Radio Driver

Reference guide

Introduction

This application note is API reference guide for Radio Driver and MCU timer driver.

The Radio Driver supports LoRa®-Based modulation technology and (G)FSK modulation.

Target Device

RL78/G23 (R7F100GLG, R7F100GSN) + RF (Semtech SX1261/SX1262)

RL78/G14 (R5F104ML) + RF (Semtech SX1261/SX1262)

Note: Radio driver uses MCU peripherals (see 1.3 Resource Usage) and has radio interface defines (see 2.1 Defines), please confirm your device configuration.

Contents

1. Overview

This application note contains API of Radio driver and Timer functions.

1.1 Radio Driver Component Diagram

Figure 1.1 Radio driver and related components overview shows a block diagram of Radio driver and related components overview.

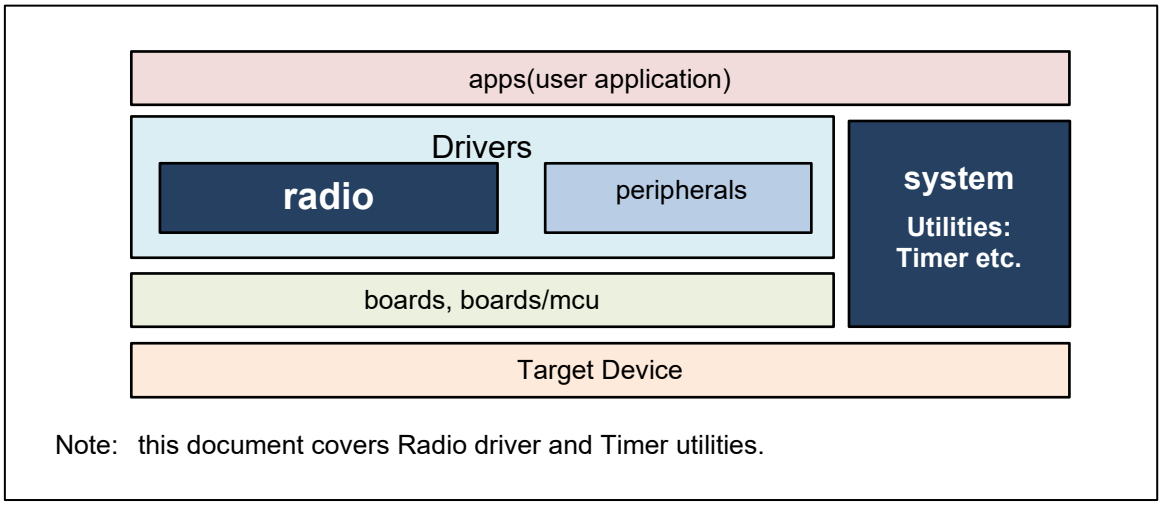

Figure 1.1 Radio driver and related components overview

1.2 Directories (informative)

Table 1.1 shows a basic concept of what kind of codes each directory includes. This is just informative.

Table 1.1 Directories

Directories	Description
apps	Application code
boards	Board specific codes
boards/mcu	MCU drivers
radio	Radio driver
system	Utility APIs etc.
peripherals	Peripheral drivers

1.3 Resource Usage

Table 1.2 shows the resources used by Radio driver and Timer.

Table 1.2 Radio driver and Timer resource usage

Resources	Function	RL78/G23-64p FPB* (rl78g23-64pfpb_sx126x)	RL78/G23-128p FPB* (rl78g23-128pfpb_sx126x)	RL78G/14 FPB* (rl78g14fpb_sx126x)
ROM(KiB)	----	29.8	29.8	31.5
RAM(KiB)	----	0.8	0.8	0.8
Stack(KiB)	----	0.6	0.6	0.6
Timer	----	32bit Interval Timer TAU02	32bit Interval Timer TAU02	RTC Timer RJ 12bit Interval Timer
SX126x	CLK MISO MOSI ANTSW NSS DIO1 BUSY XTAL_SEL DEVICE_SEL FREQ_SEL NRESET	SCK11(P30) SI11(P50) SO11(P51) OUT(P73) OUT(P76) INNTP11(P77) IN(P42) IN(P25) IN(P24) IN(P23/RFU) OUT(P22)	SCK11(P95) SI11(P96) SO11(P97) OUT(P42) OUT(P46) INNTP11(P77) IN(P106) IN(P147) IN(P117) IN(P116/RFU) OUT(P115)	SCK31(P54) SI31(P53) SO31(P52) OUT(P03) OUT(P02) INTP11(P77) IN(P75) IN(P23) IN(P24) IN(P25/RFU) OUT(P26)
UART	Tx Rx	TxD0(P12) RxD0(P11)	TxD0(P12) RxD0(P11)	TxD0(P51) RxD0(P50)
I^2C(optional) for sensor	SCL SDA	SCLA1(P62/GROVE) SDAA1(P63/GROVE)	SCLA1(P62/PMOD2) SDAA1(P63/PMOD2)	SCLA0(P14/PMOD2) SDAA0(P15/PMOD2)

Note: FPB stands for Fast Prototyping Board.

Note: ROM/RAM size includes .RLIB, .SLIB and the lower layer's code required by Radio driver.

1.4 Related Documentation

No.	Title	Author	Language
R11AN0393JJ0310	Radio Driver for Japan Radio Regulations	Renesas Electronics	Japanese

2. Radio Interface (LoRa / (G)FSK)

This section describes the Radio driver definition. The Radio driver can be used by the APIs described in this section.

At least following APIs should be executed to send or receive configuration.

Type / Functions	Description
RadioEvents_t	Events handler function table to handle the event from radio.
Radio.Init()	Initialize the radio driver with RadioEvents_t event handler.
Radio.SetTxConfig()	Set Tx parameters to send data.
Radio.SetRxConfig()	Set Rx parameters to receive data.
Radio.SetChannel()	Set channel frequency to send or receive.

To send or to receive data, the following APIs are used.

Type / Functions	Description
Radio.Send()	Set the radio to transmission mode.
Radio.Rx()	Set the radio to reception mode.
Radio.IrqProcess()	Process radio IRQ and call event handler (e.g., RxDone(), TxDone()).

2.1 Defines

2.1.1 Board configuration

RP_CPU_CLK	Set the operating clock frequency of the MCU [MHz]. Choose between 32 [MHz] and 8[MHz]. To change the frequency for RL78, it is necessary to change the user option byte as below (example). RP_CPU_CLK=32 : option byte: 6e7ae8(RL78/G23)/6effe8(RL78/G14) RP_CPU_CLK=8 : option byte: 6e7aaa(RL78/G23)/6effaa(RL78/G14)
RP_USE_DCDC_FOR_RADIO	If this macro is defined radio driver works as circuit has also LDO and DC-DC converter on the board. If it is not defined, radio driver works as circuit has LDO only. (default: Defined)
RP_USE_TCXO_FOR_RADIO	If this macro is defined, radio circuit uses external TCXO. If it is not defined, radio driver uses external crystal oscillator. (default: NOT defined)
RP_TCXO_CTRL_VOLTAGE	TCXO applied voltage . valid value is defined at RadioTcxoCtrlVoltage_t. (default: TCXO_CTRL_1_8V)
RP_TCXO_STAB_TIME	TCXO oscillation stabilization time defined below: RP_TCXO_STAB_TIME * 15.625us. (default: 640)
DEFAULT_POWER_SELECT	RADIO_LOPOWER_SEL (1): It means Radio signal power upper limit is +15dBm. (applied to SX1261) RADIO_HIPOWER_SEL (2): It means Radio signal power upper limit is +22dBm. (applied to SX1262) (default: RADIO_LOPOWER_SEL)
RP_USE_RF_SWITCH	RF switch configuration. If uses a board with RF switch, this macro should be defined. (default: NOT defined)
RP_CONTROL_ANTSW_BY_MCU	Define if MCU controls power supply of RF switch. (default: defined)
RP_DETECT_BOARD_CONFIG	If this macro is defined, radio driver detects board configuration regarding SX1261/SX1262 and XTAL/TCXO. In this case, board settings specified by DEFAULT_POWER_SELECT and RP_USE_TCXO_FOR_RADIO are ignored.(default: defined) NOTE: This macro works with SEMTECH LoRa Shield ONLY.

2.1.2 Misc

RADIO_FAR	Specifying memory allocation area. This indication is that address range 0x000000 to 0x0FFFFF for all RAM data, ROM data and functions. (default: __far)

2.2 Constants

2.2.1 RADIO_WAKEUP_TIME

RADIO_WAKEUP_TIME	Radio complete Wake-up Time [ms] with margin for temperature compensation

2.3 Enumeration

2.3.1 RadioModems_t

Radio driver supported modems.

MODEM_FSK	0	Uses radio with (G)FSK mode (default)
MODEM_LORA	1	Uses radio with LoRa mode

2.3.2 RadioState_t

Radio driver status.

RF_IDLE	0	Radio driver is idle state. (default)
RF_RX_RUNNING	1	Radio driver is in reception state.
RF_TX_RUNNING	2	Radio driver is in transmission state.
RF_CAD	3	Radio driver is channel activity detection state.
RF_COLD_SLLEP	4	Radio driver is cold sleep state.
RF_WARM_SLLEP	5	Radio driver is warm sleep state.

2.3.3 RadioTcxoCtrlVoltage_t

DIO3 TCXO voltage.

TCXO_CTRL_1_6V	0x00	1.6V
TCXO_CTRL_1_7V	0x01	1.7V
TCXO_CTRL_1_8V	0x02	1.8V (default)
TCXO_CTRL_2_2V	0x03	2.2V
TCXO_CTRL_2_4V	0x04	2.4V
TCXO_CTRL_2_7V	0x05	2.7V
TCXO_CTRL_3_0V	0x06	3.0V
TCXO_CTRL_3_3V	0x07	3.3V

2.3.4 PIB_t

PHY information list. PIB is initialized in Init() and should be set in RF_IDLE.

PIB_RSSI_OFFSET	0	Offset of RSSI [dB] (int8_t, Default 0)
PIB_CCA_BANDWIDTH	1	Bandwidth [Hz] for IsChannelFree() / Ed() function [Setting value] (uint32_t) 0 (default): Bandwidth of SetRxConfig is used for IsChannelFree() / Ed() function. Others: Specified bandwidth is used for IsChannelFree() / Ed() function. valid values below: 4800, 5800, 7300, 9700, 11700, 14600, 19500, 23400, 29300, 39000, 46900, 58600, 78200, 93800, 117300, 156200, 187200, 234300, 312000, 373600, 467000
PIB_CALL_RX_DONE_IN_PAYLOAD_CRC_ERROR	2	Change RxDone called or not when payload CRC error is occurred. [Setting value] (bool) false (default): RxDone is not called when payload CRC error is occurred. true: RxDone is called when payload CRC error is occurred.
PIB_GAIN_BOOSTED	3	Change Rx Gain Configuration. [Setting value] (bool) true: Uses Rx boosted gain false (default): Uses Rx power saving gain
PIB_XTAL_XTA_TRIM	6	XTAL trimming cap register value for XTA pin. Default: 0x13, uint8_t, range: 0x00(11.3pF) – 0x2F(33.4pF)
PIB_XTAL_XTB_TRIM	7	XTAL trimming cap register value for XTB pin. Default: 0x13, uint8_t, range: 0x00(11.3pF) – 0x2F(33.4pF)
PIB_RADIO_CFG_CHECK_ENABLE	8	Enable radio configurations validation for transmission and reception. When true is set, radio configurations are checked based on preconfigured validation items every time Radio.Send(), Radio.Rx(), Radio.SetTxContinuousWave, Radio.SetTxInfinitePreamble or Radio.CheckRfFrequency() is called. For more details, please refer to the “Radio Driver for Japan Radio Regulations” (R11AN0393JJ*). [Setting value] (bool) true: Enable validation. false (default): Disable validation.

2.3.5 RadioResult_t

Return values of API.

RADIO_SUCCESS	0	API is success.
RADIO_ARG_IS_NULL	1	Argument has NULL pointer.
RADIO_ARG_IS_INVALID	2	Argument is not correct.
RADIO_FAIL	3	Radio operation is failed.
RADIO_CHECK_FAIL _RX_CFG	100	Reception cannot be started due to invalid reception configurations.
RADIO_CHECK_FAIL _TX_CFG	101	Transmission cannot be started due to invalid transmission configurations.
RADIO_CHECK_FAIL _TX_DUTY_CYCLE	102	Transmission cannot be started due to restriction of minimum transmission interval or duty cycle.
RADIO_CHECK_FAIL_ _TX_CHANNEL_BUSY	103	Carrier sense detected radio signal in the target channel and transmission cannot be started.

2.3.6 Error Flags of Rx

Radio error occurred on receiving.

RADIO_ERROR_NONE	0x0000	No error on receiving.
RADIO_PAYLOAD_CRC_ ERROR	0x0040	Receive CRC error frame.

2.4 Type definition

None.

2.5 Structure

2.5.1 RadioEvents_t

void (RADIO_FAR *)(void)	TxDone	Tx done callback Packet transmission succeeded normally.
void (RADIO_FAR *)(void)	TxTimeout	Tx timeout callback Tx is aborted or fails.
void (RADIO_FAR *)(uint8_t *, uint16_t *, int16_t, int8_t)	RxDone	Rx done callback Packet reception succeeded normally.
void (RADIO_FAR *)(void)	RxTimeout	Rx timeout callback Not detect packet, or not complete reception in the time.
void (RADIO_FAR *)(void)	RxError	Rx error callback Wrong CRC received.
void (RADIO_FAR *)(uint8_t)	FhssChangeChannel	N/A
void (RADIO_FAR *)(bool)	CadDone	N/A

2.5.2 RadioTxFailStatus_t

This structure contains packet transmission failure flags for each predefined radio band. Users can read the flags in this structure to analyze the cause for the latest packet transmission failure.

uint8_t	numBands	Number of radio bands defined.
RadioTxFailStatus_t *	pFlag	Pointer to the head of the failure flag array. Failure flags of a band with the band ID of bandId is stored in *(pFlag + bandId). Note that bandId should not exceed numBands. (0 ≤ bandId < numBands).

2.6 Radio APIs (Radio_s Radio)

This section contains members of Radio_s. the radio interface could be used through the instance “Radio” of Radio_s structure. e.g., call Radio.Send() member function to execute Send().

Note: all API cannot be called in MCU interrupt handler.

Table 2.1 Radio interface APIs shows function in this structure below:

Table 2.1 Radio interface APIs

Function	Description
Init()	Initializes the radio driver.
SetRxConfig()	Sets the reception parameters.
SetTxConfig()	Sets the transmission parameters.
SetChannel()	Sets the channel frequency.
SetModem()	Configures the radio modem.
SetMaxPayloadLength()	Sets the maximum payload length.
SetPublicNetwork()	Set the network configuration to public or private.
Rx()	Sets the radio in reception mode.
Send()	Sets the radio in transmission mode.
Sleep() / SleepWarm()	Sets the radio in warm sleep mode (RC64K off).
Standby()	Sets the radio standby mode.
SetTxContinuousWave()	Sets the radio in continuous wave transmission mode.
SetTxInfinitePreamble()	Sets the radio in modulated continuous preamble transmission mode.
Rssi()	Reads the instantaneous RSSI value.
TimeOnAir()	Computes the packet occupied time on air.
CheckRfFrequency()	Checks the channel frequency is supported by the hardware.
GetStatus()	Gets current radio status.
IsChannelFree()	Checks whether the channel is free.
Random()	Generates a 32bit random value.
IrqProcess()	Processes radio IRQ events.
Ed()	Gets energy detection value.
GetPib()	Gets PIB. (PHY information block)
SetPib()	Sets PIB. (PHY information block)
WakeUp()	Wakes up the radio and recover board configuration.
SleepCold()	Sets the radio in cold sleep mode (RC64K off).
CalibrateImage()	Executes image calibration.
GetErrorFlag()	Gets the radio Rx error flags.
Write()	Writes to the radio register at the specified address.
Read()	Reads the radio register at the specified address.
GetTimeToNextTx()	Acquires estimated time for next packet transmission.
GetWakeupTime()	Gets the time required for the board plus radio to get out of sleep.

2.6.1 Init

RadioResult_t (RADIO_FAR *Init)(RadioEvents_t *event)		
Initializes the radio and register event callback.		
Parameters:		
	*event	Structure containing the Radio driver callback functions
	(RADIO_FAR *TxDone)(void)	[IN] Tx Done callback function address or NULL (Discard event). See 2.7.1 TxDone
	(RADIO_FAR *TxTimeout)(void)	[IN] Tx Timeout callback function address or NULL (Discard event). See 2.7.2 TxTimeout
	(RADIO_FAR * RxDone)(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)	[IN] Rx Done callback function address or NULL (Discard event). See 2.7.3 RxDone
	(RADIO_FAR *RxTimeout)(void)	[IN] Rx Timeout callback function address or NULL (Discard event). See 2.7.4 RxTimeout
	(RADIO_FAR *RxError)(void)	[IN] Rx Error callback function address or NULL (Discard event). See 2.7.5 RxError
	(RADIO_FAR *FhssChangeChannel)(uint8_t currentChannel)	[IN] N/A set to NULL
	(RADIO_FAR *CadDone) (bool channelActivityDetected)	[IN] N/A set to NULL

Return:	
RADIO_SUCCESS	Radio initialization success.
RADIO_FAIL	Radio initialization fail.
RADIO_ARG_IS_NULL	*event is null

2.6.2 SetRxConfig

RadioResult_t (RADIO_FAR *SetRxConfig)(RadioModems_t modem, uint32_t bandwidth, uint32_t datarate, uint8_t coderate, uint32_t bandwidthAfc, uint16_t preambleLen, uint16_t symbTimeout, bool fixLen, uint8_t payloadLen, bool crcOn, bool FreqHopOn, uint8_t HopPeriod, bool iqInverted, bool rxContinuous)	
This function sets the reception parameters.	
Parameters:	
modem	[IN] Radio modem to be used 0: (G)FSK 1: LoRa (see 2.3.1 RadioModems_t)
bandwidth	[IN] Sets the bandwidth (G)FSK: >= 4800 and <= 467000 Hz LoRa: 0: 125 kHz, 1: 250 kHz, 2: 500 kHz, 3: 62 kHz, 4: 41 kHz, 5: 31 kHz, 6: 20 kHz, 7: 15 kHz, 8: 10 kHz, 9: 7 kHz
datarate	[IN] Sets the datarate (G)FSK: 600...300000 bits/s LoRa: 5: SF5, …, 12: SF12
coderate	[IN] Sets the coding rate (LoRa only) (G)FSK: N/A (set to 0) LoRa: 1: 4/5, 2: 4/6, 3: 4/7, 4: 4/8
bandwidthAfc	[IN] N/A (set to 0)
preambleLen	[IN] Sets the Preamble length (G)FSK: Number of bytes (1...8191), LoRa: Length in symbols (1...0xffff) Note: In SF5 or 6, preambleLen is modified 12 if preambleLen less than 12.
symbTimeout	[IN] Set timeout for detection of frame reception in case of Rx single mode (i.e. rxContinuous is set to false). (G)FSK: Timeout in number of bytes (0…0xffff). maximum 262 seconds. LoRa: The “SymbNum” parameter (0…0xff). For more detail, please refer to the SetLoRaSymbNumTimeout in SX126x Datasheet. Set 0 in case of RX continuous mode (i.e. rxContinuous is set to true).
fixLen	[IN] Fixed length packets false: Variable/Explicit header true: Fixed/Implicit header

payloadLen	[IN] Sets payload length when fixed length is used (0...255)
crcOn	[IN] Enables CRC false: CRC OFF true: CRC ON Note: Only enable at fixLen = false
FreqHopOn	[IN] N/A (set to false)
HopPeriod	[IN] N/A (set to 0)
iqInverted	[IN] Inverts IQ signals (LoRa only) (G)FSK: N/A (set to false) LoRa: false: not inverted true: inverted
rxContinuous	[IN] Sets the reception in RX continuous mode false: RX single mode true: RX continuous mode
Return:	
RADIO_SUCCESS	Radio Config success.

2.6.3 SetTxConfig

RadioResult_t (RADIO_FAR* SetTxConfig)(RadioModems_t modem, int8_t power, uint32_t fdev, uint32_t bandwidth, uint32_t datarate, uint8_t coderate, uint16_t preambleLen, bool fixLen, bool crcOn, bool FreqHopOn, uint8_t HopPeriod, bool iqInverted, uint32_t timeout)	
This function sets the transmission parameters.	
Parameters:	
modem	[IN] Radio modem to be used [0: (G)FSK, 1: LoRa] (see 2.3.1 RadioModems_t)
power	[IN] Sets the output power [dBm] Low Power (SX1261): -17…15[dBm] High Power (SX1262): -9…22[dBm]
fdev	[IN] Sets the frequency deviation ((G)FSK only) (G)FSK: 0x000000…0xFFFFFF [Hz] LoRa: set to 0
bandwidth	[IN] Sets the bandwidth (G)FSK: set to 0 LoRa: 0: 125 kHz, 1: 250 kHz, 2: 500 kHz, 3: 62 kHz, 4: 41 kHz, 5: 31 kHz, 6: 20 kHz, 7: 15 kHz, 8: 10 kHz, 9: 7 kHz
datarate	[IN] Sets the data rate (G)FSK: 600...300000 bits/s LoRa: 5: SF5, … ,12: SF12
coderate	[IN] Sets the coding rate (LoRa only) (G)FSK: N/A (set to 0) LoRa: 1: 4/5, 2: 4/6, 3: 4/7, 4: 4/8
preambleLen	[IN] Sets the preamble length (G)FSK: Number of bytes (1...8191), LoRa: Length in symbols (1...0xffff) Note: Set SF5 or 6, preambleLen is modified 12 if preambleLen less than 12.
fixLen	false: Variable/Explicit header true: Fixed/Implicit header
crcOn	[IN] Enables CRC false: CRC OFF true: CRC ON Note: Only enable at fixLen=false
FreqHopOn	[IN] N/A (set to false)
HopPeriod	[IN] N/A (set to 0)

iqInverted	[IN] Inverts IQ signals (LoRa only) (G)FSK: N/A (set to false) LoRa: false: not inverted true: inverted
timeout	[IN] Transmission timeout [ms] (0 to 4294967295)
Return:	
RADIO_SUCCESS	Radio Config success.

2.6.4 SetChannel

void (RADIO_FAR *SetChannel)(uint32_t freq)	
This function sets the channel frequency to radio	
Parameters:	
freq	[IN] Channel frequency [Hz] (426000000...928000000)
Return:	
None	

2.6.5 SetModem

void (RADIO_FAR *SetModem)(RadioModems_t modem)	
Configures the radio modem.	
Parameters:	
modem	[IN] Modem to be used [0: (G)FSK, 1: LoRa] (see 2.3.1 RadioModems_t)
Return:	
None	

2.6.6 SetMaxPayloadLength

void (RADIO_FAR *SetMaxPayloadLength)(RadioModems_t modem, uint8_t max)	
This function sets the maximum payload length. This is only needed in fixed length payload (Implicit header). When user calls this API, it should be called after SetRxConfig().	
Parameters:	
Modem	[IN] Radio modem to be used [0: (G)FSK, 1: LoRa] (see 2.3.1 RadioModems_t)
Max	[IN] Maximum payload length in bytes (0...255)
Return:	
None	

2.6.7 SetPublicNetwork

void (RADIO_FAR *SetPublicNetwork)(bool enable)	
This function sets public or private network and changes radio modem to LoRa.	
Parameters:	
enable	[IN] true: set public network sync word for LoRaWAN® false: set private network sync word
Return:	
None	

2.6.8 Rx

RadioResult_t (RADIO_FAR *Rx)(uint32_t timeout)	
This function sets the radio to reception mode. (see 2.7.3 RxDone, 2.7.4 RxTimeout and 2.7.5 RxError)	
Parameters:	
timeout	[IN] Reception timeout (0 to 4294967295) [ms] RxTimeout() callback is called as follows; [Rx Single mode] RxTimeout() is called when argument timeout or symbTimeout (*1) is expired. [Rx Continuous mode (*2)] RxTimeout() is called when argument timeout is expired. (*1) 2.6.2 SetRxConfig symbTimeout (*2) 2.6.2 SetRxConfig rxContinuous (true)
Return:	
RADIO_SUCCESS	Success
RADIO_CHECK_FAIL _RX_CFG	Reception cannot be started due to unsupported modulation configurations. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.

2.6.9 Send

RadioResult_t (RADIO_FAR *Send)(uint8_t RADIO_FAR *buffer, uint8_t size)	
This function sends buffer data. Prepares the packet to be sent and sets the radio to transmission. (see 2.7.1 TxDone and 2.7.2 TxTimeout)	
Parameters:	
*buffer	[IN] Buffer pointer
size	[IN] Buffer size [bytes] (0...255)
Return:	
RADIO_SUCCESS	Success
RADIO_ARG_IS_NULL	*buffer is null
RADIO_CHECK_FAIL_ TX_CFG	Transmission cannot be started due to unsupported modulation configurations. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.
RADIO_CHECK_FAIL_ TX_DUTY_CYCLE	Transmission cannot be started due to restriction of transmission pause or duty cycle. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.
RADIO_CHECK_FAIL_ TX_CHANNEL_BUSY	Radio channel is busy, and transmission cannot be started. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.

2.6.10 Sleep / SleepWarm

void (RADIO_FAR *Sleep / *SleepWarm)(void)	
This function sets the radio to the warm sleep mode (RC64K off). SleepWarm() is alias of the Sleep(). WakeUp() and almost all APIs accessing the radio will wake the radio from the warm sleep state.	
Parameters:	
None	
Return:	
None	

2.6.11 Standby

void (RADIO_FAR *Standby)(void)	
This function sets the radio to the standby mode. This function can be called any time after Init().	
Parameters:	
None	
Return:	
None	

2.6.12 SetTxContinuousWave

RadioResult_t (RADIO_FAR *SetTxContinuousWave)(uint32_t freq, int8_t power, uint16_t time)	
This function sets the radio in unmodulated continuous wave transmission mode. When time expired, TxTimeout() is called.	
Parameters:	
freq	[IN] Channel frequency [Hz] (426000000...928000000)
power	[IN] Sets the transmission power [dBm] (Low Power: -17…15, High Power: -9…22)
time	[IN] Transmission mode timeout [s] (0…65535)
Return:	
RADIO_SUCCESS	Success.
RADIO_CHECK_FAIL _TX_CFG	Transmission cannot be started due to restriction on continuous transmission. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.

2.6.13 SetTxInfinitePreamble

RadioResult_t (RADIO_FAR *SetTxInfinitePreamble)(uint32_t freq, int8_t power, uint16_t time)	
This function sets the radio in LoRa modulated continuous preamble transmission mode. When time expired, TxTimeout() is called.	
Parameters:	
freq	[IN] Channel frequency [Hz] (426000000...928000000)
power	[IN] Sets the transmission power [dBm] (Low Power: -17 to 15, High Power: -9 to 22)
time	[IN] Transmission mode timeout [s] (0…65535)
Return:	
RADIO_SUCCESS	Success.
RADIO_CHECK_FAIL _TX_CFG	Transmission cannot be started due to restriction on continuous transmission. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is “true”.

2.6.14 Rssi

int16_t (RADIO_FAR *Rssi)(RadioModems_t modem)	
This function reads the instantaneous RSSI value with current modem.	
Parameters:	
modem	Don't care
Return:	
(0 to -127) + PIB_RSSI_OFFSET's value	instantaneous RSSI value in [dBm]

2.6.15 TimeOnAir

uint32_t (RADIO_FAR *TimeOnAir)(RadioModems_t modem, uint8_t pktLen)	
This function computes the packet time on air in ms for the given payload. Modem parameters must be preset. (e.g., SetRxConfig() or SetTxConfig())	
Parameters:	
modem	[IN] Radio mode to be used [0: (G)FSK, 1: LoRa] (see 2.3.1 RadioModems_t)
pktLen	[IN] Packet payload length
Return:	
time	Time [ms] for the given packet payload length

2.6.16 CheckRfFrequency

bool (*CheckRfFrequency)(uint32_t frequency)	
This function returns whether the driver supports the specified frequency.	
Parameters:	
frequency	[IN] channel frequency to be checked
Return:	
true	supported
false	unsupported. This value can return only when PIB_RADIO_CFG_CHECK_ENABLE is "true".

2.6.17 GetStatus

RadioState_t (RADIO_FAR *GetStatus)(void)	
This function returns current radio status.	
Parameters:	
None	
Return:	
RF_IDLE	Radio driver is idle(default).
RF_RX_RUNNING	Radio driver is in reception state.
RF_TX_RUNNING	Radio driver is in transmission state.
RF_CAD	Radio driver is channel activity detection state.
RF_COLD_SLEEP	Radio driver is cold sleep state.
RF_WARM_SLEEP	Radio driver is warm sleep state.

2.6.18 IsChannelFree

bool (RADIO_FAR *IsChannelFree)(RadioModems_t modem, uint32_t freq, int16_t rssiThresh, uint32_t maxCarrierSenseTime)		
Checks whether the channel is free. Note: Modem and bandwidth for carrier sense must be set by the following API parameter. Modem: 2.3.1 RadioModems_t Bandwidth: 2.3.4 PIB_t PIB_CCA_BANDWIDTH or 2.6.2 SetRxConfig bandwidth		
Parameters:		
modem		Don't care.
freq		[IN] Channel frequency [Hz] (426000000...928000000)
rssiThresh		[IN] RSSI threshold [dBm] (-127…0)
maxCarrierSenseTime		[IN] Maximum time for RSSI measurement [ms] (1…10)
Return:		
true	1	Channel is free (signal is less than rssiThresh)
false	0	Channel is not free (signal is rssiThresh and over)

2.6.19 Random

uint32_t (RADIO_FAR *Random)(void)	
This function generates a 32 bits random value.	
Parameters:	
None	
Return:	
uint32_t	32 bits random value

2.6.20 IrqProcess

bool (RADIO_FAR *IrqProcess)(void)		
Process radio IRQ event. This function calls event handler registered in RadioEvents_t when corresponding IRQ event has occurred. Note: for processing involving radio IRQ events, it is necessary to set a timeout period that considers the period until this API is called.		
Parameters:		
None		
Return:		
true	1	next event remains in IRQ (IrqProcess() should be called again).
false	0	No event remained.

2.6.21 Ed

int16_t (RADIO_FAR *Ed)(uint32_t freq, int32_t edTime)	
Gets maximum RSSI value in [dBm]. Note: Modem and bandwidth for carrier sense must be set by the following API parameter. Modem: 2.3.1 RadioModems_t Bandwidth: 2.3.4 PIB_t PIB_CCA_BANDWIDTH or 2.6.2 SetRxConfig bandwidth	
Parameters:	
Freq	[IN] Channel frequency [Hz] (426000000...928000000)
edTime	[IN] ED scan time[ms] (1…50)
Return:	
(0 to -127) + PIB_RSSI_OFFSET value	maximum RSSI [dBm] value in edTime duration.

2.6.22 GetPib

bool (RADIO_FAR *GetPib)(PIB_t id, uint8_t RADIO_FAR * pOutVal)		
Gets PIB value.		
Parameters:		
id		[IN] PIB Id. (see 2.3.4 PIB_t)
pOutVal		[OUT] address of PIB value
Return:		
true	1	Get success
false	0	Get fail

2.6.23 SetPib

bool (RADIO_FAR *SetPib)(PIB_t id, uint8_t RADIO_FAR * pInVal)		
Sets PIB value.		
Parameters:		
id		[IN] PIB Id. (see 2.3.4 PIB_t)
pInVal		[IN] address of PIB value
Return:		
true	1	Set success
false	0	Set fail

2.6.24 WakeUp

void (RADIO_FAR *WakeUp)(void)
Wake up the radio from the cold or warm sleep mode. When radio is in the cold sleep mode, this function will recover the board related device settings (RF switch settings, TCXO settings, Regulator settings).
Parameters:
None
Return:
None

2.6.25 SleepCold

void (RADIO_FAR *SleepCold)(void)
Sets the radio in cold sleep mode and RC64K off. (Note: modem/register parameter lost) To activate radio, call 2.6.24 WakeUp (see 2.8.1 Modem parameter)
Parameters:
None
Return:
None

2.6.26 CalibrateImage

void (RADIO_FAR *CalibrateImage)(uint32_t freq)	
Execute Image Calibration (Calibrate the image signal rejection filter for operating frequency band).	
Parameters:	
freq	Channel frequency (426000000...928000000)
Return:	
None	

2.6.27 GetErrorFlag

uint16_t (RADIO_FAR * GetErrorFlag)(void)	
This function returns radio Rx error flags. It is available in the RxDone() callback only.	
Parameters:	
None	
Return:	
Rx Error Flags	see 0 Error Flags of Rx

2.6.28 Write

void (*Write)(uint16_t addr, uint8_t data)	
Writes to the radio register at the specified address.	
Parameters:	
addr	[IN] Register address
data	[IN] Write data
Return:	
None	

2.6.29 Read

uint8_t (*Read)(uint16_t addr)	
Reads the radio register at the specified address.	
Parameters:	
addr	[IN] Register address
Return:	
data	Read data

2.6.30 GetTimeToNextTx

int32_t (*GetTimeToNextTx)(void)	
Acquires estimated time for next packet transmission in a predefined radio band. This function can be called only when PIB_RADIO_CFG_CHECK_ENABLE is “enable”.	
Parameters:	
None	
Return:	
-1	Invalid radio band ID is specified, or calculation failed.
0	Radio driver can accept a packet transmission request in the specified radio band.
1 to 40000	Estimated minimum time in millisecond (ms) until next packet transmission becomes possible in the radio band specified.

2.6.31 GetWakeupTime

uint32_t (*GetWakeupTime)(void)	
Gets the time required for the board plus radio to get out of sleep.[ms].	
Parameters:	
None	
Return:	
time	When using a XTAL, RADIO_WAKEUP_TIME is returned. When using a TCXO, RADIO_WAKEUP_TIME + RP_TCXO_STAB_TIME * 15.625 / 1000 is returned.

2.7 Radio Driver Event Handler (RadioEvents_t)

To handle events from Radio driver, set handler functions in RadioEvents_t and call the initialize API, “Init()”. (see 2.7.1 RadioEvents_t)

If user changes the state of the radio driver (RadioState_t) by API, the handler corresponding to the previous state will not be called. (e.g., call “Standby()” in RF_RX_RUNNING state for cancelling receive)

2.7.1 TxDone

void (*TxDone) (void)
Tx Done callback. This function is called when packet transmission succeeded normally.
Parameters:
None
Return:
None

2.7.2 TxTimeout

void (*TxTimeout) (void)
Tx timeout timer callback. This function is called when Tx is aborted or fails.
Parameters:
None
Return:
None

2.7.3 RxDone

void (RADIO_FAR *RxDone)(uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)	
Rx Done callback. This function is called when packet reception succeeded normally.	
Parameters:	
*payload	[IN] Received buffer pointer. Note: Header and CRC data are not included in payload.
size	[IN] Received buffer size [bytes] (0...255)
rssi	[IN] RSSI value computed while receiving the frame [dBm]
snr	[IN] Raw SNR value given by the radio hardware. (G)FSK: N/A LoRa: SNR value in dB
Return:	
None	

2.7.4 RxTimeout

void (RADIO_FAR *RxTimeout)(void)
Rx Timeout callback. This function is called when not detect packet, or not complete reception in the time.
Parameters:
None
Return:
None

2.7.5 RxError

void (RADIO_FAR *RxTimeout)(void)
Rx Error callback. This function is called when an incorrect CRC is received.
Parameters:
None
Return:
None

2.8 Radio Driver API Constraint

2.8.1 Modem parameter

Following APIs will change the modem parameter. So, when these APIs are called, modem parameter should be set again. (*1)

API	Description
Init ()	Initialize all parameters.
SetTxContinuousWave ()	Operation frequency and Tx Power are changed to argument values.
SetTxInfinitePreamble ()	Operation frequency and Tx Power are changed to argument values.
IsChannelFree ()	Operation frequency is changed to argument value.
Ed ()	Operation frequency is changed to argument value.
Sleep(), SleepWarm()	Only configuration for the activated modem before going to sleep is retained in the device(SX1261/SX1262). Configuration of the other modems is lost and must be re-configured.
SleepCold ()	Initialize all parameters.

(*1) [API for modem parameters]
2.6.5 SetModem () , 2.6.7 SetPublicNetwork () ,2.6.3 SetTxConfig () , 2.6.2 SetRxConfig () ,
2.6.6 SetMaxPayloadLength () ,2.6.4 SetChannel ()

2.8.2 State of Available to Calling APIs

Following table is relation between state and API.

Yes: Available
No: Not available / invalid, do not call the API in that state.

Table 2.2 Relation between state and API

State / API	RF_IDLE	RF_RX_RUNNING	RF_TX_RUNNING	RF_COLD_SLEEP
Init()	Yes	Yes	Yes	Yes
SetRxConfig()	Yes	No	No	No
SetTxConfig()	Yes	No	No	No
SetChannel()	Yes	No	No	No
SetModem()	Yes	No	No	No
SetMaxPayloadLength()	Yes	No	No	No
SetPublicNetwork()	Yes	No	No	No
Rx()	Yes	No	Yes	No
Send()	Yes	Yes	No	No
Sleep()	Yes	No	No	No
Standby()	Yes	Yes	Yes	No
SetTxContinuousWave()	Yes	No	No	No
SetTxInfinitePreamble()	Yes	No	No	No
Rssi()	No	Yes	No	No
TimeOnAir()	Yes	Yes	Yes	Yes
CheckRfFrequency()	Yes	Yes	Yes	Yes
GetStatus()	Yes	Yes	Yes	Yes
IsChannelFree()	Yes	No	No	No
Random()	Yes	No	No	No
IrqProcess()	Yes	Yes	Yes	Yes
Ed()	Yes	No	No	No
GetPib()	Yes	No	No	Yes
SetPib()	Yes	No	No	Yes
WakeUp()	Yes	No	No	Yes
SleepWarm()	Yes	No	No	No
SleepCold()	Yes	No	No	No
CalibrateImage()	Yes	No	No	No
GetErrorFlag() (*1)	-	-	-	-
Write()	Yes	No	No	No
Read()	Yes	No	No	No
GetTimeToNextTx()	Yes	Yes	No	Yes
GetWakeupTime()	Yes	Yes	Yes	Yes

(*1) Only available within RxDone() callback.

3. Timer

Timer provides timer event and a system time value.

3.1 Type Definition

Timer uses the following types.

Table 3-1 Type Definition

Type	Description
TimerEvent_t	Timer object structure. To initialize this object, call TimerInit()
TimerTime_t	Timer time variable integer in millisecond

3.1.1 TimerEvent_t

TimerEvent_t is timer control block used in the timer module internally. The detailed description is omitted in this document.

3.1.2 TimerTime_t

Table 3-2 TimerTime_t

typedef uint64_t TimerTime_t;	Timer time variable integer in millisecond

3.2 Timer APIs

The following functions are available for Timer.

Function	Description
TimerInit()	Initializes the timer object
TimerSetValue()	Sets timer new timeout value.
TimerStart()	Starts timer.
TimerStop()	Stops timer
TimerReset()	Reset the timer object
TimerGetCurrentTime()	Gets the current time in millisecond
TimerGetElapsedTime()	Gets the time elapsed since fixed moment

3.2.1 TimerInit

void TimerInit(TimerEvent_t *obj, void (*callback)(void))	
This function initializes the timer object. To set timeout value, TimerSetValue() function must be called before starting the timer. this function initializes timer object with timeout value 0. To stop and initialize a timer object that is already running, call TimerStop() before calling this function.	
Parameters:	
obj	[IN] Structure containing the timer object parameters. See for 3.1.1 the timer object.
callback	[IN] Callback function called at the end of the timeout
Return:	
None	

3.2.2 TimerSetValue

void TimerSetValue(TimerEvent_t *obj, uint32_t value)	
This function set timer new timeout value. Do not call this function for a timer object that is running.	
Parameters:	
obj	[IN] Structure containing the timer object parameters
value	[IN] New timer timeout value[ms]. Valid argument range (decimal) is 0 to 3,888,000,000.
Return:	
None	

3.2.3 TimerStart

void TimerStart(TimerEvent_t *obj)	
This function starts timer and adds the timer object to the list of timer events.	
Parameters:	
obj	[IN] Structure containing the timer object parameters.
Return:	
None	

3.2.4 TimerStop

void TimerStop(TimerEvent_t *obj)	
This function stops timer and removes the timer object from the list of timer events.	
Parameters:	
obj	[IN] Structure containing the timer object parameters.
Return:	
None	

3.2.5 TimerReset

void TimerReset(TimerEvent_t *obj)	
This function resets the timer object. This function stops timer running and then restart it.	
Parameters:	
obj	[IN] Structure containing the timer object parameters.
Return:	
None	

3.2.6 TimerGetCurrentTime

TimerTime_t TimerGetCurrentTime(void)	
This function returns the current time in millisecond.	
Parameters:	
None	
Return:	
TimerTime_t	Current time [ms]. Note: 0xFFFFFFFFFFFFFFFF is returned as error if internal hardware calendar holds time earlier than the system startup time.

3.2.7 TimerGetElapsedTime

TimerTime_t TimerGetElapsedTime(TimerTime_t savedTime)	
This function returns the time elapsed since a fix moment in millisecond.	
Parameters:	
savedTime	Fix moment in Time
Return:	
TimerTime_t	Elapsed time [ms]. Note: 0xFFFFFFFFFFFFFFFF is returned if current time value is larger than savedTime,

Revision History

Rev.	Date	Description	
		Page	Summary
01.00	Jan 23, 2019	---	First official version
3.00	Mar. 3, 2021	1, 5 7	Support RL78/G23(rl78g23-64pfpb_sx126x) Delete default RP_CPU_CLK, Add option byte for RL78/G23
3.01	June 30, 2021	5	Update resource information.
3.10	Sep. 20, 2021	1, 5 7	Support RL78/G23(rl78g23-128pfpb_sx126x) Change option bytes settings for RL78/G23
3.11	Nov. 5, 2021	---	Revised version number, no functional changes.
3.12	Jan. 17, 2022	---	No API changes. Fixed a timer implementation bug.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Semtech, the Semtech logo, LoRa, LoRaWAN and LoRa Alliance are registered trademarks or service marks, or trademarks or service marks, of Semtech Corporation and/or its affiliates.

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.