LPRF Board API Reference Manual

JN-RM-2003
Revision 2.3
18 December 2013

Contents

About this Manual

This manual provides a detailed reference for the NXP Low-Power Radio Frequency (LPRF) Board Application Programming Interface (API), which can be used by an application to interact with the resources of the boards supplied in an NXP JN51xx evaluation kit. The LPRF Board API consists of C functions that can be used in an application that runs on a JN516x, JN514x or JN5139 microcontroller which resides on a module attached to the board.

Organisation

This manual consists of 7 chapters, as follows:

- Chapter 1 introduces the LPRF Board API.
- Chapter 2 describes the functions for controlling the light sensor.
- Chapter 3 describes the functions for controlling the temperature and humidity sensors.
- Chapter 4 describes the functions for controlling the LEDs.
- Chapter 5 describes the functions for controlling the buttons.
- Chapter 6 describes the functions for controlling the LCD panel.
- Chapter 7 describes the functions for controlling the Generic Expansion Board.

Conventions

Files, folders, functions and parameter types are represented in **bold** type.

Function parameters are represented in *italics* type.

Code fragments are represented in the `Courier New` typeface.

This is a **Tip**. It indicates useful or practical information.

This is a **Note**. It highlights important additional information.

*This is a **Caution**. It warns of situations that may result in equipment malfunction or damage.*

Acronyms and Abbreviations

API	Application Programming Interface
LED	Light Emitting Diode
LCD	Liquid Crystal Display
LPRF	Low-Power Radio Frequency
PWM	Pulse Width Modulation
SDK	Software Developer's Kit

Related Documents

JN-UG-3066	JN51xx Integrated Peripherals API User Guide (JN5139/JN514x)
JN-UG-3087	JN516x Integrated Peripherals API User Guide
JN-RM-2007	DR1020 Controller Board Reference Manual
JN-RM-2008	DR1021 Sensor Board Reference Manual
JN-RM-2029	DR1047 Controller Board Reference Manual
JN-RM-2030	DR1048 Sensor Board Reference Manual
JN-RM-2063	Carrier Board and Expansion Boards Reference Manual

Support Resources

To access online support resources such as SDKs, Application Notes and User Guides, visit the Wireless Connectivity TechZone:

www.nxp.com/techzones/wireless-connectivity

For JN514x/JN5139 resources, visit the NXP/Jennic web site: **www.nxp.com/jennic**

Trademarks

"JenNet" and "JenNet-IP" are trademarks of NXP B.V..

All trademarks are the property of their respective owners.

1. Introduction

The Low-Power Radio Frequency (LPRF) Board Application Programming Interface (API) contains C functions that can be used by an application which runs on a board from an NXP JN51xx evaluation kit, in order to interact with the on-board resources (e.g. temperature sensor, buttons, LEDs). The API can be used with a board based around the JN516x, JN514x or JN5139 microcontroller.

The LPRF Board API provides a thin layer above the registers used to control the board components. Several register accesses are encapsulated into a single function call. The API therefore allows use of the on-board components without a detailed knowledge of their operation.

Note: This manual does not describe the API for interacting with features found on the JN51xx chip itself (such as the UARTs and DACs). The API for interfacing with the on-chip resources is detailed in the appropriate *Integrated Peripheral API User Guide*: *JN-UG-3066* for the JN514x/JN5139 and *JN-UG-3087* for the JN516x.

1.1 Software Installers

The LPRF Board API is supplied in the following installers of NXP Software Developer's Kits (SDKs):

- JN-SW-4030: SDK Libraries for JN5139
- JN-SW-4040: SDK Libraries for JN5148-001 and JN5142-001
- JN-SW-4051: JenNet-IP SDK for JN5148-J01 and JN5142-J01
- JN-SW-4060: JN516x ZigBee RF4CE SDK
- JN-SW-4062: JN516x ZigBee Light Link SDK
- JN-SW-4064: JN516x ZigBee Smart Energy SDK
- JN-SW-4065: JN516x JenNet-IP SDK

For SDK installation instructions, see the relevant guide: JN-UG-3064 (JN516x/JN514x) or JN-UG-3035 (JN5139).

Separate header files are provided for different groups of functions in the LPRF Board API. The relevant header files are referenced in the remaining chapters of this manual.

1.2 Board Types

NXP evaluation kit boards of the following types can be used with the Board API:

- **DR1048 and DR1021 Sensor boards:** Each contains LEDs, buttons, a light sensor and a temperature/humidity sensor.
- **DR1047 and DR1020 Controller boards:** Contain similar resources to the sensor boards with the addition of an LCD screen.
- **DR1174 Carrier Board fitted with DR1175 (Lighting/Sensor Expansion Board), DR1199 (Generic Expansion Board) or DR1215* (LCD Expansion Board):** Each combination contains LEDs, a light sensor and a temperature/humidity sensor.

The board hardware is fully detailed in the Reference Manual for the particular board type (see "Related Documents" on page 8).

* The LCD Expansion Board DR1215 replaces DR1201, which is now obsolete. However, the Board API supports both boards.

1.3 API Organisation and Location

The functions of the LPRF Board API are divided into groups which are described in separate chapters of this manual - they are:

- Light Sensor Functions (Chapter 2)
- Temperature and Humidity Sensor Functions (Chapter 3)
- LED Functions (Chapter 4)
- Button Functions (Chapter 5)
- LCD Screen Functions (Chapter 6)
- Generic Expansion Board (DR1199) Functions (Chapter 7)

The relevant header files for the functions are referenced at the start of each chapter. They can be found in the following locations, depending on the evaluation kit and SDK:

- For a JN51xx evaluation kit that requires the SDK Libraries supplied in the installer JN-SW-4040 or JN-SW-4030, the API header files are located in:

  .../Platform/DK2/Include

  These SDKs are relevant to the 'DK2' type boards: DR1020, DR1021, DR1047 and DR1048 (see Section 1.2).
- For the JN516x-EK001 or JenNet-IP EK040 evaluation kit, which requires one of the other SDKs (see Section 1.1), the API header files are located in:

  .../Platform/DK4/Include

  These SDKs are relevant to the 'DK4' type boards: DR1174 fitted with DR1175, DR1199 or DR1215 (see Section 1.2).
- Header files that contain functions common to all boards are located in:

  .../Platform/Common/Include

2. Light Sensor Functions

The chapter describes the functions that can be used to interact with the ambient light sensor on a board. This sensor provides an indication of the level of light falling on the board. The functions are defined in the header file **AlsDriver.h** which is located in the folder **.../Platform/DKx/Include** (see Section 1.3).

The TAOS TSL2550 ambient light sensor used on the boards contains two photo-diodes connected to two separate channels:

- The Channel 0 photo-diode is sensitive to both visible and infra-red light.
- The Channel 1 photo-diode is primarily sensitive to infra-red light.

The light-levels obtained from the two channels (*Ch0* and *Ch1*) can be combined to approximate the response of the human eye, which is sensitive to visible light but insensitive to infra-red light. This adjusted light-level (in units of lux) is given by:

$$Lightlevel = 0.39 \times (Ch0 - Ch1) \times e^{0.181R^2}$$

where *R* = *Ch1*/(*Ch0* - *Ch1*).

The result of this calculation allows a second device to be controlled to match the sensitivities of the human eye, e.g. to control the back-light required for an LCD screen in the given ambient lighting conditions.

Typical channel values and the corresponding light-levels for different light sources are summarised in the following table:

Light Source Type	Ch0 Value	Ch1 Value	Light-level (lux)
Fluorescent	830	70	296
Daylight (in the shade)	900	340	204
Incandescent	960	670	43

Table 1: Typical Light-Level Readings

For more information on the light sensor, refer to the TAOS TSL2550 datasheet.

The light sensor functions are listed below, along with their page references:

Function	**Page**
vALSreset	12
vALSstartReadChannel	13
u16ALSreadChannelResult	14
vALSpowerDown	15

vALSreset

```
void vALSreset(void);
```

Description

This function is used to initialise the ambient light sensor and must be called before using any other light sensor function.

Parameters

None

Returns

None

vALSstartReadChannel

void vALSstartReadChannel(uint8 *u8Channel*);

Description

This function can be used to initiate a read on one of the two channels (Channel 0 or Channel 1) of the ambient light sensor.

Tip: In the Home Sensor Demonstration applications (supplied with some evaluation kits and available as Application Notes), only Channel 0 is used and after the first call to this function, the device continually restarts conversions (so there is no need to call it again).

The result from the initiated read can be obtained by calling the function **u16ALSreadChannelResult()**.

Parameters

u8Channel	Channel to be read: 0 - Channel 0 1 - Channel 1

Returns

None

u16ALSreadChannelResult

```
uint16 u16ALSreadChannelResultvoid);
```

Description

This function can be used to obtain the result of the last read of the ambient light sensor initiated by **vALSstartReadChannel()**.

The returned result is a value in the range 0 to 4015.

Note: The results of consecutive reads of Channel 0 and Channel 1 can be used to estimate the visible light-level as perceived by the human eye. This calculation is shown at the beginning of this chapter.

Parameters

None

Returns

Integer value in the range 0 to 4015

vALSpowerDown

```
void vALSpowerDown(void);
```

Description

This function can be used to power down the ambient light sensor unit.

The function is helpful in minimising the current drawn by the board - for example, it should be called on a board which acts as an End Device before entering sleep mode.

Parameters

None

Returns

None

3. Temperature/Humidity Sensor Functions

The chapter describes the functions that can be used to interact with the temperature and humidity sensors on a board:

- Temperature can be measured in the range 0 to 124^{o}C.
- Relative humidity can be measured in the range 0-100%.

These two measurements are actually provided by a single sensor unit on the board.

The functions are defined in the header file **HtsDriver.h** which is located in the folder **.../Platform/DKx/Include** (see Section 1.3).

The temperature and humidity sensor functions are listed below, along with their page references:

Function	Page
vHTSreset	18
vHTSstartReadTemp	19
u16HTSreadTempResult	20
vHTSstartReadHumidity	21
u16HTSreadHumidityResult	22

vHTSreset

```
void vHTSreset(void);
```

Description

This function is used to initialise the combined temperature and humidity sensor, and must be called before using any other temperature and humidity function.

Parameters

None

Returns

None

vHTSstartReadTemp

```
void vHTSstartReadTemp(void);
```

Description

This function can be used to initiate a read of the temperature sensor.

The result from the initiated read can be obtained by calling the function **u16HTSreadTempResult()**.

Parameters

None

Returns

None

u16HTSreadTempResult

```
uint16 u16HTSreadTempResult(void);
```

Description

This function can be used to obtain the result of the last read of the temperature sensor initiated by **vHTSstartReadTemp()**.

u16HTSreadTempResult() blocks until the result is available.

The returned result is a value in the range 0 to 124^{o}C.

Parameters

None

Returns

Integer value in the range 0 to 124

vHTSstartReadHumidity

```
void vHTSstartReadHumidity(void);
```

Description

This function can be used to initiate a read of the relative humidity sensor.

The result from the initiated read can be obtained by calling the function **u16HTSreadHumidityResult()**.

Parameters

None

Returns

None

u16HTSreadHumidityResult

```
uint16 u16HTSreadHumidityResult(void);
```

Description

This function can be used to obtain the result of the last read of the relative humidity sensor initiated by **vHTSstartReadHumidity()**.

u16HTSreadHumidityResult() blocks until the result is available.

The returned result is a value in the range 0-100%.

Parameters

None

Returns

Integer value in the range 0 to 100

4. LED Functions

The chapter describes the functions that can be used to control LEDs on a board. There are functions for two categories of controllable LED:

- General LEDs controlled via JN51xx DIOs
- RGB LEDs controlled via PCA9634 LED driver chip (DR1175 board only)
- White LED cluster (DR1175 board only)

The functions for the above two LED categories are described below in Section 4.1 and Section 4.2 respectively.

4.1 General LED Functions

The functions described in this section can be used to control general-purpose LEDs via JN51xx DIOs:

- A Controller board (DR1047 or DR1020) has 4 controllable LEDs (D0 to D3)
- A Sensor board (DR1048 or DR1021) has 2 controllable LEDs (D0 and D1)

These boards also have a power LED that cannot be controlled by these functions.

Note that the LED functions can also be used with other NXP boards on which the LEDs are similarly connected, e.g. the *USB IEEE 802 15.4 Expansion Board Reference Design (JN-RD-6024)* which has two controllable LEDs.

The functions are defined in the header file **LedControl.h** which is located in the folder **.../Platform/DK2/Include**.

The General LED functions are listed below, along with their page references:

Function	Page
vLedInitRfd	24
vLedInitFfd	25
vLedControl	26

vLedInitRfd

```
void vLedInitRfd(void);
```

Description

This function is used to initialise the two controllable LEDs on a sensor board or carrier board, and must be called before attempting to control the LEDs.

Note that this function must be used with a sensor board irrespective of whether the board will be used as a network Co-ordinator, Router or End Device.

Parameters

None

Returns

None

vLedInitFfd

```
void vLedInitFfd(void);
```

Description

This function is used to initialise the four controllable LEDs on a controller board and must be called before attempting to control the LEDs.

Note that this function must be used with a controller board irrespective of whether the board will be used as a network Co-ordinator, Router or End Device.

Parameters

None

Returns

None

vLedControl

```
void vLedControl(uint8 u8Led, bool_t bOn);
```

Description

This function can be used to control (illuminate or extinguish) an individual LED.

Before using this function, the appropriate LED initialisation function for the board must have been called - **vLedInitRfd()** for a sensor or carrier board, or **vLedInitFfd()** for a controller board.

Parameters

u8Led	LED to be controlled: 0 - D1 (sensor or controller board) or Rx (carrier board) 1 - D2 (sensor or controller board) or Tx (carrier board) 2 - D3 (controller board only) 3 - D4 (controller board only)
bOn	Action to be applied to LED: TRUE - illuminate LED FALSE - extinguish LED

Returns

None

4.2 RGB and White LED Functions (DR1175 only)

The DR1175 Lighting/Sensor Expansion Board features a variable-intensity RGB LED cluster and a variable-intensity White LED cluster.

The RGB LEDs are driven from a PCA9634 LED driver chip on the board. This driver has 8 channels, where channels 0, 1 and 2 control the blue, green and red LEDs, respectively (the remaining 5 channels can be used to drive PWM outputs on the header CN6).

The White LEDs are driven from the JN514x-J01 or JN516x device using the on-chip PWM Timers.

Functions are provided that allow the RGB and White LEDs to be controlled from an application running on a JN516x or JN514x-J01 device on the board. These functions are defined in the header files **LightingBoard.h** and **pca9634.h** which are located in the folder **.../Platform/DK4/Include**.

The LED functions are listed below, along with their page references:

Function	Page
bRGB_LED_Enable	28
bRGB_LED_Disable	29
bRGB_LED_SetLevel	30
bRGB_LED_On	31
bRGB_LED_Off	32
bRGB_LED_SetGroupLevel	33
bPCA9634_Init	34
bPCA9634_SetChannelLevel	35
bPCA9634_SetGroupLevel	36
bWhite_LED_Enable	37
bWhite_LED_Disable	38
bWhite_LED_SetLevel	39
bWhite_LED_On	40
bWhite_LED_Off	41

Caution: *The LEDs on the DR1175 board are very bright at maximum intensity. To avoid damage to your eyes, do not look into them directly for an extended period of time.*

bRGB_LED_Enable

```
bool bRGB_LED_Enable(void);
```

Description

This function is used to initialise the PCA9634 LED driver device.

This function must be called before using any of the other RGB LED functions.

Parameters

None

Returns

TRUE - driver successfully initialised
FALSE - driver failed to initialise

bRGB_LED_Disable

```
bool bRGB_LED_Disable(void);
```

Description

This function is used to disable the PCA9634 LED driver device.

Parameters

None

Returns

TRUE - driver successfully disabled

FALSE - driver failed to disable

bRGB_LED_SetLevel

```
bool bRGB_LED_SetLevel(uint8 u8RedLevel,
                       uint8 u8GreenLevel,
                       uint8 u8BlueLevel);
```

Description

This function can be used to set the RGB LED brightness level in the range 0 (dimmest) to 255 (brightest).

Parameters

u8RedLevel	Red brightness level (0-255)
u8GreenLevel	Green brightness level (0-255)
u8BlueLevel	Blue brightness level (0-255)

Returns

TRUE - success

FALSE - fail

bRGB_LED_On

```
bool bRGB_LED_On(void);
```

Description

This function can be used to switch on the RGB LEDs. The RGB LED brightness levels will be set to the values previously configured using **bRGB_LED_SetLevel()**.

Parameters

None

Returns

TRUE - success

FALSE - fail

bRGB_LED_Off

```
bool bRGB_LED_Off(void);
```

Description

This function can be used to switch off the RGB LEDs.

Parameters

None

Returns

TRUE - success

FALSE - fail

bRGB_LED_SetGroupLevel

```
bool bRGB_LED_SetGroupLevel(uint8 u8Level);
```

Description

This function can be used to apply a global scaling factor (of less than or equal to 1) across all channels of the PCA9634 LED driver device on the DR1175 board. This allows the brightnesses of the LEDs of the RGB cluster to be scaled simultaneously by the same factor. The scaling factor parameter *u8Level* can be set to a value in the range 0 to 255. Once the function has been called, the brightness of each LED becomes *u8Level*/255 of its original brightness (the brightness with no global scaling factor applied). The default value of *u8Level* is 255, corresponding to full brightness. A factor of 0 will extinguish the LEDs.

This feature is useful when the LEDs of the RGB cluster are set to different brightness levels in order to achieve a certain colour. The function allows the brightness of the LED cluster to be adjusted without affecting this colour.

Note that when calling the function multiple times, on each occasion the scaling factor is applied to the original brightness levels of the LED cluster. For example, calling the function first with a scaling factor of 128 will result in a brightness reduction to approximately 50% of the original brightness. Then calling the function again with a scaling factor of 192 will result in a brightness increase to approximately 75% of the original brightness.

Also note that the scaling factor is applied to channels 3-7 of the PCA9634 driver device (that may drive PWM outputs on header CN6), as well as to channels 0-2 used for the RGB cluster.

Parameters

u8Level	Brightness scaling factor in the range 0 to 255

Returns

TRUE - success

FALSE - fail

bPCA9634_Init

```
bool bPCA9634_Init(void);
```

Description

This function is used to initialise the PCA9634 LED driver device and must be called before the RGB LED cluster on the DR1175 board can be controlled using any of the other RGB LED functions.

Note: The function **bRGB_LED_Enable()** is identical to **bPCA9634_Init()** and may be called instead of this one.

Parameters

None

Returns

One of:

TRUE - driver successfully initialised

FALSE - driver failed to initialise

bPCA9634_SetChannelLevel

```
bool bPCA9634_SetChannelLevel(uint8 u8Channel,
                              uint8 u8Level);
```

Description

This function can be used to control one of the RGB LEDs on the DR1175 board by setting the brightness level for the LED, in the range 0 (dimmest) to 255 (brightest).

The function requires the relevant channel number for the LED, which must be 0 (blue), 1 (green) or 2 (red) for the RGB LED cluster. If another channel is specified (3-7) then the level of the relevant output will be set (if connected).

Note: The brightness level for the RGB channels may instead be set using a single call to function **bRGB_LED_SetLevel()**.

Parameters

u8Channel	LED channel to be controlled, in the range 0-7 (where 0, 1, 2 correspond to the blue, green and red LEDs, respectively)
u8Level	Brightness level to which specified LED is to be set, in the range 0-255

Returns

One of:

TRUE - brightness level successfully set

FALSE - brightness level not set

bPCA9634_SetGroupLevel

bool bPCA9634_SetGroupLevel(uint8 *u8Level***);**

Description

This function can be used to apply a global scaling factor (of less than or equal to 1) across all channels of the PCA9634 LED driver device on the DR1175 board. This allows the brightnesses of the LEDs of the RGB cluster to be scaled simultaneously by the same factor. The scaling factor parameter *u8Level* can be set to a value in the range 0 to 255. Once the function has been called, the brightness of each LED becomes *u8Level*/255 of its original brightness (the brightness with no global scaling factor applied). The default value of *u8Level* is 255, corresponding to full brightness. A factor of 0 will extinguish the LEDs.

This feature is useful when the LEDs of the RGB cluster are set to different brightness levels in order to achieve a certain colour. The function allows the brightness of the LED cluster to be adjusted without affecting this colour.

Note that when calling the function multiple times, on each occasion the scaling factor is applied to the original brightness levels of the LED cluster. For example, calling the function first with a scaling factor of 128 will result in a brightness reduction to approximately 50% of the original brightness. Then calling the function again with a scaling factor of 192 will result in a brightness increase to approximately 75% of the original brightness.

Also note that the scaling factor is applied to channels 3-7 of the PCA9634 driver device (that may drive PWM outputs on header CN6), as well as to channels 0-2 used for the RGB cluster.

Note: The function **bRGB_LED_SetGroupLevel()** is identical to **bPCA9634_SetGroupLevel()** and may be called instead of this one.

Parameters

u8Level	Brightness scaling factor in the range 0 to 255

Returns

One of:

TRUE - scaling factor successfully applied

FALSE - scaling factor not applied

bWhite_LED_Enable

```
bool bWhite_LED_Enable(void);
```

Description

This function is used to initialise the PWM Timers and DIO for driving the White LED cluster. This function must be called before using any other White LED cluster function.

Parameters

None

Returns

TRUE - success

FALSE - fail

bWhite_LED_Disable

```
bool bWhite_LED_Disable(void);
```

Description

This function is used to disable control of the White LED cluster.

Parameters

None

Returns

TRUE - success

FALSE - fail

bWhite_LED_SetLevel

bool bWhite_LED_SetLevel(uint8 *u8BrightnessLevel***);**

Description

This function can be used to set the White LED brightness level in the range 0 (dimmest) to 255 (brightest).

Parameters

u8BrightnessLevel	Brightness level (0-255)

Returns

TRUE - success

FALSE - fail

bWhite_LED_On

```
bool bWhite_LED_On(void);
```

Description

This function can be used to switch on the White LED cluster. The White LED brightness level will be set to the value previously configured using **bWhite_LED_SetLevel()**.

Parameters

None

Returns

TRUE - success

FALSE - fail

bWhite_LED_Off

```
bool bWhite_LED_Off(void);
```

Description

This function can be used to switch off the White LED cluster.

Parameters

None

Returns

TRUE - success

FALSE - fail

5. Button Functions

The chapter describes the functions that can be used to interact with the general-purpose buttons on a board:

- A Controller board (DR1047 or DR1020) has 4 buttons (SW1 to SW4)
- A Sensor board (DR1048 or DR1021) has 2 buttons (SW1 and SW2)
- A Carrier board (DR1174) has 1 button (SW1 but labelled DIO8 on the board)

A board may also have power, reset and programming switches that cannot be accessed by these functions.

Note that the button functions can also be used with other NXP boards on which the buttons are similarly connected, e.g. the *USB IEEE 802 15.4 Expansion Board Reference Design (JN-RD-6024)* which has two general-purpose buttons.

The functions are defined in the header file **Button.h** which is located in the folder **.../Platform/DKx/Include** (see Section 1.3).

The button functions are listed below, along with their page references:

vButtonInitRfd

```
void vButtonInitRfd(void);
```

Description

This function is used to initialise the two buttons on a sensor board or the button on a carrier board, and must be called before attempting to access the button(s).

Note that this function must be used with a sensor board irrespective of whether the board will be used as a network Co-ordinator, Router or End Device.

Parameters

None

Returns

None

vButtonInitFfd

```
void vButtonInitFfd(void);
```

Description

This function is used to initialise the four buttons on a controller board and must be called before attempting to access the buttons.

Note that this function must be used with a controller board irrespective of whether the board will be used as a network Co-ordinator, Router or End Device.

Parameters

None

Returns

None

u8ButtonReadRfd

```
uint8 u8ButtonReadRfd(void);
```

Description

This function can be used to obtain the states of the two buttons on a sensor board or the button on a carrier board.

The states (pressed or not pressed) of the buttons are returned in a single bitmap:

- To obtain the state of button SW1 (sensor board or carrier board), the returned bitmap must be logical-ANDed with the mask BUTTON_0_MASK.
- To obtain the state of button SW2 (sensor board only), the returned bitmap must be logical-ANDed with the mask BUTTON_1_MASK.

A non-zero result indicates that the button is pressed and a zero result indicates that it is not pressed.

Note that there is no de-bounce circuit or algorithm employed. It is possible and legitimate for both buttons on a sensor board to be pressed at the same time.

Before using this function, the button initialisation function **vButtonInitRfd()** must have been called.

Parameters

None

Returns

Bitmap containing the states of the buttons (see above)

u8ButtonReadFfd

```
uint8 u8ButtonReadFfd(void);
```

Description

This function can be used to obtain the states of the four buttons on a controller board.

The states (pressed or not pressed) of all four buttons are returned in a single bitmap:

- To obtain the state of button SW1, the returned bitmap must be logical-ANDed with the mask BUTTON_0_MASK.
- To obtain the state of button SW2, the returned bitmap must be logical-ANDed with the mask BUTTON_1_MASK.
- To obtain the state of button SW3, the returned bitmap must be logical-ANDed with the mask BUTTON_2_MASK.
- To obtain the state of button SW4, the returned bitmap must be logical-ANDed with the mask BUTTON_3_MASK.

A non-zero result indicates that the button is pressed and a zero result indicates that it is not pressed.

Note that there is no de-bounce circuit or algorithm employed. It is possible and legitimate for more than one button to be pressed at the same time.

Before using this function, the button initialisation function **vButtonInitFfd()** must have been called.

Parameters

None

Returns

Bitmap containing the states of the four buttons (see above)

6. LCD Screen Functions

The chapter describes the functions that can be used to interact with the LCD screen on a Controller board or LCD Expansion Board.

The board specific functions are defined in the header file **LcdDriver.h** which is located in the folder **.../Platform/DKx/Include** (see Section 1.3).

LCD screen functions that are common to all boards are defined in the header files **LcdDraw.h** and **LcdExtras.h** which are located in the folder **.../Platform/Common/ Include**.

6.1 LCD Screen and Data Display

The LCD screen has a resolution of 64 rows by 128 columns, but is internally arranged so that one byte contains the pixel information for a single column of 8 rows. As a result, the LCD driver is considerably simplified by only allowing the positioning of text or graphics on 8-row boundaries. From now on in this manual, a block of 8 rows will be referred to as a 'character row', with the LCD panel containing 8 character rows. There is no such limitation on the columns.

This principle of data display on the LCD screen is illustrated in the figure below and is described in more detail in Section 6.1.3.

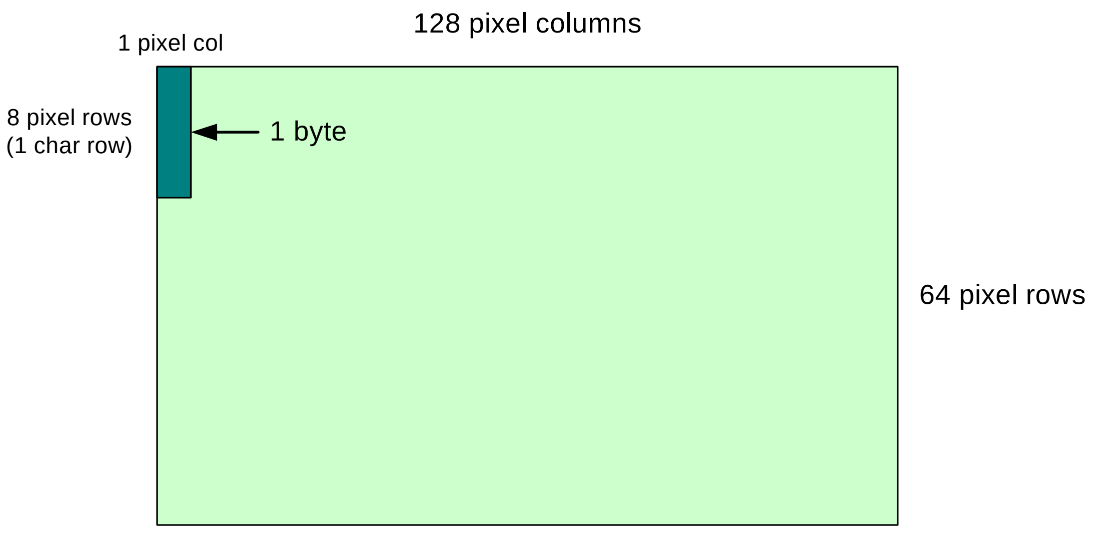

Figure 1: Data Display on LCD Screen

6.1.1 Shadow Memory

The LCD driver uses shadow memory. A screen of information can first be built in the shadow memory and then a command can be issued to update the LCD screen with the contents of shadow memory in one action, thereby improving performance and minimising the possibility of seeing a partially changed screen.

6.1.2 LCD Font

A proportional font is used to display text on the LCD screen. In this font, most characters are 5 pixels wide, although several characters are narrower than this and some special characters are 7 pixels wide. All characters are 8 pixels high. When printing text, there is a blank column before the first character, between each character and after the last character.

The font is mapped approximately to the ASCII character set, although some characters are moved to simplify the font processing. The map is as follows:

ASCII Code	ASCII Character	LCD Font Character
37	‘%’	Percent symbol
38-44	‘&’-‘,’	Full dark moon – full light moon symbols
48-57	‘0’-‘9’	‘0’-‘9’
65-90	‘A’-‘Z’	‘A’-‘Z’
91	‘[‘	Degrees symbol
92	‘\’	Plus symbol
93	‘]’	Minus symbol
94	‘^’	Space
97-122	‘a’-‘z’	‘a’-‘z’

Other characters are displayed as a space.

6.1.3 LCD Bitmaps

Bitmaps are designed to be simple to render and, as such, map to the way that the LCD screen displays pixels. A bitmap can contain any number of columns but must always be a multiple of 8 pixels high. Each bitmap is treated as a 'C' structure consisting of the width of the bitmap in pixels, its height in character rows and a pointer to an array of bytes containing the pixel data (see the structure below).

```
typedef struct
{
    uint8 *pu8Bitmap;
    uint8 u8Width;
    uint8 u8Height;
} tsBitmap;
```

Consider the bitmap shown below, consisting of 16 rows and y columns. Each pixel is represented by a two-letter name, e.g. ab.

		column					
		0	**1**	**2**		**y-2**	**y-1**
row	**0**	aa	ab	ac	..	aw	ax
	1	ba	bb	bc	..	bw	bx
	2	ca	cb	cc	..	cw	cx
		..	..	..	..	..	..
	7	ha	hb	hc	..	hw	hx
	8	ia	ib	ic	..	iw	ix
	9	ja	jb	jc	..	jw	jx
		..	..	..	..	..	..
	15	pa	pb	pc	..	pw	px

The first element in the array of pixel data contains the first column of the top character row of pixels, i.e. pu8Bitmap[0] = (MSB) ha ga fa ea da ca ba aa (LSB).

The second element in the array of pixel data contains the second column of the top character row of pixels, i.e. pu8Bitmap[1] = (MSB) hb gb fb eb db cb bb ab (LSB).

This continues until the end of the first character row, so the final column of the first row is in array element y-1, i.e. pu8Bitmap[y-1] = (MSB) hx gx fx ex dx cx bx ax (LSB).

The first column of the second character row of pixels is the next element in the array, i.e. pu8Bitmap[y] = (MSB) pa oa na ma la ka ja ia (LSB).

This row continues until the final column, which will be in element 2y-1,
i.e. pu8Bitmap[2y-1] = (MSB) px ox nx mx lx kx jx ix (LSB).

If there are further rows, they repeat in the same manner.

6.2 Functions

The LCD screen functions are listed below, along with their page references:

Function	Page
vLcdResetDefault	53
vLcdReset	54
vLcdStop	55
vLcdClear	56
vLcdWriteText	57
vLcdWriteTextRightJustified	58
vLcdWriteInvertedText	59
vLcdWriteBitmap	60
vLcdPlotPoint	61
bLcdGetPixel	62
vLcdDrawLine	63
vLcdDrawCircle	64
vLcdFloodFill	65
vLcdRefreshAll	66
vLcdRefreshArea	67
vLcdContrastLevel	68
u8LcdCalcContrastLevel	69
vLcdBackLightEnable	70
vLcdGrabSpiBus	71
vLcdFreeSpiBus	72
vLcdPowerOff	73
vLcdPowerSavingMode	74
vLcdButtonInit	75
u8LcdButtonRead	76

vLcdResetDefault

```
void vLcdResetDefault(void);
```

Description

This function is used to initialise the LCD panel using default settings for bias and gain, which should give a good level of contrast. The LCD screen is also cleared.

Alternatively, the LCD panel can be initialised with custom settings for bias and gain using the function **vLcdReset()**.

Parameters

None

Returns

None

vLcdReset

void vLcdReset(uint8 *u8Bias*, **uint8** *u8Gain***);**

Description

This function is used to initialise the LCD panel using the specified settings for bias and gain. The LCD screen is also cleared.

Alternatively, the LCD panel can be initialised with the default settings for bias and gain using the function **vLcdResetDefault()**.

Parameters

u8Bias	Bias value to use, in the range 0-3
u8Gain	Gain value to use, in the range 0-3

Returns

None

vLcdStop

```
void vLcdStop(void);
```

Description

This function can be used to power off the LCD screen. The function is normally only used before shutting down the controller board, in order to allow the LCD screen to discharge itself properly.

Parameters

None

Returns

None

vLcdClear

```
void vLcdClear(void);
```

Description

This function can be used to clear the shadow memory of any text or graphics. However, the function does not update the LCD screen itself.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

None

Returns

None

vLcdWriteText

```
void vLcdWriteText(char *pcString,
                   uint8 u8Row,
                   uint8 u8Column);
```

Description

This function can be used to write text to the shadow memory.

The text is left-justified, starting at the row and column specified. No attempt is made to prevent the text from spilling over the end of the specified row and if this occurs, the text will wrap around to the next row.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

***pcString*	Null-terminated text string to display
u8Row	Character row on which to display text (in the range 0-7)
u8Column	Column in which to start displaying text (in the range 0-127)

Returns

None

vLcdWriteTextRightJustified

```
void vLcdWriteTextRightJustified(char *pcString,
                                 uint8 u8Row,
                                 uint8 u8EndColumn);
```

Description

This function can be used to write text to the shadow memory.

The text is right-justified, finishing at the row and column specified. No attempt is made to prevent the text from spilling over the start of the specified row and if this occurs, the text will wrap around to the previous row.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

***pcString**	Null-terminated text string to display
u8Row	Character row on which to display text (in the range 0-7)
u8Column	Column in which to finish displaying text (in the range 0-127)

Returns

None

vLcdWriteInvertedText

```
void vLcdWriteInvertedText(char *pcString,
                           uint8 u8Row,
                           uint8 u8Column);
```

Description

This function can be used to write inverted text (pixel colours reversed) to the shadow memory.

The text is left-justified, starting at the row and column specified. No attempt is made to prevent the text from spilling over the end of the specified row and if this occurs, the text will wrap around to the next row.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

pcString	Null-terminated text string to display
u8Row	Character row on which to display text (in the range 0-7)
u8Column	Column in which to start displaying text (in the range 0-127)

Returns

None

vLcdWriteBitmap

```
void vLcdWriteBitmap(tsBitmap *psBitmap,
                     uint8 u8LeftColumn,
                     uint8 u8TopRow);
```

Description

This function can be used to put the specified bitmap into the shadow memory at the specified screen location. If the bitmap goes past the edge of the display area, it is truncated.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

psBitmap	Pointer to structure containing bitmap information
u8LeftColumn	Left-most column of bitmap (in the range 0-127)
u8TopRow	First character row of bitmap (in the range 0-7)

Returns

None

vLcdPlotPoint

```
void vLcdPlotPoint(uint8 u8X, uint8 u8Y);
```

Description

This function can be used to plot a point in the shadow memory at the pixel with the specified screen coordinates (x,y). If either x or y is out of range, the pixel cannot be plotted.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

u8X	x-coordinate of pixel to be plotted (in the range 0-127)
u8Y	y-coordinate of pixel to be plotted (in the range 0-63)

Returns

None

bLcdGetPixel

```
bool_t bLcdGetPixel(uint8 u8X, uint8 u8Y);
```

Description

This function can be used to obtain the status of the pixel at the specified screen coordinates (x,y) in the shadow memory - that is, whether a point has been plotted at the pixel location. If either x or y is out of range, the function returns FALSE.

Parameters

u8X	x-coordinate of pixel to be checked (in the range 0-127)
u8Y	y-coordinate of pixel to be checked (in the range 0-63)

Returns

TRUE - point plotted at pixel location

FALSE - point not plotted at pixel location or invalid screen location specified

vLcdDrawLine

```
void vLcdDrawLine(uint8 u8x1,
                  uint8 u8y1,
                  uint8 u8x2,
                  uint8 u8y2);
```

Description

This function can be used to draw a straight line between two specified screen locations in the shadow memory.

The end-points of the line are specified in terms of coordinates (x1,y1) and (x2,y2). The line is clipped if either of these points lies outside the screen area.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

u8x1	x-coordinate of first end-point of line (in the range 0-127)
u8y1	y-coordinate of first end-point of line (in the range 0-63)
u8x2	x-coordinate of second end-point of line (in the range 0-127)
u8y2	y-coordinate of second end-point of line (in the range 0-63)

Returns

None

vLcdDrawCircle

void vLcdDrawCircle(int *Xc*, **int** *Yc*, **int** *Radius***);**

Description

This function can be used to draw the circle with specified centre and radius in the shadow memory.

The centre of the circle is specified in terms of coordinates (Xc,Yc). The circle is clipped if any points on its circumference lie outside the screen area.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

Xc	x-coordinate of circle centre (in the range 0-127)
Yc	y-coordinate of circle centre (in the range 0-63)
Radius	Radius of circle, in pixels

Returns

None

vLcdFloodFill

```
void vLcdFloodFill(int i32x, int i32y);
```

Description

This function can be used to perform a ‘flood fill’ of a screen region in shadow memory - that is, all pixels within the region will be plotted (black).

The region to be filled must have been pre-defined as a closed boundary - for example, a circle created using the **vLcdDrawCircle()** function. **vLcdFloodFill()** requires any starting point to be specified inside but not on the boundary. If the boundary is not closed, the fill will 'leak' and corrupt the display.

To update the LCD screen from the shadow memory, you can use the function **vLcdResfreshAll()** or **vLcdRefreshArea()**.

Parameters

i32x	x-coordinate of starting point (in the range 0-127)
i32y	y-coordinate of starting point (in the range 0-63)

Returns

None

vLcdRefreshAll

```
void vLcdRefreshAll(void);
```

Description

This function can be used to copy the contents of shadow memory to the LCD panel and therefore to refresh the entire screen contents. This copy takes approximately 4.5 ms.

Alternatively, to update a rectangular portion of the LCD screen from the shadow memory, use the function **vLcdRefreshArea()**.

Parameters

None

Returns

None

vLcdRefreshArea

```
void vLcdRefreshArea(uint8 u8LeftColumn,
                     uint8 u8TopRow,
                     uint8 u8Width,
                     uint8 u8Height);
```

Description

This function can be used to copy the contents of the specified rectangle in shadow memory to the appropriate part of the LCD screen without disturbing the rest of the display.

The rectangular screen area to refresh is specified in terms of the left-most column, the top row, the width and the height of the rectangle.

Alternatively, to update the entire LCD screen from shadow memory, use the function **vLcdRefreshAll()**.

Parameters

u8LeftColumn	Left-most column of rectangle (in the range 0-127)
u8TopRow	First character row of rectangle (in the range 0-7)
u8Width	Number of columns in width of rectangle (in the range 1-128)
u8Height	Number of rows in height of rectangle (in the range 1-8)

Returns

None

vLcdContrastLevel

```
void vLcdContrastLevel(uint8 u8Gain);
```

Description

This function can be used to control the contrast level of the LCD screen.

Parameters

u8Gain	Contrast level (in the range 0-15 maximum)

Returns

None

u8LcdCalcContrastLevel

```
uint8 u8LcdCalcContrastLevel(void);
```

Description

This function can be used to calculate the correct contrast level required for the LCD screen. The correct LCD contrast level may then be subsequently set by using the return value of this function as the input paramater value in the call to function **vLCDContrastLevel()**.

Parameters

None

Returns

Contrast level (in the range 0-15)

vLcdBackLightEnable

```
void vLcdBackLightEnable(uint8 u8Status);
```

Description

This function can be used to control the LCD back-light.

Parameters

u8Status	0 - Disable (switch off) the LCD back-light 1 - Enable (switch on) the LCD back-light Note, all other values will have no effect.

Returns

None

vLcdGrabSpiBus

```
void vLcdGrabSpiBus(void);
```

Description

This function can be used to take control of the SPI bus for the LCD panel.

Parameters

None

Returns

None

vLcdFreeSpiBus

```
void vLcdFreeSpiBus(void);
```

Description

This function can be used to free control of the SPI bus for the LCD panel.

Parameters

None

Returns

None

vLcdPowerOff

```
void vLcdPowerOff(void);
```

Description

This function can be used to power off the LCD screen. The function is normally only used before shutting down the controller board, in order to allow the LCD screen to discharge itself properly.

Parameters

None

Returns

None

vLcdPowerSavingMode

```
void vLcdPowerSavingMode(bool_t bSelectMode);
```

Description

This function can be used to control the LCD screen power-saving mode.

Parameters

bSelectMode	TRUE - Power-saving mode ON
	FALSE - Power-saving mode OFF

Returns

None

vLcdButtonInit

```
void vLcdButtonInit(void);
```

Description

This function can be used to initialise use of the buttons on the LCD board.

Parameters

None

Returns

None

u8LcdButtonRead

```
uint8 u8LcdButtonRead(void);
```

Description

This function can be used to read the current state of the buttons on the LCD board. The function **vLcdButtonInit()** must first be called before calling this function.

Parameters

None

Returns

A bitmap that represents the status of the LCD buttons (1 indicates a button depress):

bit 0-3:	SW1-SW4 status
bit 4-7:	unused

7. Generic Board (DR1199) Functions

This chapter describes the functions that can be used to interact with the specific resources provided by the Generic Expansion Board (DR1199) - this board is described in *Carrier Board and Expansion Boards Reference Manual (JN-RM-2063)*.

The Generic Expansion Board provides the following resources:

- 50kohm potentiometer (R4)
- 4 x push-button (SW1-SW4)
- 3 x LED (D1-D3)

The potentiometer is connected in series with a 33kohm resistor (R8) to form a potential divider between the supply and ground. The voltage across the potentiometer may be measured by the JN51xx device using the on-chip ADC.

It is possible to fit a fourth LED (D4) to the board, but for correct operation this will require removal of the fourth push button (SW4) because these components share the same IO pin.

The functions described in this chapter are provided to simplify the use of these hardware resources and resolve the device IO connections for this board - for example, SW1 on this board is not assigned to the same DIO pin as on the DR1047 board.

The functions are defined in the header file **GenericBoard.h** which is located in the folder **.../Platform/DK4/Include**.

The functions are listed below, along with their page references:

bPotEnable

```
bool_t bPotEnable(void);
```

Description

This function is used to configure the IO and on-chip ADC for reading the voltage across the potentiometer on the Generic Expansion Board. This function should be called before using any other potentiometer function.

Parameters

None

Returns

TRUE - success

FALSE - fail

u16ReadPotValue

```
uint16 u16ReadPotValue(void);
```

Description

This function can be used to read the voltage across the potentiometer on the Generic Expansion Board as measured by the on-chip ADC.

Before using this function, the potentiometer initialisation function **bPotEnable()** must have been called.

Parameters

None

Returns

ADC reading of voltage measured across potentiometer:

10-bit value for JN516x

12-bit value for JN5148 and JN5139

8-bit value for JN5142

bPotDisable

```
bool_t bPotDisable(void);
```

Description

This function can be used to disable the IO and on-chip ADC from reading the voltage across the potentiometer on the Generic Expansion Board.

Before using this function, the potentiometer initialisation function **bPotEnable()** must have been called.

Parameters

None

Returns

TRUE - success

FALSE - fail

vGenericButtonInit

```
void vGenericButtonInit(void);
```

Description

This function is used to configure the IO for reading the status of the push-buttons on the Generic Expansion Board. This function should be called before using any other push-button function for this board.

Parameters

None

Returns

None

u8GenericButtonRead

```
uint8 u8GenericButtonRead(void);
```

Description

This function can be used to read the status of the push-buttons on the Generic Expansion Board.

Before using this function, the push-button initialisation function **vGenericButtonInit()** must have been called.

Parameters

None

Returns

A bitmap of the push-button status (a bit set to 1 indicates a push-button depress):

bit 0-3:	SW1-SW4 status
bit 4-7:	unused

vGenericLEDInit

```
void vGenericLEDInit(void);
```

Description

This function is used to configure the IO for driving the LEDs on the Generic Expansion Board. This function should be called before using any other LED function for this board.

Parameters

None

Returns

None

vGenericLEDSetOutput

```
void vGenericLEDSetOutput(uint8 u8LEDBitmap, bool_t bOn);
```

Description

This function can be used to control the status of the LEDs on the Generic Expansion Board.

Before using this function, the LED initialisation function **vGenericLEDInit()** must have been called.

Parameters

u8LEDBitmap	Bitmap of LEDs to control (bit 0-3: D1-D4, bit 4-7: unused)
bOn	Status that the LEDs should be set to (TRUE: On, FALSE: Off)

Returns

None

Revision History

Version	Date	Comments
1.0	12-Sep-2005	First release
1.1	14-Nov-2005	Updated document style
1.2	10-Mar-2006	Removed references to specific evaluation kit
1.3	06-Oct-2006	Name of API changed from Evaluation Kit Library to Board API
1.4	08-Jan-2007	Updated for JN513x chip series
2.0	22-Nov-2010	Manual re-worked in new template and API re-named the Low-Power Radio Frequency (LPRF) Board API. JN5148 chip added and minor corrections made
2.1	18-Sept-2012	Updated to include DR1174 and DR1175 boards. JN5142 chip also added and other minor changes made
2.2	17-Dec-2012	Updated to include DR1199 and DR1201 boards. JN516x chip family also added and SDK information updated.
2.3	18-Dec-2013	LCD Expansion Board DR1201 replaced with DR1215

Important Notice

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

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

www.nxp.com

For online support resources, visit the Wireless Connectivity TechZone:

www.nxp.com/techzones/wireless-connectivity