$  to variable  $T_{cal}$  and clears variable Cal-flag.

If the radio calibration callback *bRadioCB\_WriteNVM* has been implemented in the application, then the temperature will be stored with the calibration data. This allows the radio to re-load these calibration values without requiring a new calibration if the temperature is within this defined range (+-20deg).

In BLE applications, the radio is recalibrated before the device goes to sleep and the result saved as normal. If the temperature is within the range on wakeup then no recalibration is required.

This procedure is shown in the diagram below:



### 3.2 Application Changes

1. The application should read the current temperature  $T_c$  at startup. If this is not done, the radio is calibrated the temperature will not be stored with the results and further recalibration will be required.
2. Temperature  $T_c$  will then be passed to radio driver during initialisation. The radio driver performs the calibration and stores temperature  $T_c$  as  $T_{cal}$ , the temperature when last calibration was performed.
3. Current temperature  $T_c$  should be read by the application every  $x$  seconds. The period will vary depending upon how much temperature variation anticipated by the application.
4. Temperature  $T_c$  is passed to radio driver; radio driver compares  $T_c$  to temperature  $T_{cal}$  and sets internal flag if new calibration is needed
5. MAC calls into radio driver when it is between activities; this may happen many times per second (busy network) or as rarely as once every 15 seconds (quiet network, ZigBee ping); radio driver can perform calibration if needed, and if so, it updates  $T_{cal}$  to match  $T_c$ . If no calibration needed, radio driver exits function immediately
6. Repeat steps 3 to 5 whilst running

## 4 Radio Recalibration Software Functions

### 4.1 Radio Recalibration Function

**void vRadio\_Temp\_Update(int16\_t s16Temp);**

**int16\_t s16Temp:** Temperature expressed in half of degree C (2's complement 16-bit value)

- For example, 40 (or 0x28) for 20 degree Celsius or -40 (0xFFD8) for -20 degree Celsius

This function provides the radio driver with the current temperature value. If no temperature has provided, radio driver assumes 25°C but will not store this temperature with the results. The recalibration is initiated automatically by the MiniMac when it changes state. The recalibration can be called manually by calling vRadio\_Recal(). If calling manually, you must ensure that the radio is not in use. The vRadio\_Temp\_Update() function can be called repeatedly as the temperature varies even if no radio recalibration has taken place since the last call (the radio did not change state as no packet was received or transmitted). The next recalibration will simply use the latest temperature value as the current temperature.

**void vRadio\_Recal(void);**

Manually perform the radio recalibration. Radio recalibration must be performed when the MAC is not using radio. This cannot be guaranteed when the Zigbee stack is running but may be known in other applications.

## 5 Including the Software Functions

### 5.1 Function Header File

The recalibration functions are included in the library **libRadio.a** which is supplied as part of the SDK libraries, as is the header file **radio.h**.

To access these library functions, you are required to include the header file in the application source code. To do this, add **#include "radio.h"** to the application source code. No modifications will be required to the makefile.

### 5.2 Modifying the Application

Make the following additions in your application code to implement periodic recalibration:

1. Read the temperature at startup and call `vRadio_Temp_Update()` before the radio is initialised.
2. Periodically, read the temperature and call `vRadio_Temp_Update()` from within your main loop or by using a timer.

## Revision History

Version	Notes
1.0	First release

### How To Reach Us

#### Home Page:

[nxp.com](http://nxp.com)

#### Web Support:

[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.



NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, UMEMS, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V.

All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2019.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

All rights reserved.

Date of release: 11/2019

Document identifier: JN-AN-1264