RT568

API User Guide

1. Introduction

RT568 is a Bluetooth®5 Low Energy (BLE) transceiver IC. It includes an RF radio and modulator/demodulator. This document explains how to configure the chip and also describes register usage for BLE applications.

2. System Block Diagram

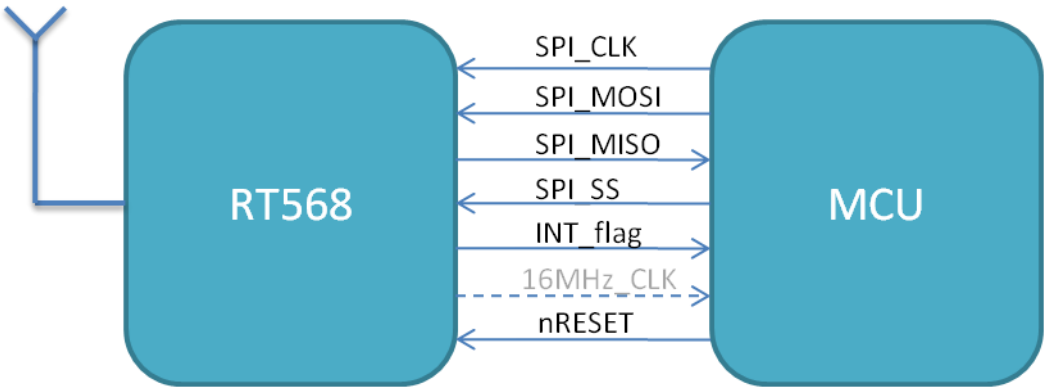


Figure 1. RT568 and MCU block diagram for BLE application

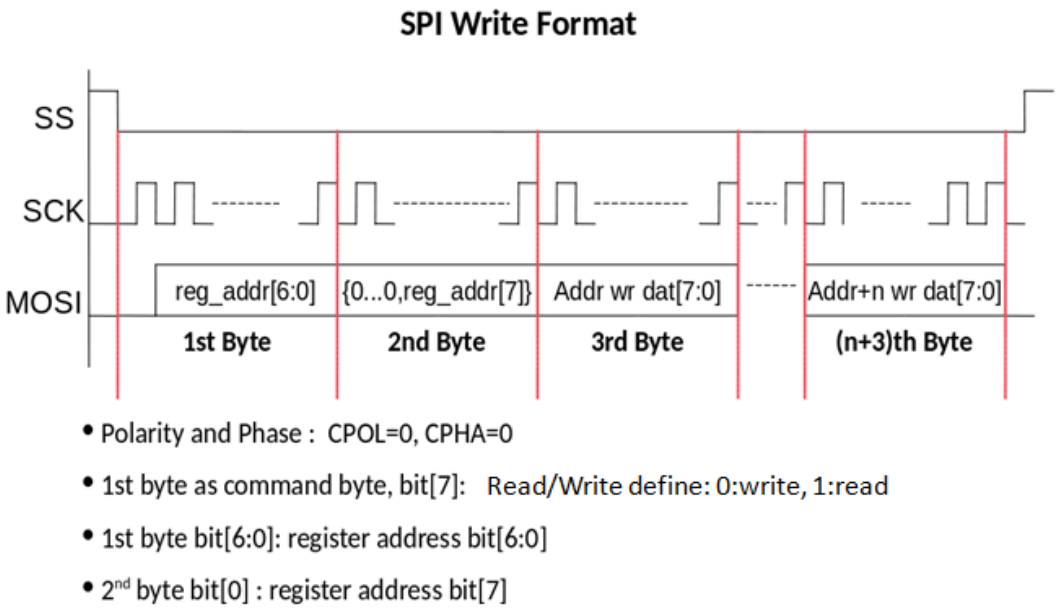
The MCU controls the RT568 through the SPI interface. The RT568 supports a maximum SPI clock rate up to 16MHz. The RT568 outputs an INT_flag which connects to the MCU GPIO pin. When RT568 enters specific states, the INT_flag pin outputs a pulse to interrupt the MCU. The MCU must read its interrupt status byte immediately to know the state of the RT568. Please see the **Interrupt and ISR** section for more details. RT568 provides a 16MHz clock output. The MCU can use this signal as a clock source or utilize some other source. The RT568 also provides an input for the external MCU to reset the RT568 outputting an active low pulse to RT568 nRESET pin.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

3. SPI Waveform

RT568 implements a proprietary SPI command protocol for register read/write and data buffer read/write.

(1)SPI Write Register



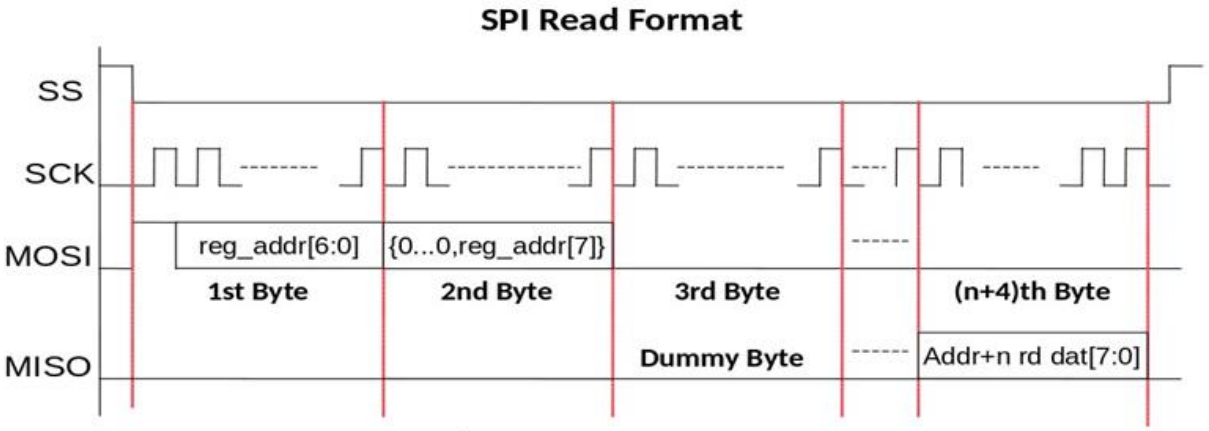
- Polarity and Phase : CPOL=0, CPHA=0
- 1st byte as command byte, bit[7]: Read/Write define: 0:write, 1:read
- 1st byte bit[6:0]: register address bit[6:0]
- 2nd byte bit[0] : register address bit[7]

NOTE: SS must be held continuously low during the entire write process.

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

(2) SPI Read Register



- Polarity and Phase : CPOL=0, CPHA=0
- 1st byte as command byte, bit[7]: Read/Write define: 0:write, 1:read
- 1st byte bit[6:0]: register address bit[6:0]
- 2nd byte bit[0] is register address bit[7]
- 3rd byte is Dummy byte, always output 0
- 4th byte output address register ....

NOTE: SS must be held continuously low during the entire read process.

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

4. Interrupt and ISR

Reg_61 is the interrupt enable register. Reg_62 is the interrupt status register.

Register Address	Name	Description
Reg_61	B0 ~ B7 : En_INT	B0 : 1, enable RX PDU Header Ready INT 0, mask RX PDU Header Ready INT B1 : 1, enable RX PDU 16 Bytes Ready INT 0, mask RX PDU 16 Bytes Ready INT B2 : 1, enable TX Packet End INT 0, mask TX Packet End INT B3 : 1, enable RX Timeout INT 0, mask RX Timeout INT B4 : 1, enable RX Packet End INT 0, mask RX Packet End INT B5 : 1, enable RX Access Code Sync INT 0, mask RX Access Code Sync INT B6 : 1, enable Wake Up INT defined in Reg_115~119 0, mask Wake Up INT B7 : Reserved
Reg_62	B0 ~ B7 : INT_Status	Write 1 to clear corresponding INT status. B0 : 1, Trigger INT when RX PDU Header Ready in RX FIFO B1 : 1, Trigger INT when RX PDU 16byte Data Ready B2 : 1, TX Packet End INT triggered. B3 : 1, RX Timeout INT triggered B4 : 1, RX Packet End INT triggered B5 : 1, RX Access Code Searched INT triggered B6 : 1, Trigger when (RTC Time == Wakeup Time). Periodic trigger every R159 ~ R156 period when B7 of R159 = 1 B7 : Reserved

*If the RT568 INT_flag pin output level is HIGH, the MCU GPIO interrupt is triggered. The MCU must clear INT_Status (Reg 62) for the INT_flag output to become LOW.

Register Address	Name	Description
Reg_156	B7 ~ B0 : Period_us[7:0]	Period Timer interrupt period: 0 ~ 0x7fffffff (μs)
Reg_157	B7 ~ B0 : Period_us[15:8]	
Reg_158	B7 ~ B0 : Period_us[23:16]	
Reg_159	B6 ~ B0 : Period_us[30:24]	
	B7 : Periodic or One-shot INT in EN_INT[6]	1 : Enable periodic interrupt in EN_INT[6], 0 : EN_INT[6] is a one-shot interrupt. Match counter is defined in R115 ~ R119

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

GPIO Interrupt Service Routine (ISR) example code:

```
void GPIO_IRQHandler(void)
{
    uint8_t interrupt_sataus;

    //clear MCU GPIO interrupt status
    //read RT568 INT status
    rafael_spi_read_single(SPI_MASTER_PORT, 62, &interrupt_sataus);
    //clear RT568 INT status
    rafael_spi_write_single(SPI_MASTER_PORT, 62, interrupt_sataus);

    //b5: Access code searched
    if(interrupt_sataus & RAFAEL_IRQ_SYNC)   {
        . . .
    }
    //b4: RX packet received
    if(interrupt_sataus & RAFAEL_IRQ_RECEIVED)   {
      . . .
    }
    //b2: TX packet send complete
    if(interrupt_sataus & RAFAEL_IRQ_TXEND)   {
        . . .
    }
    //b3: RX timeout, do not receive packet
    if(interrupt_sataus & RAFAEL_IRQ_RXSTOP)   {
      . . .
    }
    //b6: Wakeup and T/R triggered
    if(interrupt_sataus & RAFAEL_IRQ_WAKEUP)   {
     . . .
    }
    . . . . .
}   //end of ISR
```


The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

5. TX Data Port and RX Data Port

RT568 internal TX Buffer (320 bytes) and RX FIFO (320 bytes) for data transmission and reception.

Register Address	Name	Description
Reg_104	TX_START_ADDR[7:0]	TX_START_ADDR[8:0] : with TX_Enable, this tells the RF section the start address in the TX Buffer for transmit data
Reg_105	B0 : TX_START_ADDR[8]	
	B1 ~ B6 : reserved	
	B7: En payload underflow check	1 : Enable Tx Payload underflow check, used when transmitting packets
Reg_106	RX_FIFO_CNT[7:0]	RX_FIFO_CNT[8:0] : number of data bytes in the RX FIFO
Reg_107	B0 : RX_FIFO_CNT[8]	
	B1 ~ B5 : reserved B6 : Clear TX Payload Counter B7 : Clear RX FIFO	B6 : Write 1 to clear the TX Payload Counter, must write 0 after write 1 B7 : Write 1 to clear the RX FIFO, must write 0 for normal RX FIFO read/write operation
Reg_254	TX Buffer data Write Port	Write this register to put data into the TX Buffer
Reg_255	RX FIFO data Read Port	Read this register to get data from the RX FIFO

To write data into the TX_Buffer, the process is similar to writing registers via SPI. The only difference is reg_addr becomes 254 (Byte2[0],Byte1[6:0]) and TX_Buffer must indicate the write start address (Byte2[7:0],Byte3[0]); set TX data type with Byte3[7] = 1 write BLE payload; = 0 write BLE header.

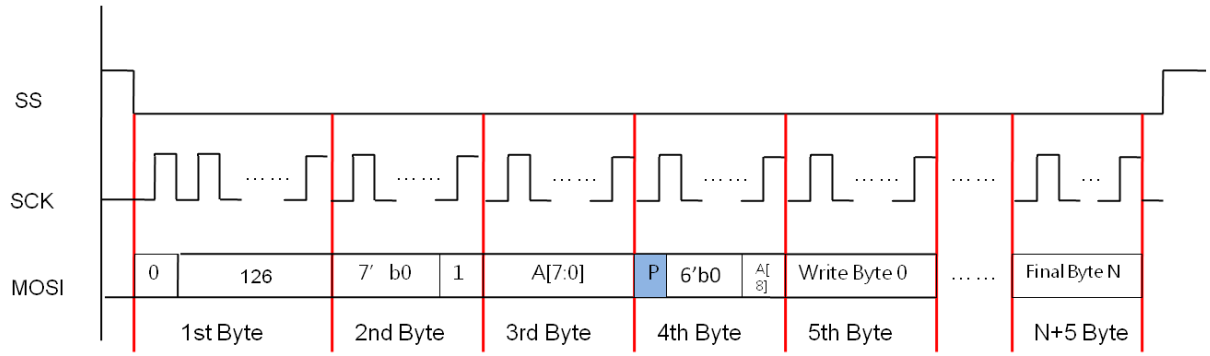
To read data from the RX_FIFO, the process is similar to reading registers via SPI. The only difference is reg_addr becomes 255 (Byte2[0],Byte1[6:0]) and Byte2, Byte3 are dummy bytes.

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

(1) SPI Write TX Buffer

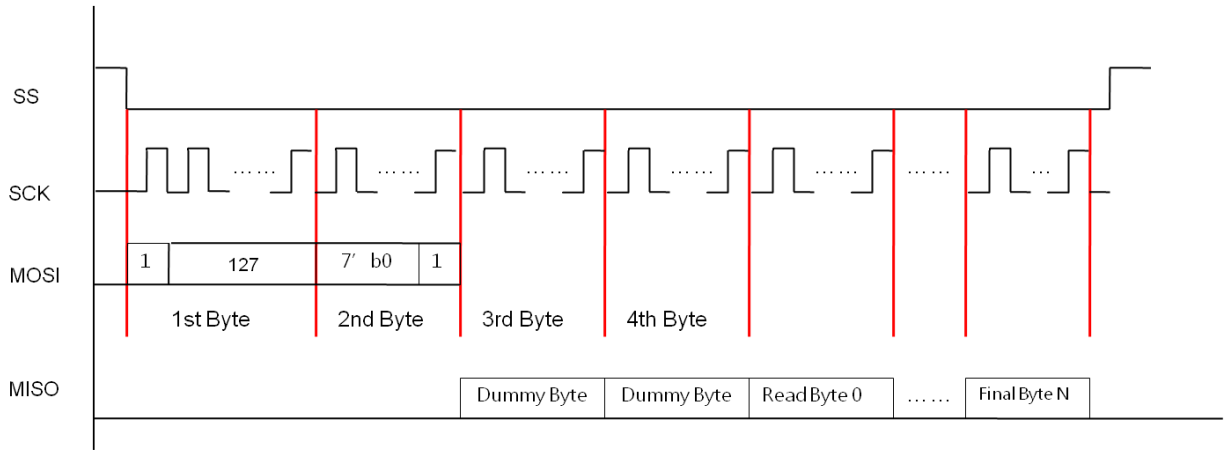
SPI Tx_Buffer Write Format



- 1st Byte bit7 is 0 means SPI write operation
- Register address is 254 (2nd byte[0], 1st byte[6:0]), means to write TX_buffer
- (4th byte[0], 3rd byte) is Tx buffer start address that user desires to write
- 4th byte[7] bit to indicate HW that SW is writing payload data or header data
  - 1: write payload 0: write header
- 5th byte and follow bytes are data

(2) SPI Read RX FIFO

SPI Rx FIFO Read Format



- 1st Byte bit7 is 1 means SPI read operation
- Register address is 255 (2nd byte[0], 1st byte[6:0]), means to read RX_FIFO
- 3rd byte, 4th byte are dummy bytes
- 5th byte and follow bytes are the data read from RX_FIFO

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

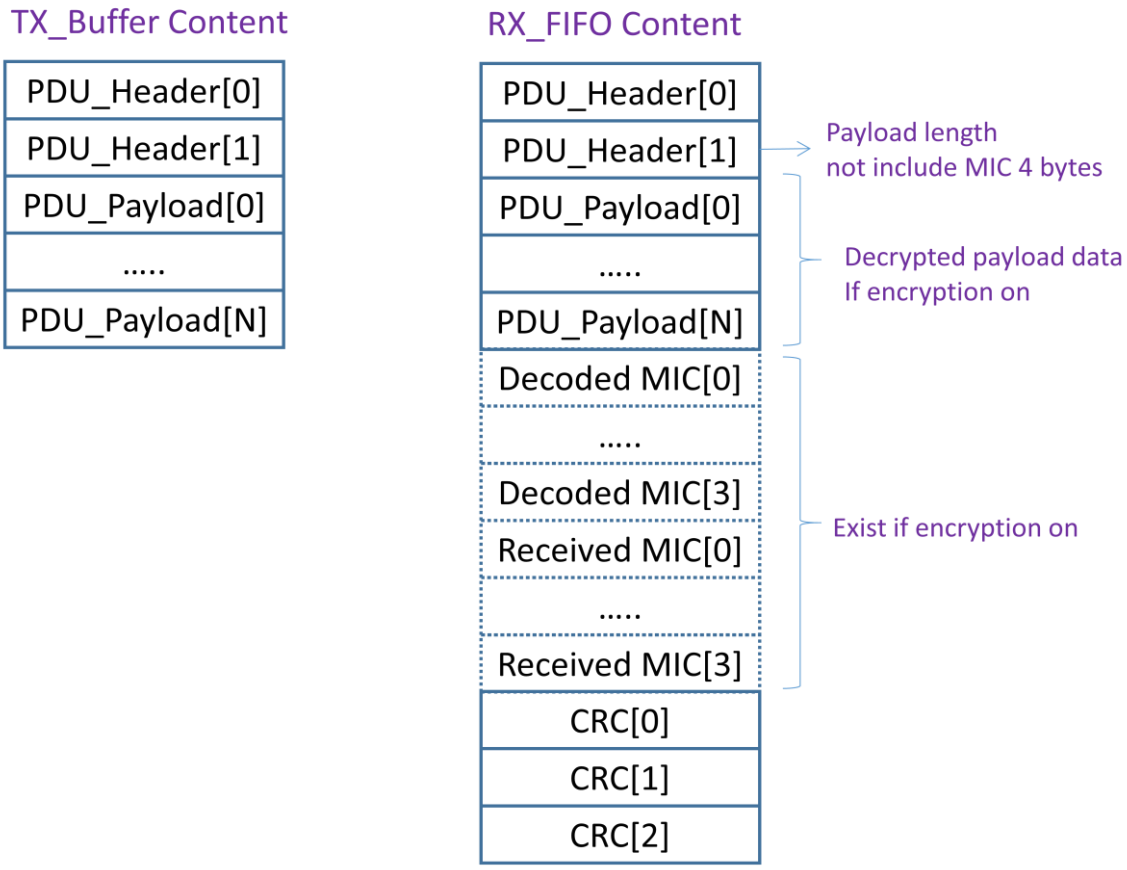


Figure 2. TX_Buffer and RX_FIFO Content

The RT568 supports AES-CCM encryption/decryption.

In Connection mode, if security transmission is enabled then software must configure AES_CCM_Nonce registers (Reg_126 ~ Reg_138), AES_CCM_Key registers (Reg_139 ~ Reg_154) and enable AES-CCM accelerator (Reg_155[2:0]=1) before the TX/RX task is enabled.

At the transmission side, write the raw data into the TX buffer. Once the TX_Task is enabled, the RT568 will automatically do AES-CCM encryption and send-out packets over the air. At the receiver side, decrypted data is read from the RX_FIFO. The data format is shown in Figure 2 (above).

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

6. PMU (Power Management Unit) Finite State Machine

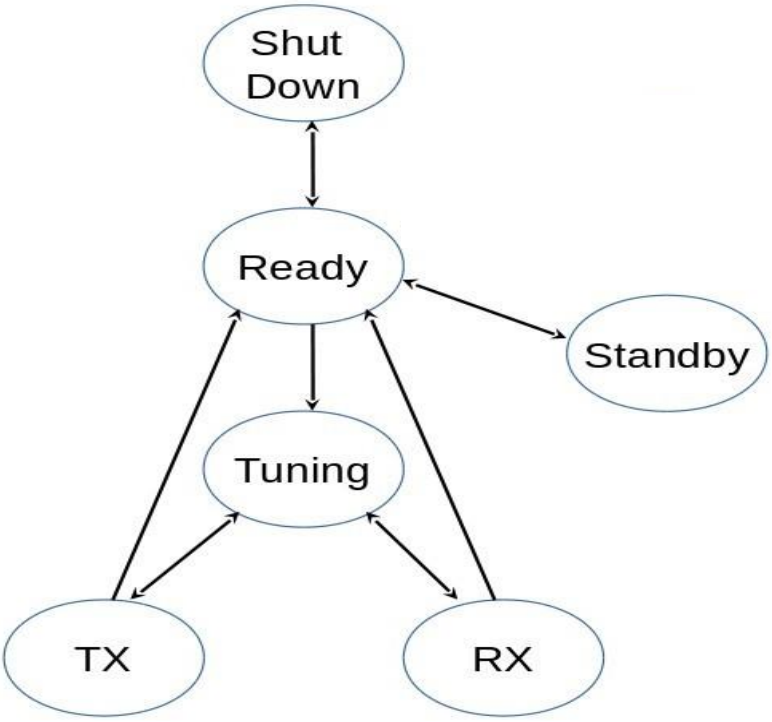


Figure 3. RT568 PMU Finite State Machine

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

7. SPI I/O Re-mapping

To easily integrate the RT568 and MCU in one package (SIP), the RT568 supports dynamic SPI pin allocation.

At the beginning of IC power-on, the software MUST run the following SPI I/O mapping flow:

(1) After power-on, the SPI I/O pins are inputs. The MCU must set all SPI pins to be GPIO pins.

(2) The SIP designer must know the RT568's SPI pin assignments connected to the MCU. The MCU will utilize these RT568 GPIO pins for SPI communication (SS, CLK, MOSI) to set the I/O registers (GPIO0_SEL ~ GPIO4_SEL). This step re-configures the RT568 I/O pins to map with the SPI's MCU pin definition.

Table1. RT568 SPI I/O Configuration

GPIO5_SEL	0					1				
GPIO0_SEL[2:0]	0	1	2	3	4	0	1	2	3	4
GPIO0	SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT	I[0]				
GPIO5	I[0]					SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT

GPIO5_SEL	0					1				
GPIO1_SEL[2:0]	0	1	2	3	4	0	1	2	3	4
GPIO1	SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT	I[1]				
GPIO6	I[1]					SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT

GPIO5_SEL	0					1				
GPIO2_SEL[2:0]	0	1	2	3	4	0	1	2	3	4
GPIO2	SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT	I[2]				
GPIO7	I[2]					SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT

GPIO5_SEL	0					1				
GPIO3_SEL[2:0]	0	1	2	3	4	0	1	2	3	4
GPIO3	SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT	I[3]				
GPIO8	I[3]					SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT

GPIO5_SEL	0					1				
GPIO4_SEL[2:0]	0	1	2	3	4	0	1	2	3	4
GPIO4	SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT	I[4]				
GPIO9	I[4]					SPI_CS	SPI_CLK	SPI_SDI	SPI_SDO	INT

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

Table 2. RT568 SPI I/O Registers

No.	Register Address	Name	Description	Power on Value
1	Reg_246	Bit[4:0] : reg_pull_en	GPIO5_SEL=0, Control GPIO0~4 pull en GPIO5_SEL=1, Control GPIO5~9 pull en	0x1f
2	Reg_247	Bit[4:0] : reg_up_down	GPIO5_SEL=0, Control GPIO0~4 pull up or pull down GPIO5_SEL=1, Control GPIO5~9 pull up or pull down	0x01
3	Reg_248	Bit[2:0] : GPIO0_SEL Bit[5:3] : GPIO1_SEL Bit[7:6] : GOIO2_SEL[1:0]	Refer to GPIO Mux EN_SDO_INT_Out, 1: enable SDO and INT function pin output 0: Keep SDO and INT function pin input	GPIO0_SEL = 0 GPIO1_SEL = 1 GPIO2_SEL = 2 GPIO3_SEL = 3 GPIO4_SEL = 4 EN_SDO_INT_Out = 0
4	Reg_249	Bit[0] : GPIO2_SEL[2] Bit[3:1] : GPIO3_SEL Bit[6:4] : GPIO4_SEL Bit[7] : EN_SDO_INT_Out		

For example, to configure with GPIO5_SEL=0:
GPIO0 = SPI_CS (SS)
GPIO1 = SPI_CLK (CLK)
GPIO2 = SPI_SDI (MOSI)
GPIO3 = SPI_SDO (MISO)
GPIO4 = INT (INT_flag)
then execute MCU write Reg_248 = 8'b01 001 000 , Reg_249 = 8'b0 100 011 0 by GPIO pins.

(3) set five MCU pins (SPI_SS, SPI_CLK, SPI_MOSI, SPI_MISO, INT_flag) as outputs and hold the levels HIGH for 10ms.

(4) configure the MCU SPI peripheral interface pins

(5) MCU SPI set Reg_249[7] = 1 to configure the RT568 MISO and INT pins as outputs.

(6) MCU SPI write Reg_40 = 0x90, Reg_53 = 0x80 to enable the RT568 integrated DC/DC converter.

(7) Wait 25ms for the RT568 to power-up.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

8. System Initialization

(a) ***Reset RT568*** by MCU GPIO control of the RT568 nReset pin:
High → Low (10us) → High then delay 30ms to wait for reset to complete.
(b) Do ***SPI I/O re-mapping*** (see Section 7).
(c) Call ***RAFAEL_Init( )*** to initialize the RT568.
This function performs the following operations:
1. writes initial values & default timing parameters into registers Reg_8 ~ Reg_127.
2. clears Reg_62 interrupt status bits
3. clears the RX_FIFO and TX_Payload_Counter
4. disables Auto TX/RX mode switching (TX/RX mode is controlled manually)
5. resets the PMU Finite State Machine
6. disables the integrated AES-CCM accelerator

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

9. MAC Register Description

Set the CRC generator's initial value at Reg_112 ~ Reg_114.
Set Access Address at Reg_122 ~ Reg_125 so the RT568 auto-appends a preamble according to the Access Address and Symbol_Rate (Reg_120[3]).

Register Address	Name	Description
Reg_109	CRC_CHECK_RESULT[7:0]	BLE received packet's computed CRC result (read only)
Reg_110	CRC_CHECK_RESULT[15:8]	
Reg_111	CRC_CHECK_RESULT[23:16]	
Reg_112	CRC_INIT[7:0]	BLE CRC generator's initial value Note: When En_Direct_Test (Reg_96[0])=1, R112 changes to payload length
Reg_113	CRC_INIT[15:8]	
Reg_114	CRC_INIT[23:16]	
Reg_122	ACCESS_CODE[7:0]	BLE packet Access Address
Reg_123	ACCESS_CODE[15:8]	
Reg_124	ACCESS_CODE[23:16]	
Reg_125	ACCESS_CODE[31:24]	

Reg_91 ~ Reg_97 define the timing parameters

Register Address	Name	Description
Reg_91	PLL_LOCK_Time	RF PLL first lock time from power-on (μs)
Reg_92	FAST_PLL_LOCK_Time	RF PLL second lock time from power-on (us)
Reg_93	NEXT_RX_PERIOD	Mac wait period from auto Tx → Rx
Reg_94	NEXT_TX_PERIOD	Mac wait period from auto Rx → Tx
Reg_95	Rx_Time_out_Period[7:0]	Rx time-out value for search packet access code fail (μs)
Reg_96	B7 ~ B4 : PA_On_Time B3 : En_Whiten B2 ~ B1 : Direct_Test_Pattern B0 : En_Direct_Test (B2 ~ B0 : Reserved)	PA_On_Time : The time between PA power-on and Baseband Modulator power-on. Direct_Test_Pattern : Select Tx PDU Pattern in BLE Direct Test Mode. 00: prbs 9, 01: 0x0F, 10: 0x55 En_Direct_Test : 1: BLE Direct Test Mode 0 : BLE normal mode En_Whiten : 0: Disable Whiten function in T/R 1: Enable Whiten function in T/R

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

Reg_97	Xtal_turn_on_time[7:0]	Xtal_turn-on_time defines the time which the crystal needs from turn-on to stable operation. The value is Xtal_turn_on_time * 31.25µs

NEXT_TX_PERIOD is used to fine tune the T_IFS timing for RX →TX.
NEXT_RX_PERIOD is used to fine tune the T_IFS timing for TX->RX.
Initial values of the timing parameters:

Reg_91 = 70
Reg_92 = 70
Reg_93 = 1
Reg_94 = 37
Reg_95 = 255
Reg_96[7:4] = 5
Reg_97 = 80

RT568 has an integrated free-running real-time clock (RTC). The RTC measures time in micro-seconds (µs).

Reg_98 ~ Reg_102[4:0] are the RTC counter registers.

Register Address	Name	Description
Reg_98	RTC_US[7:0]	Reg_98 ~ Reg_102[B4:B0] : Read to get the RTC values. Reg_102[B6:B5] : latch RTC value 0: disable R98 ~ R102 RTC latch. It is a free-running value. 1: latch RTC value when TX_End trigger. 2: latch RTC value when Access_code search ok. 3: latch RTC value when RX_End trigger. Reg_102[B7] : Write 1 to set the RTC with contents of Reg_98 ~ Reg_102. If write 0 to this bit, hardware will view the RTC value at Reg_98 ~ Reg_102 as invalid and keep the RTC counter going without changing the values of Reg_98 ~ Reg_102
Reg_99	RTC_US[15:8]	
Reg_100	RTC_US[23:16]	
Reg_101	RTC_US[31:24]	
Reg_102	B4 ~ 0: RTC_US[36:32] B6 ~ 5: RTC latch B7: Update RTC_US_latch	
Reg_103	Reserved	

NOTE: When reading the RTC, Reg_98 ~ Reg_102 are frozen when SPI starts reading Reg_98. These registers resume after Reg_102 is read.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

Reg_115 ~ Reg_121 are used to control the RT568's PMU state machine.

Register Address	Name	Description
Reg_115	Wake_Up_US[7:0]	Wake-up Time : Reg_115 ~ Reg_119[4:0] (Wake_Up_US[36:0]) is used to wake-up the RT568 when the content matches the RTC timer count. Write 1 into Reg_120[B7] during READY state to let the MAC go to STANDBY state. The PMU will wake-up & turn-on the XTAL at (Wake-up Time – Xtal_turn_on_time – 2048 µs). If write 0 to Reg_120[B7], RT568 will not go to **Standby State**, the PMU will stay in **Ready State** and go to **TX State** or **RX State** at Wake-up Time. B6 : TR_Trig_Mode, 1: Mac T/R task triggered when wake_up_time = RTC_time. 0: Mac T/R task triggered by write 1 to Man_En_TR (Reg_119,Bit 7) B7: Man_En_TR,: Write 1 to Manual enable T/R Task. The RT568 will automatically clear to 0.
Reg_116	Wake_Up_US[15:8]	
Reg_117	Wake_Up_US[23:16]	
Reg_118	Wake_Up_US[31:24]	
Reg_119	B4~B0: Wake_Up_S[36:32] B5: Reserved B6: TR_Trig_Mode B7: Man_En_TR	
Reg_120	B0: Reserved B1: Wake_Up_Mode B2: AUTO T/R B3 : SYMBOL_RATE B4 : PLL_CONFIG_DIRECT B5 : EN_RX_TIME_OUT B6 : RX_CRC_MIC_STORE B7:UPDATE_WAKE_UP_TIME	B1 : Wake_Up_Mode 0: Internal RTC Wake Up: CLK_32K remains always on 1: SPI External Wake Up: CLK_32K turned off, Wake Up by external SPI (write 1 to REG_8 Bit 0) B2 : AUTO T/R, Write 1 = MAC switches to TX or RX automatically when one packet is received or the transfer is finished, Write 0 = MAC switches back to ready state when one packet is received or the transfer is finished. B3 : SYMBOL_RATE, Write 1= BLE 2MHz mode; Write 0 = BLE 1MHz mode. B4: 1: RF LO frequency is configured by Reg32 ~ Reg34 0: RF LO Frequency configured by Reg121[5:0] channel index. B5 : EN_RX_TIME_OUT, Write 1 = enable RX access search time-out function. MAC goes to READY state when time-out is reached during access code searching. Write 0 = MAC continues searching for access code and goes to READY state when 1 is written into Reg_121[B7]. B6 : RX_CRC_MIC_STORE. 1: received CRC, MIC and decoded MIC are in RX_FIFO. 0: received CRC MIC and decoded CRC MIC are discarded. B7 : UPDATE_WAKE_UP_TIME, Write 1 to set the wake-up time using contents of Reg_115 ~ Reg_119. If write 0 to Reg_120[B7], RT568 will to go to **Standby State**, the PMU will remain in **Ready State** and go to **TX State** or **RX State** at the Wake-up Time.
Reg_121	B5 ~ B0 : Channel Index B6 : FIRST T/R B7 : RESET_MAC_STATE	B5 ~ B0 :Channel Index, BLE **Link Layer** channel index number B6 : FIRST T/R, Write 0 = MAC goes directly to TX state after wake-up or Manual Enable TR, Write 1 = MAC goes directly to RX state after wake-up or Manual Enable TR. B7 : RESET_MAC_STATE, Write 1 to end all TX & RX events and reset the MAC state to **READY STATE**, must write 0 to this bit for the PMU state machine to proceed normally.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

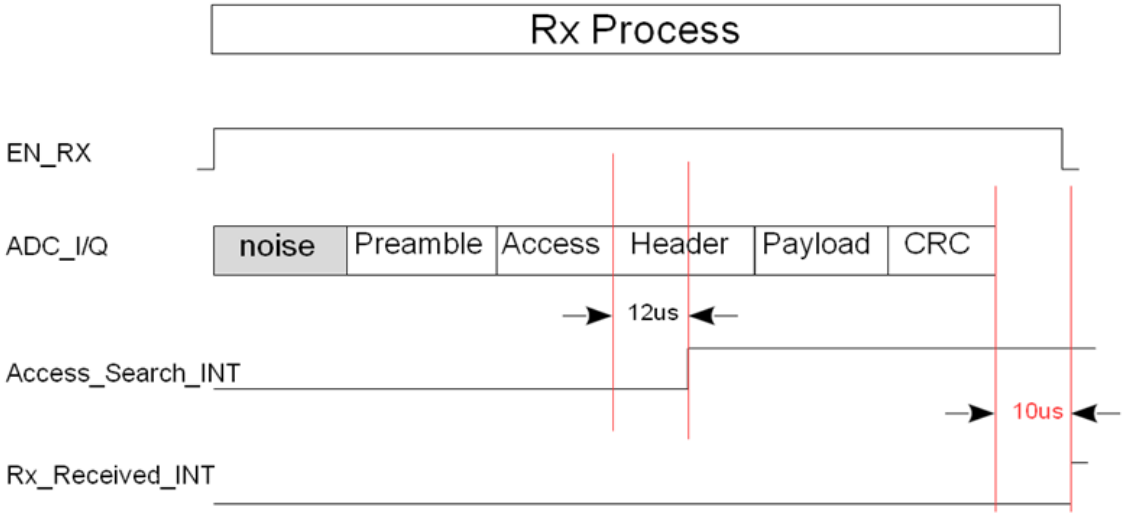


Figure 4. Rx Process Timing

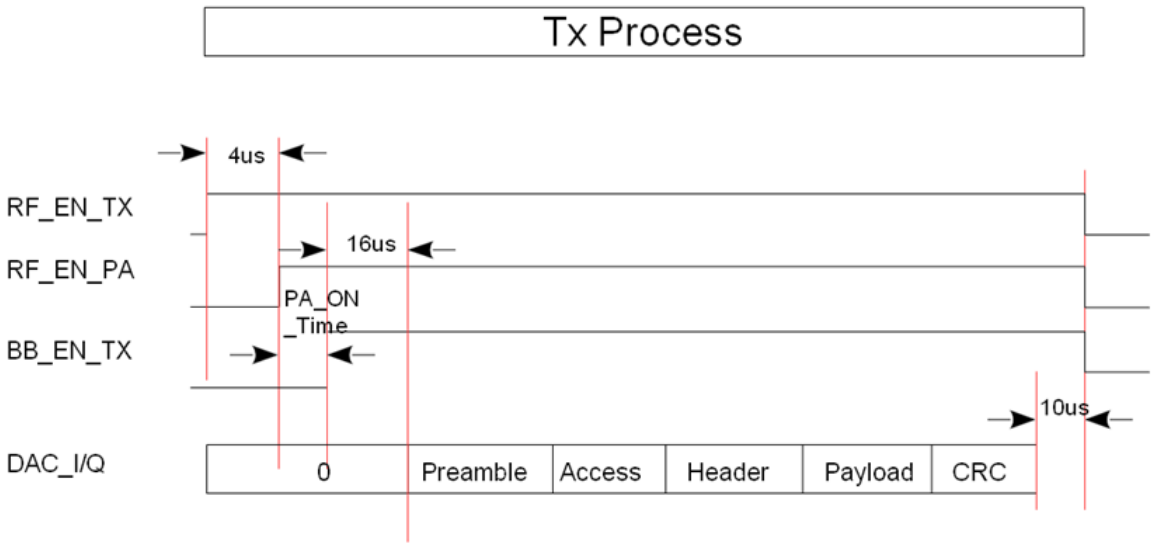


Figure 5. Tx Process Timing

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

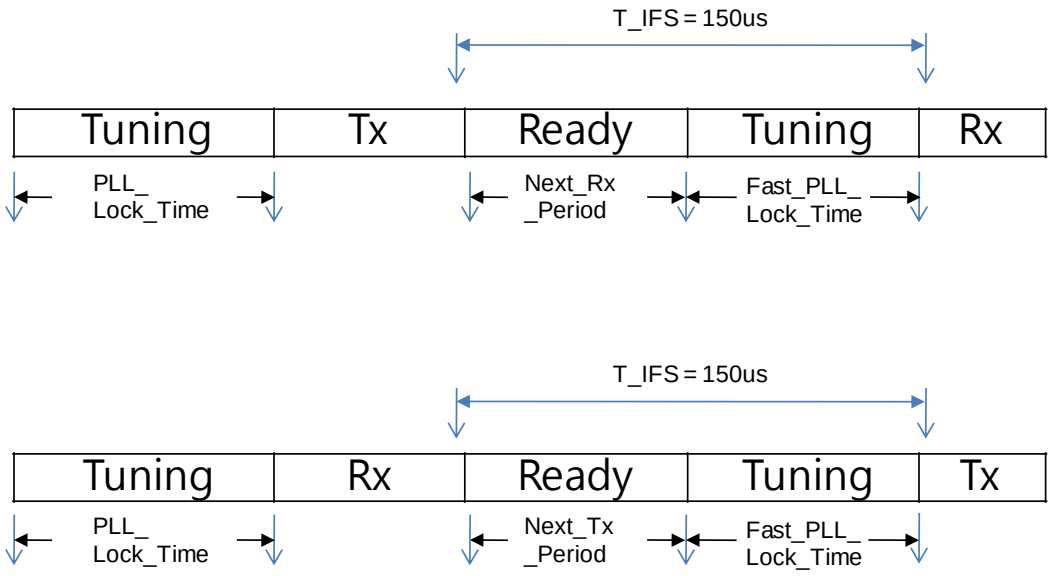


Figure 6. TX/RX Switch Timing

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

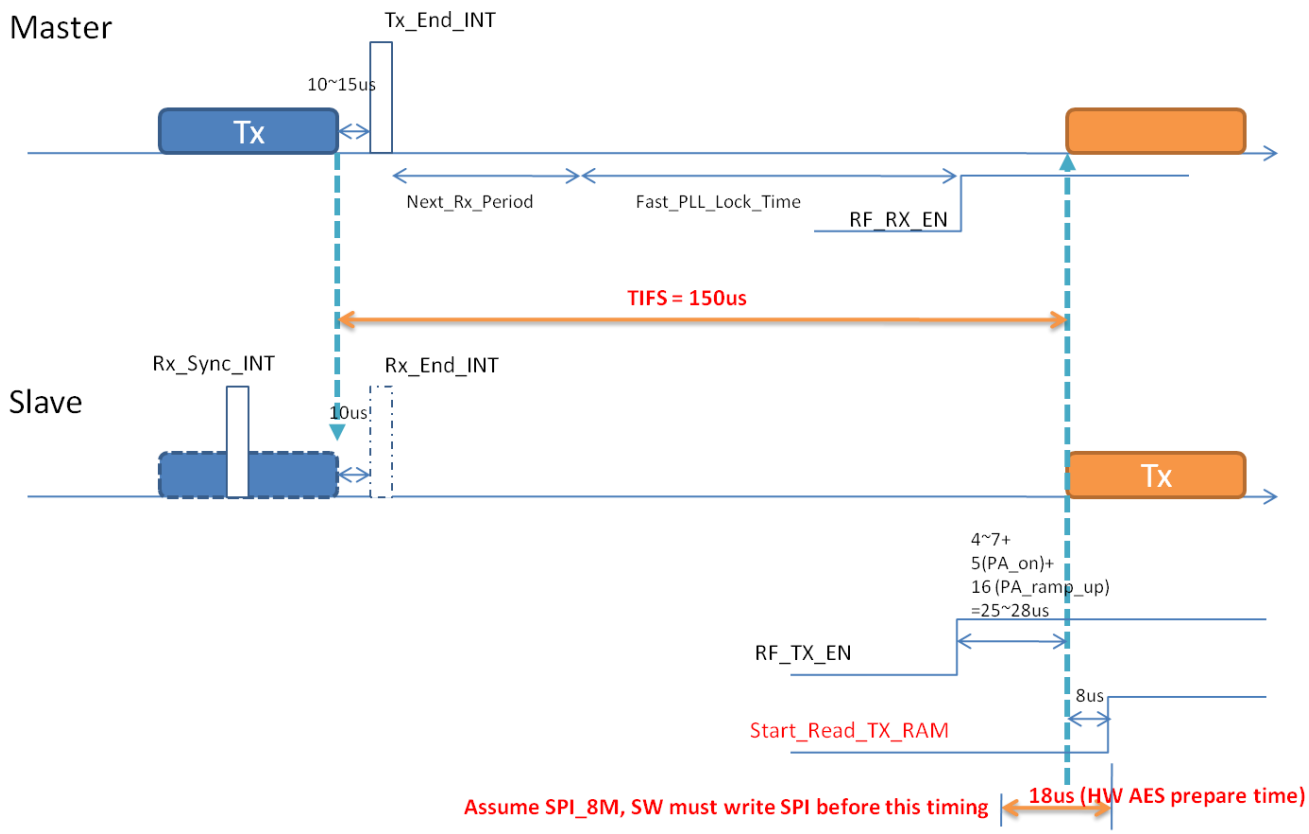


Figure 7. Rx→TIFS→Tx Processing Time

Figure 6 shows the timing for Rx→TIFS→Tx scenario.

Start_Read_TX_RAM rising edge is the time that RT568 starts reading the TX_Buffer data. The AES-CCM engine needs some preparation time. Software must fill ≧18 bytes of data into the Tx_Buffer before the rising edge of Start_Read_TX_RAM. If the total data length is < 18 bytes, just send it the data before the rising edge of Start_Read_TX_RAM.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

10. AES-CCM Encryption / Decryption

AES-CCM Encryption

In order to do HW-based AES-CCM encryption on transmitted data, perform the following steps:
(1) write the TX_packet into the TX_Buffer
(2) write AES_CCM_Nonce into Reg_126 ~ Reg_138
(3) write AES_CCM_Key into Reg_139 ~ Reg_154
(4) enable AES_CCM_mode Reg_155[2:0] = 1

Steps (1) ~ (4) do not need to be executed in that specific order.
In the case of automatic Rx→TIFS→Tx switching: steps (1) ~ (4) must be done before **Start_Read_TX_RAM** is pulled high (see Figure 7, previous page).
In the case of manual transmission control: (1) ~ (4) must be done before calling **rafael_config_tx_enable_tx()**.
Header byte1 is the payload length. If the transmission packet is an empty packet, even if AES_CCM_mode is enabled the RT568 will not perform AES-CCM encryption.

AES-CCM Decryption

In order to do HW-based AES-CCM decryption on received data, perform the following steps:
(1) write AES_CCM_Nonce into Reg_126 ~ Reg_138
(2) write AES_CCM_Key into Reg_139 ~ Reg_154
(3) enable AES_CCM_mode Reg_155[2:0] = 1

Steps (1) ~ (3) do not need to be executed in that specific order.
In the case of automatic Rx→TIFS→Tx switching: steps (1) ~ (3) must be done before the 1st preamble bit is received.
In the case of manual reception: steps (1) ~ (3) must be done before calling **rafael_config_rx_enable_rx()**.
Header byte1 is the payload length. If the received packet is an empty packet, even if AES_CCM_mode is enabled the RT568 will not perform AES-CCM decryption.

The received packet format is shown in Figure 2. Header byte1 shows the raw payload length not including 4 bytes MIC length.
Reg_155[4] and Reg_155[5] indicate the MIC and CRC correctness of the received packet.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

AES Registers

Register Address	Name	Description
Reg_126	AES_CCM_NONCE_0	BLE 5.0 AES-CCM Nonce Byte 0
Reg_127	AES_CCM_NONCE_1	BLE 5.0 AES-CCM Nonce Byte 1
Reg_128	AES_CCM_NONCE_2	BLE 5.0 AES-CCM Nonce Byte 2
Reg_129	AES_CCM_NONCE_3	BLE 5.0 AES-CCM Nonce Byte 3
Reg_130	AES_CCM_NONCE_4	BLE 5.0 AES-CCM Nonce Byte 4
Reg_131	AES_CCM_NONCE_5	BLE 5.0 AES-CCM Nonce Byte 5
Reg_132	AES_CCM_NONCE_6	BLE 5.0 AES-CCM Nonce Byte 6
Reg_133	AES_CCM_NONCE_7	BLE 5.0 AES-CCM Nonce Byte 7
Reg_134	AES_CCM_NONCE_8	BLE 5.0 AES-CCM Nonce Byte 8
Reg_135	AES_CCM_NONCE_9	BLE 5.0 AES-CCM Nonce Byte 9
Reg_136	AES_CCM_NONCE_10	BLE 5.0 AES-CCM Nonce Byte 10
Reg_137	AES_CCM_NONCE_11	BLE 5.0 AES-CCM Nonce Byte 11
Reg_138	AES_CCM_NONCE_12	BLE 5.0 AES-CCM Nonce Byte 12
Reg_139	AES_CCM_KEY_0	BLE 5.0 AES-CCM Key Byte 0
Reg_140	AES_CCM_KEY_1	BLE 5.0 AES-CCM Key Byte 1
Reg_141	AES_CCM_KEY_2	BLE 5.0 AES-CCM Key Byte 2
Reg_142	AES_CCM_KEY_3	BLE 5.0 AES-CCM Key Byte 3
Reg_143	AES_CCM_KEY_4	BLE 5.0 AES-CCM Key Byte 4
Reg_144	AES_CCM_KEY_5	BLE 5.0 AES-CCM Key Byte 5
Reg_145	AES_CCM_KEY_6	BLE 5.0 AES-CCM Key Byte 6
Reg_146	AES_CCM_KEY_7	BLE 5.0 AES-CCM Key Byte 7
Reg_147	AES_CCM_KEY_8	BLE 5.0 AES-CCM Key Byte 8
Reg_148	AES_CCM_KEY_9	BLE 5.0 AES-CCM Key Byte 9
Reg_149	AES_CCM_KEY_10	BLE 5.0 AES-CCM Key Byte 10
Reg_150	AES_CCM_KEY_11	BLE 5.0 AES-CCM Key Byte 11
Reg_151	AES_CCM_KEY_12	BLE 5.0 AES-CCM Key Byte 12
Reg_152	AES_CCM_KEY_13	BLE 5.0 AES-CCM Key Byte 13
Reg_153	AES_CCM_KEY_14	BLE 5.0 AES-CCM Key Byte 14
Reg_154	AES_CCM_KEY_15	BLE 5.0 AES-CCM Key Byte 15
Reg_155	B2 ~ B0 : AES_CCM_Mode B3: Manual_EN_AES B4 : RX_MIC_OK	[B2:B0] AES_CCM_MODE : 0 : bypass mode, no encryption. 1 : encryption / decryption mode. 2 : Tx encryption / Rx bypass

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

	B5 : RX_CRC_OK B6 : AES_128_BUSY B7 : TX successful	3: Rx decryption / Tx bypass 4 : Local AES-128 mode, write 16 bytes plain text to TX Buffer & enable AES by setting Manual_EN_AES, read result at RX FIFO B[3] Manual_EN_AES : in Local AES-128 mode, write 1 to enable AES-128 Encryption; when RX FIFO Cnt = 16, write 0 to disable AES-128 function B[4] RX_MIC_OK : 1, RX MIC correct; 0, RX MIC error B[5] RX_CRC_OK: 1, RX CRC correct; 0, RX CRC error B[6] AES_128_BUSY: 1 : AES and CCM modules are busy B[7] TX_successful: 0 : Tx data successfully sent without Tx_buffer underflow; 1 : TX underflow

Other Registers

Register Address	Name	Description
Reg_47	Bit[0] : Enable 16M clock output	0: on; 1: off
Reg_165	MAX_Payload_LEN[7:0]	RX maximum valid payload length. If decoded header length exceed this number, HW view received packet fail and report CRC error.
Reg_250	Bit[0] : EN DPI Mode Bit[7:1] : Reserved	0: SPI operates in normal SPI mode 1: SPI operates in dual I/O mode

Gain and RSSI related Registers

Register Address	Name	Description
Reg_166	Bit[7:4] : Latch LNA_GAIN	0 : -62.7 dB 1 : -61.2 dB 2 : -59.2 dB 3 : -57.2 dB 4 : -55.5 dB 5 : -53.5 dB 6 : -51.1 dB 7 : -49 dB 8 : -47.2 dB 9 : -45.4 dB 10 : -44.1 dB 11 : -41.4 dB 12 : -38.7 dB 13 : -36.9 dB 14 : -35.7 dB 15 : -35 dB
Reg_167	Bit[7:4] : Latch VGA_GAIN	0 : -6 dB 1 : -3 dB 2 : 0 dB 3 : LPF 2nd +3dB 4 : LPF 2nd +6dB 5 : LPF 2nd +9dB 6 : +3 dB 7 : +6 dB 8 : +9 dB 9 : +12 dB 10 : +15 dB 11 : +18 dB 12 : +21 dB 13 : +24 dB 14 : same as 13 15 : same as 13

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

	Bit[3:0] : Latch TIA_GAIN	0 : LPF 0 dB 8 : 0 dB 1 : LPF +3 dB 9 : +3 dB 2 : LPF +6 dB 10 : +6 dB 3 : LPF +9 dB 11 : +9dB 4 : LPF +12 dB 12 : +12 dB 5 : LPF +15 dB 13 : +15 dB 6 : same as 5 14 : +18 dB 7 : 0 dB 15 : same as 14
Reg_168	Latch ADC_RSSI[7:0]	Signed Value of Base Band ADC RSSI Range: 0dBm + Reg_90 to -60dBm + Reg_90
Reg_90	ADC_RSSI_BASE[7:0]	RSSI Base compensation

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

RT568 API Files and Functions

RT568 physical layer files:

Filename	Description
rafael_phy.c	RT568 basic API
rafael_phy.h	Header file
BlePhysicalLayer.c	RT568 physical layer API
BlePhysicalLayer.h	Header file

RT568 basic API

- void **rafael_spi_write_single**(SPI_T *spi_instance, uint8_t reg_address, uint8_t reg_value);

- Purpose: write a value to one register
- Input 1: *spi_instance: SPI_handle
- Input 2: reg_address
- Input 3: reg_value

*The implementation depends on MCU platform.

- void **rafael_spi_write**(SPI_T *spi_instance, uint8_t reg_address, uint8_t *p_tx_data, uint32_t tx_data_length);

- Purpose: write consecutive values from the start register address
- Input 1: *spi_instance: SPI_handle
- Input 2: reg_address: start register address
- Input 3: *p_tx_data: data values to write
- Input 4: tx_data_length: length of data to be written

*The implementation depends on MCU platform.

- void **rafael_spi_read**(SPI_T *spi_instance, uint8_t reg_address, uint8_t *p_rx_data, uint32_t rx_data_length);

- Purpose: read consecutive values from the start register address
- Input 1: *spi_instance: SPI_handle
- Input 2: reg_address: start register address
- Input 3: *p_rx_data: data values read from registers
- Input 4: rx_data_length: length of data to be read

*The implementation depends on MCU platform.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

- void **rafael_spi_buffer_write**(SPI_T *spi_instance, uint8_t *p_tx_data, uint16_t tx_data_length, uint16_t ram_start_addr, uint8_t fill_payload_mode);

- Purpose: write header or payload to the TX_Buffer
- Input 1: *spi_instance: SPI_handle
- Input 2: *p_tx_data: data values to write
- Input 3: tx_data_length: length of data to be written
- Input 4: ram_start_addr: start of RAM address to write the data
- Input 5: fill_payload_mode: fill this bit to 4th Byte bit7. 1: fill payload data; 0: fill header data. This bit is effective when R105[7]=1 (enable TX buffer underflow check)

*The implementation depends on MCU platform.

- void **rafael_spi_buffer_read**(SPI_T *spi_instance, uint8_t *p_rx_data, uint16_t rx_data_length);

- Purpose: read data from the RX_FIFO.
- Input 1: *spi_instance: SPI_handle
- Input 2: *p_rx_data: data values read from RX_FIFO
- Input 3: rx_data_length: length of reading data

*The implementation depends on MCU platform.

- void **rafael_spi_write_startAddr**(SPI_T *spi_instance, uint16_t ram_start_addr);

- Purpose: Notify RF the address to start reading data from TX_Buffer.
- Input 1: *spi_instance: SPI_handle
- Input 2: * ram_start_addr: Tx_buffer start address for RF to transmit data

- void **rafael_spi_clear_fifo**(SPI_T *spi_instance);

- Purpose: clear RX_FIFO and reset RX_FIFO_counter
- Input 1: *spi_instance: SPI_handle

- void **RAFAEL_Init**(void);

- Purpose: initialize RT568

- void **rafael_config_tx_enable_tx**(SPI_T *spi_instance);

- Purpose: enable TX process. RT568 HW executes PLL lock, preamble append, whitening, add CRC and transmit data.
- Input 1: *spi_instance: SPI_handle

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

- void **rafael_config_rx_enable_rx**(SPI_T *spi_instance);

- Purpose: enable RX process. The RT568 executes PLL lock, access address searching, de-whitening, CRC check and receives data
- Input 1: *spi_instance: SPI_handle

- void **rafael_reset_phy_fsm**(SPI_T *spi_instance);

- Purpose: reset PMU Finite State Machine to Ready state
- Input 1: *spi_instance: SPI_handle

- void **rafael_read_rtc_timer**(SPI_T *spi_instance, uint32_t* RTC_timer_us_low, uint32_t* RTC_timer_us_high);

- Purpose: read RT568 RTC timer count. The RTC timer count total is 37 bits, each count is 1 micro-second
- Output 1: *RTC_timer_us_low: RTC[31:0], LSB 32 bits
- Output 2: *RTC_timer_us_high: RTC[36:32], MSB 5 bits

- void **rafael_update_wakeup_timer**(SPI_T *spi_instance, uint32_t timeout);

- Purpose: put the RT568 into Sleep mode (32KHz clock on, RTC on) and configure the RTC timeout value for wake-up. When the timeout expires, the RT568 will create a Sync_flag interrupt to inform MCU.
- Output 1: timeout: Sleep duration (µs).

- void **rafael_phy_power_down**(uint32_t RtcTimeout);

- Purpose: put the RT568 into Sleep mode (32KHz clock on, RTC on) or Deep Sleep mode (32KHz clock off, RTC off).
- Output 1: RTC Timeout: Sleep duration (µs). Only useful in Sleep mode.

- void **rafael_phy_manual_wakeup**(void);

- Purpose: Immediately wake-up the RT568 from Deep Sleep mode (32KHz clock off, RTC off).

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

Physical Layer API

- void **BLEPHYSICALLAYER_ConfigureRadio** (U32 accessAddress, U8whiteningEnable,U32 crcInit, U8 channel, U8 txPhy, U8 rxPhy, U8 autoSwitchModeEnable);

- Purpose: configure PHY parameters for a TX or RX process
- Input 1: accessAddress: four bytes Access Address. Little Endian format in API. For Advertising, it is always set to 0x71764129.
- Input 2: whiteningEnable: default = 1 (whitening is enabled)
- Input 3: crcInit: three bytes CRC initial values. Little Endian format in API. For Advertising, it is always set to 0x555555.
- Input 4: channel: BLE link layer channel number. Range: 0~39. The RT568 will configure the PLL according to this parameter.
- Input 5: txPhy: TX symbol rate: 1 = 1MHz, 2 = 2MHz
- Input 6: rxPhy: RX symbol rate: 1 = 1MHz, 2 = 2MHz
- Input 7: autoSwitchModeEnable: 1 = enable Auto T/R switch, 0 = disable

- void **BLEPHYSICALLAYER_StartRx** (void);

- Purpose: Start scanning on the radio channel previously configured by BLEPHYSICALLAYER_ConfigureRadio(). The RT568 will generate an interrupt when the access address is matched or a complete packet is received. The packet is stored in the RX_FIFO.

- void **BLEPHYSICALLAYER_SetRxTimeout** (void);

- Purpose: Enable RxTimeout and set timeout value (255μs). It is enabled for TX/RX auto switch mode. The RT568 generates an interrupt if no peer response is received in the TX/RX or RX/TX/RX sequence.

- void **BLEPHYSICALLAYER_StopRF**(void);

- Purpose: stop current radio operation (RX or TX). It can also be called to stop TX/RX auto-switch mode.

- void **BLEPHYSICALLAYER_SetTxPacket**(const U8 *pHeader, const U8 headerLength, const U8 *pData, const U8 dataLength);

- Purpose: put a TX packet into the TX_FIFO before transmitting it.
- Input 1: *pHeader: header data

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

- Input 2: headerLength: header length
- Input 3: *pData: payload data
- Input 4: dataLength: data length

*The implementation depends on MCU platform.

- void **BLEPHYSICALLAYER_StartTx** (void);

- Purpose: Transmit a previously buffered packet. The RT568 will generate an interrupt after the entire packet is transmitted.

- S8 **BLEPHYSICALLAYER_GetTXPowerLevel** (void);

- Purpose: Get the Tx power level
- Return: Tx power level in dBm

- S8 **BLEPHYSICALLAYER_GetLastRSSI** (void);

- Purpose: Get the latest RSSI.
- Return: RSSI value in dBm

<intentionally blank>

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.

Contact Information

Address	8F., No.28, Chenggong 12th St., Zhubei City, Hsinchu County 30264 Taiwan R.O.C.		
Contact	Tel: 886-3-5506258	Fax: 886-3-5506228	www.rafaelmicro.com
Sales	**World-wide**	886-3-5506258 ext. 206	michael.gauer@rafaelmicro.com
	China	86-1360-2679953	jaff.wu@rafaelmicro.com

Revision History

Revision	Description	Owner	Date
1.0	First preliminary draft	Michael Gauer	2019/11/27

© 2019 by Rafael Microelectronics, Inc.

All Rights Reserved.

Information in this document is provided in connection with **Rafael Microelectronics, Inc.** ("**Rafael Micro**") products. These materials are provided by **Rafael Micro** as a service to its customers and may be used for informational purposes only. **Rafael Micro** assumes no responsibility for errors or omissions in these materials. **Rafael Micro** may make changes to this document at any time, without notice. **Rafael Micro** advises all customers to ensure that they have the latest version of this document and to verify, before placing orders, that information being relied on is current and complete. **Rafael Micro** makes no commitment to update the information and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to its specifications and product descriptions.

THESE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, RELATING TO SALE AND/OR USE OF **RAFAEL MICRO** PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, CONSEQUENTIAL OR INCIDENTAL DAMAGES, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. **RAFAEL MICRO** FURTHER DOES NOT WARRANT THE ACCURACY OR COMPLETENESS OF THE INFORMATION, TEXT, GRAPHICS OR OTHER ITEMS CONTAINED WITHIN THESE MATERIALS. **RAFAEL MICRO** SHALL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS.

Rafael Micro products are not intended for use in medical, lifesaving or life sustaining applications. **Rafael Micro** customers using or selling **Rafael Micro** products for use in such applications do so at their own risk and agree to fully indemnify **Rafael Micro** for any damages resulting from such improper use or sale. **Rafael Micro, logos** and **RT568** are **Trademarks** of **Rafael Microelectronics, Inc**. Product names or services listed in this publication are for identification purposes only, and may be trademarks of third parties. Third-party brands and names are the property of their respective owners.

The information contained herein is the exclusive property of Rafael Microelectronics, Inc. and shall not be distributed, reproduced or disclosed in whole or in part without prior written permission of Rafael Microelectronics, Inc.