RW-BLE Core

Functional Specifications

RW-BLE-CORE-FS

Version 8.0.05

May 13, 2015

RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	Date	Revision Description	Author
8.0.01	February 2, 2015	Initial release / (Ported from BLE Core version 7.0.24)	JPL
8.0.02	February 17, 2015	Feature #3900 / Added Enhanced Privacy v1.2 support Added Resolving Address List engine section Added Resolving Address List structure Updated AES-CCM section Updated Control Structure / Rx Descriptors Added registers for RAL support Feature #4197 / Moved SAMP field in BASETIMECNT register Feature #4408 / Updated WLAN coexistence interface Feature #4530 / Updated RWBLECONF register content Feature #4537 / Added EM clock gating enable output Feature #4539 / Extended RFTESTCNTL use to all event type Feature #4550 / Removed preamble from correlation Feature #4552 / Extended TXPWRDN register field by 1 bit Feature #4554 / Deprecated CSEM Atlas RF Updated register section Updated IO Port map	JPL
8.0.03	March 2, 2015	Included review comments from FB Updated RPA FSM Renamed CS-LOCAL_RPA into CS-LOCAL_RPA_SEL Added missing MSCs / Closed TBD-2	JPL
	March 5, 2015	Feature #4537 / Updated Validation table related to EM clock gating enable control support	JPL
	March 10, 2015	Included review comments from SB	JPL
8.0.04	March 16, 2015	Feature #3900 / Added a separate EM port for RAL engine, and refined RAL controls description	JPL
	April 1, 2015	Feature #3900 / Updated Validation table with RAL related signals / Updated RAL FSM / Clarified some RAL text / Added RAL underrun error report	JPL
	April 28, 2015	Feature #3900 / Removed RAL Filtering enable control	JPL
	May 7, 2015	Added signal availability details regarding Diagnostic Port table	JPL
8.0.05	May 13, 2015	Feature#4449 / Updated COEXIFCNTL0 register content for Dual Mode WLAN/MWS purpose / Corrected some editorials	JPL

Changes between a version and the previous one is reflected by the addition of **change bars**, like for the line below:

…This line has been modified from previous version…

Items to be Determined

As this specification is not yet a final version, the items summarize in the table below remain to be clarified or to be determined in later releases.

TBD	Date entered	Description	Status
TBD-1	February 2, 2015	Clarify Resolving Address List block behavior and operating modes	Closed
TBD-2	February 20, 2015	Don't forget to add missing MSCs for Enhanced Privacy v1.2 in section 3.7.2.4.2 ➔ done	Closed

Table of Contents

List of Figures

List of Tables

List of Registers

List of Acronyms and Abbreviations

Acronym or abbreviation	Writing out in full / Meaning
AES	Advanced Encryption Standard
AHB	Advanced High Performance Bus
ASIC	Application-Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BD_ADDR	Bluetooth Device Address
BLE	Bluetooth Low Energy
BR	Basic Rate
CCM	Counter with CBC-MAC (Cipher Block Chaining – Message Authentication Code)
CPU	Core Processing Unit
CRC	Cyclic Redundancy Check
CRM	Clock Reset Manager
CS	Control Structure
DM	Dual Mode
DFT	Design For Test
EDR	Enhanced Data Rate
EM	Exchange Memory
ET	Exchange Table
FDMA	Frequency Division Multiple Access
FPGA	Field Programmable Gate Array
HD	High Duty (Cycle Directed Advertising)
IP	Intellectual Property
IRK	Identity Resolving Key
LD	Low Duty (Cycle Directed Advertising)
LLID	Logical Link Identifier
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
MD	More Data
MIC	Message Integrity Check
MSB	Most Significant Bit
MSC	Message Sequence Chart
MWS	Mobile Wireless System
NESN	Next Sequence Number
PER	Packet Error Rate
PDU	Protocol Data Unit
PPM	Parts Per Million
PRBS	Pseudo Random Binary Sequence
RAL	Resolving Address List
RF	Radio Frequency
RFU	Reserved for Future Use
RPA	Resolvable Private Address
RSSI	Received Signal Strength Indication
RTC	Real Time Counter
Rx	Reception
RxADD	Received Bluetooth Device Address Public / Private indicator
RxDESC	Reception Descriptor (used for data buffers)
SPI	Serial Port Interface
SN	Sequence Number
SoC	System On Chip
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
Tx	Transmit

TxADD	Transmitted Bluetooth Device Address Public / Private indicator
TxDESC	Transmit Descriptor (used for data buffers)
UPF	Un-Plugged Festival (Interoperability Event organized by the Bluetooth SIG)
WL	White List
WLAN	Wireless Local Area Network
X-Bar	Cross-Bar Switch

1 Overview

1.1 Document overview

This document describes the functioning of the RW-BLE Core, and details its architecture. It is intended for use by designers to design the Bluetooth Low Energy Digital Core, and for integrators who want to understand the architecture and functionality.

1.2 Product overview

The RW-BLE Core has the following features:

- Bluetooth Low Energy v4.2 Specifications compliant (See [1])
- Support AMBA AHB bus (See [2]) or X-Bar bus (See [3])
- All packet types (Broadcasting / Advertising / Data / Control)
- Encryption / Decryption (AES-CCM)
- Bit stream processing (CRC, Whitening)
- Frequency Hopping calculation
- FDMA / TDMA / events formatting and synchronization
- WLAN Coexistence mechanism (Signaling)
- All device classes support (Broadcaster, Central, Observer, Peripheral)
- Low power modes supporting 32.0kHz or 32.768kHz low–power clock frequencies
- Support of different radio chips via a customized interface
- Designed for easy integration into an ASIC
- Designed in synthesizable RTL code for easy technology migration
- DFT included, accepted by major ATPG tools
- Simple, clean and optimized interface to RW-BLE Software
- Low power consumption
- Low frequency

Bluetooth Low Energy v4.0 features are fully supported.

Bluetooth Low Energy v4.1 features are fully supported.

1.3 Writing convention

The fields of the Exchange Table are identified in the following way: **ET–<FieldName>**.

The fields of the Control Structure are identified in the following way: **CS–<FieldName>**.

The fields of the Tx/Rx Descriptor are identified in the following way: **Tx/RxDESC–<FieldName>**.

The fields of the Resolving Address List structure are identified in the following way: **RAL–<FieldName>**.

The registers are labeled like this: **<RegName>–<FieldName>**.

2 Architecture

2.1 Top–level architecture

Figure 2-1 shows the top–level block diagram of the RW-BLE Core, along with the main surrounding blocks when it is integrated into an ASIC.

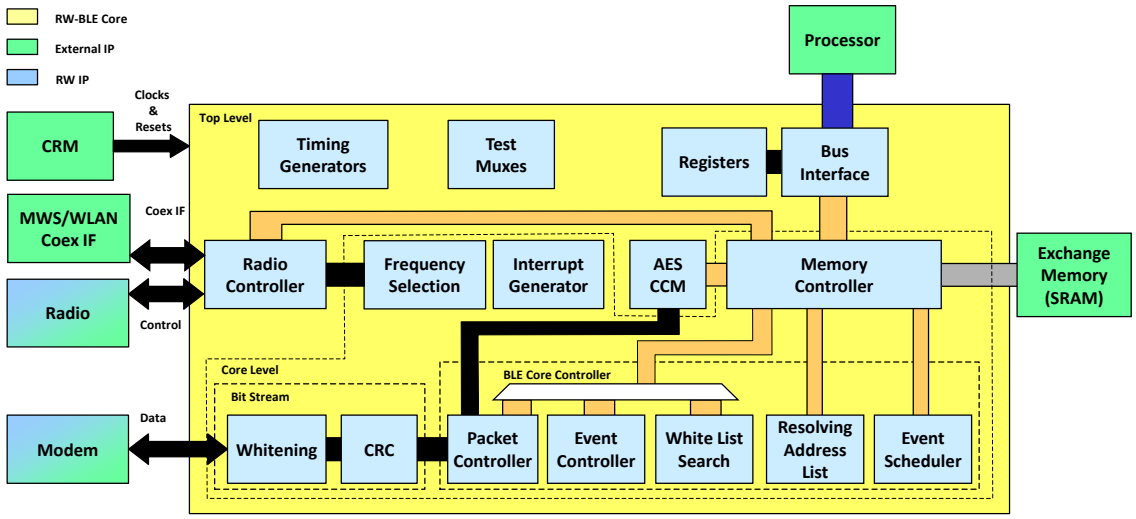

Figure 2-1 – RW-BLE Core Block Diagram

It interfaces to either an AHB bus as a slave, or an X-Bar bus as a slave, connected to a processor platform. This bus ensures that data can be moved quickly between the processor and the Exchange Memory. A radio component is plugged on the corresponding interface, to convert baseband signals to radio frequency and vice–versa. The RW-BLE Core interface is digital and some additional converters may be required to interface to specific RF chips.

Furthermore, the RW-BLE Core level has an internal Core level that contains all the blocks except the Timing Generator, the Bus Interface, the Radio Controller, the AES-CCM encryption/decryption engine, and the Diagnostic Port. This can be useful for different implementation between FPGA and SoC device target.

As example, in FPGA the top level is directly instantiated, while in a SoC only the core level can be used; in order to let flexibility to move the Timing Generator either into RTC (Real Time Counter) or into CRM (Clock Reset Manager) unit, and to move the Memory Controller and the Bus Interface in the CPU power domain. This allows the Processor to access to RW-BLE registers while the RW-BLE Core is powered off: this helps power island/domain integration.

It is also possible, by the mean of a specific define parameter, to extract the low power block of the timing generator (i.e. that runs on low power clock) in order to be more flexible for specific power gating strategy.

2.2 Configuration Parameters

Table 2-1 lists the main block configuration parameters.

Configuration Parameter Name	Description
RW_BLE_XBAR_BUS	Instantiate X-Bar bus Interface support in place of AHB Bus Interface
RW_BLE_CPU_HIGHEST_PRIORITY	Defines CPU access priority to the Exchange Memory
RW_BLE_ADDRESS_WIDTH	AHB / X-Bar / EM Bus Address width – Represents the number of bytes in the EM
RW_BLE_DATA_WIDTH_CONF_<16/32>	Processor / Memory Data Bus width, select between 16 or 32 bits width.

	Note this parameter is supported with AHB Bus only
RW_BLE_MAX_FREQUENCY	Maximum Frequency Value, integer with 32 as max frequency
RW_BLE_AHB_TO_EM_PATH_ENABLE	Enable AHB to EM path / removes related logic when not set
RW_BLE_INT_MODE_<PULSE/LEVEL>	Select in between interrupt provided as pulse or level triggered
RW_BLE_TIMING_GEN_LP_EXTERNAL	Allows to instantiate timing Generator Low Power block outside the RW-BLE Core
RW_BLE_CRYPT_INST	AES-CCM block instantiation / removes related logic when not set
RW_BLE_AES_DECIPHER_INST	AES decipher block instantiation / removes related logic when not set
RW_BLE_RIPPLE_INST	RivieraWaves Ripple Radio Interface Supported / removes related logic when not set
RW_BLE_EXTRC_INST	External Radio Controller Interface Supported / removes related logic when not set
RW_BLE_ICYTRX_INST	RivieraWaves IcyTRx Radio Interface supported / removes related logic when not set
RW_BLE_WLAN_COEX_INST	WLAN Coexistence Interface Instantiation / removed related logic when not set
RW_BLE_DIAG_INST	Allows instantiation of Diagnostic Ports / removed related logic when not set

Table 2-1 – Configuration Parameter List

For further details please refer to [4].

2.3 I/O Ports

Table 2-2 lists the main block interfaces that need to be connected to the RW-BLE Core.

Block Interface	Description
CRM	Clock and Reset Manager
PROC	Processor AHB platform, or X-Bar platform, that runs the RW-BLE Core Software stack
RF	RW-BLE Core compliant Radio
EM	Exchange Memory
TGLP	Timing Generator Low Power block
COEX	MWS/WLAN Coexistence interface
DBG	Debug Interface

Table 2-2 – Interface List

Table 2-3 describes the RW-BLE Core I/O ports. The type of each port is also indicated (In/Out and the number of wires for already defined busses).

Port Name	I/O	Width	Interface	Description
Clock And Resets				
master_nrst	I	1	CRM	Main asynchronous reset, active low
crypt_nrst	I	1	CRM	Encryption / Decryption asynchronous reset, active low Exists if *RW_BLE_CRYPT_INST* is defined
low_power_nrst	I	1	CRM	Low Power asynchronous reset, active low *Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is not defined*
hreset_n	I	1	CRM	CPU interface asynchronous reset, active low Used for both AHB and X-Bar Bus
master1_gclken	O	1	CRM	Primary Master Clock enable, active high *Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is not defined*
master1_gclk (1)	I	1	CRM	Primary Master Gated Clock, rising edge active.
master2_gclken	O	1	CRM	Secondary Master Clock enable, active high
master2_gclk (1)	I	1	CRM	Secondary Master Gated Clock, rising edge active.
crypt_gclken	O	1	CRM	Encryption / Decryption Clock Enable, active high
crypt_gclk (1)	I	1	CRM	Encryption / Decryption Clock, rising edge active

				Exists if RW_BLE_CRYPT_INST is defined
low_power_clk	I	1	CRM	Low Power Clock, rising edge active, 32kHz or 32.768kHz *Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is not defined*
hclk [(2)]	I	1	CRM	CPU interface Clock, rising edge active Used for both AHB and X-Bar Bus
em_gclken	O	1	CRM	EM clock enable control, active high
clk_sel	I	6	PROC	Clock Frequency selection
phase_match_p	I	1	CRM	Indicates whether *master1/2_gclk* and *hclk/p_clk* phases (i.e. rising edges) are aligned, active high pulse.
Sleep Mode control				
clk_status	O	1	CRM	Clock status, defines the current active clock in use 0: System running on *master1/2_gclk* 1: System running on *low_power_clk*
osc_en	O	1	RF	Enable Oscillator / Xtal that drives the RF, active high *Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is not defined*
radio_en	O	1	RF	Enable Radio, active high *Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is not defined*
wakeup_req	I	1	PROC	Wake Up Request, used to sort–out sleep modes, active high
Timing Generator Low Power Interface *(Exists if RW_BLE_TIMING_GEN_LP_EXTERNAL is defined)*				
fine_cnt_32k_samp	I	10	TGLP	Fine counter retention
basetime_cnt_32k_samp	I	27	TGLP	Base Time counter retention
deep_sleep_stat_32k	I	1	TGLP	Deep Sleep status on low power clock domain
wakeup_lp	I	1	TGLP	Low power wake-up
toggle_sleepref	I	1	TGLP	Sleep reference toggle
deep_sleep_on_32k_trig	I	1	TGLP	Deep Sleep On triggered in low power clock domain
deepsldur	I	32	TGLP	Deep Sleep duration in low power clock cycle
extwkupdsb	O	1	TGLP	External Wake Up disable control
finecnt	O	10	TGLP	Fine Counter current value
basetimecnt	O	27	TGLP	Base time counter current value
deep_sleep_stat	O	1	TGLP	Deep Sleep status
wakeup_req_trig	O	1	TGLP	External wake-up triggered
soft_wakeup_req	O	1	TGLP	Software wake-up request
deep_sleep_corr_en	O	1	TGLP	Fine / Base time counter correction enabled
deep_sleep_on	O	1	TGLP	Deep Sleep request from SW
radio_sleep_en	O	1	TGLP	Enable *radio_en* control in sleep mode
osc_sleep_en	O	1	TGLP	Enable *osc_en* control in sleep mode
deepsleep_time	O	32	TGLP	Deep Sleep duration time required by SW
twext	O	11	TGLP	Time to wake-up *osc_en* on external wake-up request
twosc	O	11	TGLP	Time to wake-up *osc_en* before *deepsleep_time* expiration
twrm	O	10	TGLP	Time to wake-up radio module
AHB Interface *(Exists if RW_BLE_XBAR_BUS is not defined)*				
hsel	I	1	PROC	AHB select. Indicates RW-BLE Core is selected
haddr	I	17 [(3)]	PROC	AHB Address
htrans	I	2	PROC	AHB transfer type
hwrite	I	1	PROC	AHB Write enable
hsize	I	3	PROC	AHB access size (8, 16, 32 bits)
hwdata	I	16/32 [(4)]	PROC	AHB write data bus
hrdata	O	16/32 [(4)]	PROC	AHB read data bus
hready_in	I	1	PROC	AHB ready flag input
hready_out	O	1	PROC	AHB ready flag output
hresp	O	2	PROC	AHB transfer status
X-Bar Interface *(Exists if RW_BLE_XBAR_BUS is defined)*				
p_ben	I	4	PROC	X-Bar byte enable controls
p_ad	I	17 [(3)]	PROC	X-Bar address
p_wr	I	1	PROC	X-Bar write enable
p_rd	I	1	PROC	X-Bar read enable

p_wdata	I	16/32 [4]	PROC	X-Bar write data bus
p_rdata	O	16/32 [4]	PROC	X-Bar read data bus
p_wait	O	1	PROC	X-Bar wait control
Interrupts				
ble_cscnt_irq	O	1	PROC	625µs base time reference interrupt, generated each 625µs in active mode.
ble_slp_irq	O	1	PROC	Sleep Mode Interrupt, generated at end of sleep period.
ble_rx_irq	O	1	PROC	Received Packet Interrupt, generated on receipt.
ble_event_irq	O	1	PROC	Event Interrupt, generated each time advertising / scanning / connection event is terminated (normal or anticipated pre-fetch abort request).
ble_crypt_irq	O	1	PROC	Encryption engine Interrupt, generated when encryption engine ciphering task is terminated, and controlled from registers. *Exists if RW_BLE_CRYPT_INST is defined*
ble_error_irq	O	1	PROC	Error Interrupt, generated when CPU and RW_BT-LE system try to access to same memory space at the same time (as example).
ble_grosstgtim_irq	O	1	PROC	Gross Timer Target interrupt, generated when gross timer target value is expired, with 10ms precision.
ble_finetgtim_irq	O	1	PROC	Fine Timer Target interrupt, generated when fine timer target value is expired, with 625µs precision.
ble_sw_irq	O	1	PROC	SW triggered interrupt, generated on SW request
RF Control Interface				
radio_in	I	48	RF	Generic 48 bits RF Control bus input
radio_out	O	48	RF	Generic 48 bits RF Control bus output
Coexistence Interface (WLAN)				
ble_tx	O	1	COEX	Bluetooth Low Energy transmit indicator *Exists if RW_BLE_WLAN_COEX_INST is defined*
ble_rx	O	1	COEX	Bluetooth Low Energy reception indicator *Exists if RW_BLE_WLAN_COEX_INST is defined*
ble_in_process	O	1	COEX	Bluetooth Low Energy Event in process indicator *Exists if RW_BLE_WLAN_COEX_INST is defined*
ble_pti	O	4	COEX	Bluetooth Low Energy Packet Traffic Information *Exists if RW_BLE_WLAN_COEX_INST is defined*
ble_sync	O	1	COEX	Bluetooth Low Energy 625µs timing pulse reference *Exists if RW_BLE_WLAN_COEX_INST is defined*
wlan_tx	I	1	COEX	WLAN collocated device transmit active *Exists if RW_BLE_WLAN_WLAN_COEX_INST is defined*
wlan_rx	I	1	COEX	WLAN collocated device reception active *Exists if RW_BLE_WLAN_COEX_INST is defined*
Exchange Memory Interface				
em_cs	O	1	EM	Exchange Memory select, active high
em_write	O	1	EM	Global Write enable, active high
em_bl	O	2/4 [4]	EM	Block selection
em_add	O	16 [3]	EM	Address
em_wdata	O	16/32 [4]	EM	Write data
em_rdata	I	16/32 [4]	EM	Read data
em_ready	I	1	EM	Exchange Memory ready indicator, active high.
Diagnostic Port *(Exists if RW_BLE_DIAG_INST is defined)*				
ble_dbg0	O	8	DBG	RW-BLE Core Debug interface port 0
ble_dbg1	O	8	DBG	RW-BLE Core Debug interface port 1
ble_dbg2	O	8	DBG	RW-BLE Core Debug interface port 2
ble_dbg3	O	8	DBG	RW-BLE Core Debug interface port 3

Table 2-3 – IO Port map of the RW-BLE Core

(1): Clock frequency value is defined by *clk_sel* input.

(2): Clock frequency value can be defined as an integer multiple of *clk_sel* input.

(3): Processor Bus address (i.e width is *RW_BLE_ADDRESS_WIDTH*+1) and Exchange Memory address (i.e width is *RW_BLE_ADDRESS_WIDTH*) are defined by configuration parameter. The Processor Address MSB selects either register (when equal to 0), or Exchange Memory (when equal to 1).

(4): Exchange Memory data width set by *RW_BLE_DATA_WIDTH_CONF_<16/32>* parameter. Means EM interface must be connected accordingly, and unused output ports are grounded to logic 0.

2.4 Clocking Strategy

2.4.1 General Considerations

The RW-BLE Core has five different clock inputs:

- The bus clock (i.e. *hclk*), coming from the AHB, or X-Bar bus, is used as a source for the BLE Core's clocking, in order to avoid unnecessary resynchronization logic and add delay in the processor transfers. Its frequency can be equal either to *clk_sel* input, or to a multiple integer of *clk_sel* input
- The encryption clock (i.e. *crypt_gclk*), is used for AES-CCM encryption / decryption of data during connection events. This clock is controlled by the Packet Controller. *crypt_gclk* gating is controlled by *crypt_gclken.* Its frequency is defined by to *clk_sel* input.
- The primary master clock (i.e. *master1_gclk*), is used for active mode operation and scheduling decisions. Its frequency is defined by *clk_sel* input. *master1_gclk* gating is controlled by *master1_gclken*. *master1_gclken* is driven by the Timing Generator.
- The secondary master clock (i.e. *master2_gclk*), is used for active mode operation and real time packet processing. Its frequency is defined by *clk_sel* input. *master2_gclk* gating is controlled by *master2_gclken*. *master2_gclken* is driven by both the Timing Generators and the Event Scheduler, with priority given onto the Timing Generator.
- The low power clock (i.e. *low_power_clk*), is used for deep sleep mode operations, when the system needs to maintain its internal 625µs timing reference, but does not need to process any data.

It is important to be very careful with the clocking strategy since the RW-BLE Core can be used in a variety of designs, using different design flows and tools. The basic clocking scheme is depicted in Figure 2-2. All clock gates are external to the design, and the corresponding enable signals are available at the top–level interface (See Table 2-3).

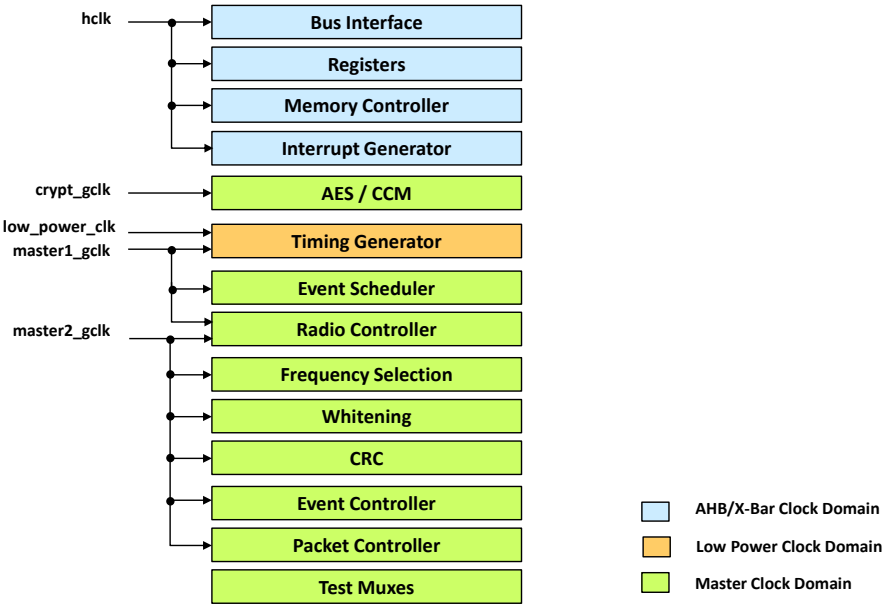

Figure 2-2 – RW-BLE Core Clock Scheme

Please note that

1. Resynchronization mechanisms have to be added in between clocks from different clock domains (i.e. *low_power_clk* versus other clock domains).
2. If *RW_BLE_AHB_TO_EM_PATH_ENABLE* is not set, then Exchange Memory Controller is clocked by *master1_gclk* clock.

Table 2-4 gives the synchronicity relationship between clock domains of the RW-BLE Core

	low_power_clk	**hclk**	**crypt_gclk**	**master1_gclk** **master2_gclk**
low_power_clk	=	asynchronous	asynchronous	asynchronous
hclk	asynchronous	=	synchronous	synchronous
crypt_gclk	asynchronous	synchronous	=	
master1_gclk **master2_gclk**	asynchronous	synchronous	synchronous	=

Table 2-4 – RW-BLE Core Clock to Clock Relations

2.4.2 Modes of Operation

Basically the RW-BLE Core has three modes of operation that enable low power capabilities:

1. "Off" mode, in which no clocks are considered as running, the RW-BLE Core can be completely powered off. *hclk* may, or may not run, as under CPU control. This mode is the default mode when the RW-BLE Core is disabled and has no device(s) to connect with.

2. "Deep-Sleep" Mode, in which only *low_power_clk* clock is considered running, in order to maintain its internal reference clock. *hclk* may, or may not run, as under CPU control. This mode is possible when the RW-BLE core is enabled. It is basically used in between advertising / scanning / connection events.
3. "Active" Mode, in which only *master1_gclk* and *master2_gclk* clocks are considered running, that allows to perform advertising, scanning, connection events, and real time packet processing. *hclk* may run in order to ensure register access by the processor (e.g. Interrupt acknowledgment, etc…). This mode is the active mode of the RW-BLE Core. Please note that depending on the connection (not encrypted or encrypted), *crypt_gclk* may not run as controlled in real time. Note also that *master2_gclk* is running only when Event Scheduler has a valid Exchange Table entry that means any event to perform.

Please refer to section 2.5.2 for "Active" mode operation description.

Please refer to section 2.5.3 for "Off" and "Deep Sleep" operations (i.e Low Power modes) description.

2.4.3 Master Clock Frequency Determination

The frequency of the *master1/2_gclk* clock input is determined by the *clk_sel* input. It is up to the chip designer to either wire the *clk_sel* input to an external control register or hard–code it to a fix frequency value.

The selected frequency must remain a static reference value, set at initialization time. No change of this parameter must occur while the RW-BLE Core is in use.

Note that *clk_sel* input must not be set at a higher value than *RW_BLE_MAX_FREQUENCY* define parameter.

2.5 Timing Generator

2.5.1 Timing References

The timing generators maintain the RW-BLE Core 625µs timing reference, in order to synchronize with RW-BLE Software. This is necessary to support slave operation.

Three counters are required to maintain synchronicity in between two Bluetooth Low Energy devices during connection, which are:

- A 10-bit Fine Counter to determine the exchange memory pre-fetch instant, and start the packets, with a 1µs precision.
- A 27-bit Base Time Counter with a 625µs base time precision.
- A 32-bit Wake–Up Counter that counts the number of *low_power_clk* clock cycles. Please refer to section 3.12.2 for further details.

All the divisions shown in Figure 2-3 are used to generate data enables.

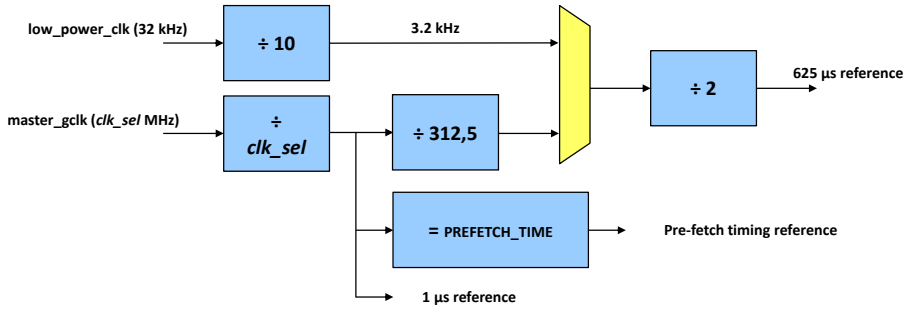

Figure 2-3 – Timing scheme inside the RW-BLE Core

The packet starting point is determined by the 3.2kHz clock which is used for the 625µs internal reference timing generation. Please note this diagram must be understood as a high–level view of the internal reference timing generation of the RW-BLE Core, and does not reflect the current implementation.

Please refer to section 2.5.4 for 32.768kHz low power clock support.

2.5.2 Active Mode

When the device is a master device, Base Time Counter and Wake–Up counter are used for network synchronization, and all slave devices connected to the network must align on these counters.

When the device is a slave device, it resynchronizes onto master's network timing each time a synchronization word is detected. However, the internal counters are not updated.

On each 625µs reference base time event, the RW-BLE Core stores the current Base Time Counter value in BASETIMECNT register. When reading this register, the software is able to align its internal counters with the RW-BLE Core base time reference, and then program the next event correctly. Base time Counter can also be updated by the hardware when exiting from sleep mode if any time compensation is required (See section 2.5.4).

Fine Counter counts between 624 and 0. Base Time Counter counts between 0 and 2^{27}-1.

Each time the Fine Counter equals TIMGENCNTL-PREFETCH_TIME, the Exchange Table is pre-fetched.

Also, depending on the Event Scheduler, *master2_gclk* can be gated while *master1_gclk* runs, as shown in Figure 2-4 below.

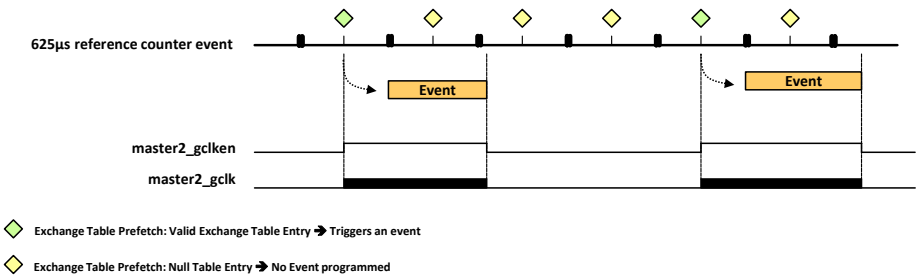

Figure 2-4 – Master Clock Gating control principle

2.5.3 Low Power Modes

Due to the tolerances on the active clock (±50 ppm) and the low power clock (±500 ppm), and depending on the device activity, the resulting maximum drift between a master and a slave can vary.

As example, if the master is in active mode, and the slave in low power mode, then the resulting maximum drift between the devices is 550 ppm in the worst case. As the maximum duration time is 4s between packet exchanges when devices are in connected state, the drift can reach nearly ±4s × $550e^{-6}$ = ±2.2ms, that is roughly equivalent to ±4 times 625µs base time reference.

It is also possible both master and slave to be in sleep mode in between connection events, then the resulting drift between master and slave device is ±1000 ppm worst case. The drift can reach nearly 4s × $1000e^{-6}$ = ±4ms, that is roughly equivalent to ±7 times 625µs base time reference.

This highlights the need of a programmable wide synchronization window opening capability for a slave device.

Worst case is when the maximum possible time interval between packets exchanges (4s) and maximum connection sub-rating (slave latency = 499) are used, then the connection interval theoretically equals to 4 × 499 = 1996s = 33mn16s. Then, wake–Up counter needs to be able to count *low_power_clk* clock cycles up to 63,872,000 (worst case with a 32kHz clock) before waking up the system and opening a new synchronization window, hence at least 31-bit width minimum requirement. Please note that this case never happens as maximum connection interval cannot be higher than link supervision time out delay (i.e. 32s).

However, in order to cope with dual mode implementation, and being able to mimic the “Legacy” Bluetooth native clock (i.e. CLKN[27:1]) and have a higher precision based onto *low_power_clk*, DEEPSLWKUP-DEEPSLTIME is implemented over 32-bit width.

2.5.3.1 Switch from Active Mode to Deep Sleep Mode

By activating the corresponding DEEPSLCNTL-DEEP_SLEEP_ON register bit, the software puts the RW-BLE Core into Deep Sleep mode. The RW-BLE Core then uses *low_power_clk* (usually 32.0kHz or 32.768 kHz) in order to maintain its internal 625µs timing reference. Once the *low_power_clk* is used for base time reference, *master1_gclk* and *master2_gclk* clock can be gated. The Radio Module and the Oscillator can be powered down, driving low *osc_en* and *radio_en* outputs, according to their respective DEEPSLCNTL bit fields. The radio and oscillator enable signals are conveyed only if the corresponding DEEPSLCNTL bits are set. The DEEPSLWKUP register sets the time to stay in Deep Sleep before wake-up in number of *low_power_clk* clock cycles.

Figure 2-5 shows active mode to low power mode switch timing diagram.

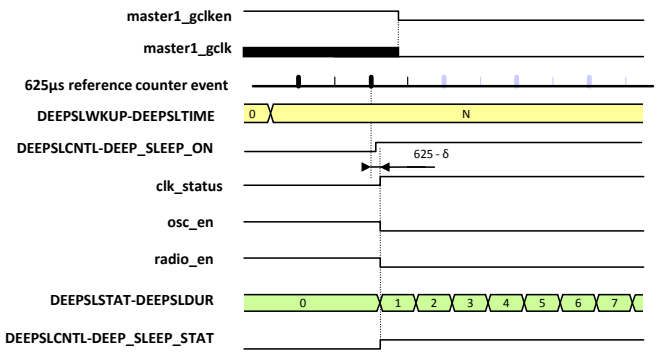

Figure 2-5 – Active to Low Power mode timing diagram

On DEEPSLPCNT-DEEP_SLEEP_ON rising edge, on *low_power_clk* first rising edge, *clk_status* goes high, when *osc_en* and *radio_en* go low (depending on their respective DEEPSLPCNTL register bits). When *clk_status* goes high, Fine Counter value is blocked at δ value, and *master1_gclk* clock can be gated on next *low_power_clk* falling edge. Fine Counter resumes later at δ (on wake-up) in order maintaining the correct 625µs reference timing.

2.5.3.2 Switch from Deep Sleep Mode to Active Mode

There are two possibilities for RW-BLE Core to terminate a Deep-Sleep phase:

1. Termination at the end of a predetermined time
2. Termination at external wake-up request, or software wake-up request

Figure 2-6 shows a typical deep sleep phase that is terminated at its predetermined time. In deep sleep mode RW-BLE Core is waiting for the Deep Sleep counter to equal DEEPSLWKUP–DEEPSLTIME. After a configurable time before wake up time (i.e. using ENBPRESET register fields), the RW-BLE Core powers up the radio module and the oscillator, according to their respective DEEPSLCNTL bit fields. Once this value is reached, it resets the *clk_status* signal indicating that the high frequency clock is available and can be used again. In order to ensure a clean wake–up of the processor and the RW-BLE Core, the high frequency oscillator is powered up before this value is reached. A separate signal (also enabled before the end of the low power mode) is provided for radio disabling purposes. The anticipation time of both the oscillator control (T_{WOSC} and T_{WEXT}) and the radio control (T_{WRM}) is configurable via the ENBPRESET register with a granularity of one low power clock cycle.

At the end of the sleep phase the DEEPSLCNTL-DEEP_SLEEP_ON bit is reset, leading to the RW-BLE Core wake up, which switches back to the fast clock, and restore proper 625µs internal reference time. The output *clk_status* indicates that the master clock coming from the fast oscillator is available and stable.

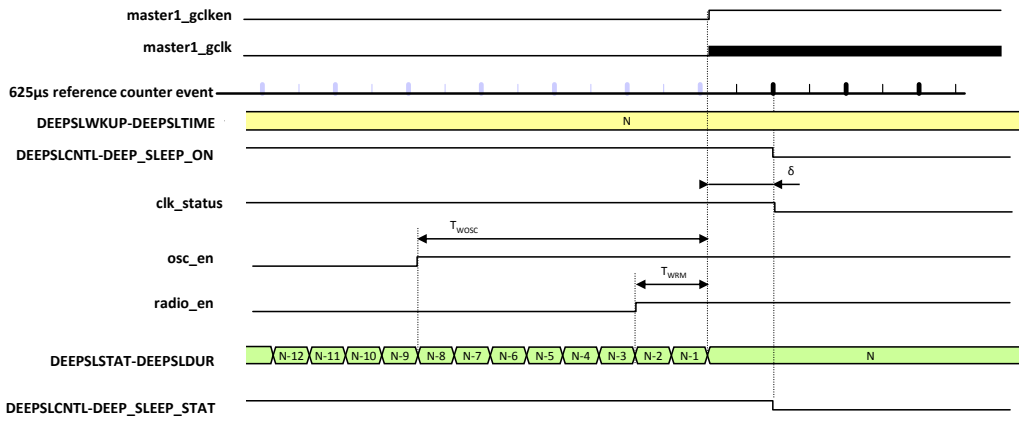

Figure 2-6 – Deep sleep mode – Standard termination

Figure 2-7 shows a wake up from a deep sleep period forced by the *wakeup_req* signal.

As the high frequency clock must be switched off sharply, it is internally gated and there are then no restrictions on the power down delay. The power up and stabilization delay of the oscillator must be less than T_{WOSC} in case the wake–up happened due to sleep timer expiry and T_{WEXT} in case when aborted by an external of software wake–up request. Distinction between the two cases allows optimizing the wake–up time if e.g. the oscillator is already running when woken up by external request. If a *wakeup_req* signal, or a software wake-up request, is asserted anytime

during the sleep phase, the RW-BLE Core jumps active mode. As long as the *wakeup_req* is high, entering the sleep mode is prohibited. Please note that external wake up request can be filtered when DEEPSLCNTL-EXTWKUPDSB is set.

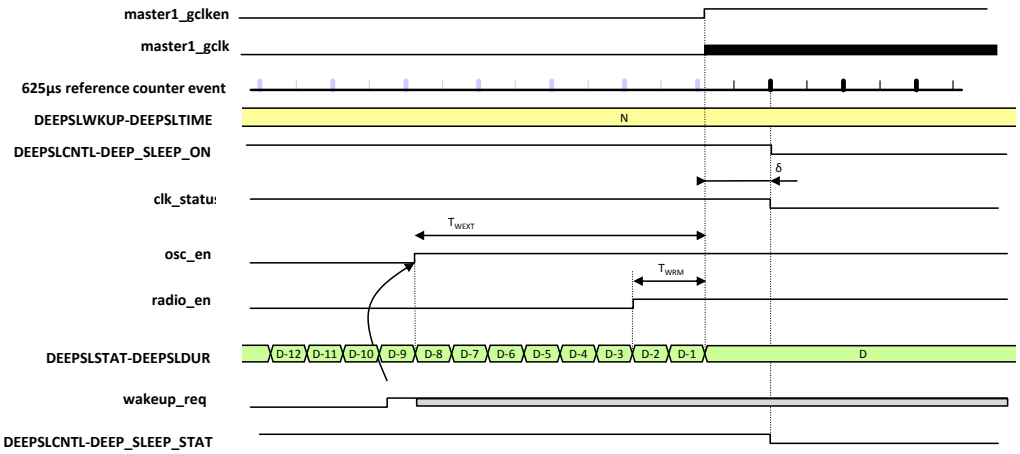

Figure 2-7 – Deep sleep mode – Aborted by wake up request

At the end of each low power time, the duration of low power time is reported into DEEPSLSTAT-DEEPSLDUR register. Then the RW-BLE Software must apply Base time Counter and Fine Counter correction in order to restore the 625µs base time reference alignment (See section 2.5.4 for details).

2.5.3.3 Sleep Modes Operations

Different functioning modes can be defined, as shown in Table 2-5.

In modes 2 to 4, the RW-BLE Core is internally running on the low power clock, but the rest of the system can still run with the high frequency clock. Depending on the selected configuration, the radio power down signal can be connected to the oscillator enable, or not. The low power strategy is defined by the end-product target, and the platform in which the RW-BLE Core is instantiated.

Mode	High frequency oscillator	Radio	RW-BLE Core	Processor	Comment
1	on	on	on	on	Normal active mode
2	on	on	off	–	Only RW-BLE Core in sleep mode
3	on	off	off	–	RW-BLE Core and radio in sleep mode
4	off	off	off	off	Deep Sleep mode

Table 2-5 – Modes of Operation

2.5.4 Frequency Adaptation for the Low Power Oscillator

The deep sleep mode supports either 32.0kHz or 32.768kHz oscillators. In both cases, the coherency of the 625µs reference timing event before and after the deep sleep period has to be guaranteed.

This correction mechanism can be achieved using BASTIMECNTCORR and FINECNTCORR registers. This mechanism is mandatory since Base Time Counter is running onto *master1_gclk* clock that is gated during Deep Sleep mode,

freezing both counters, and let the system run onto *low_power_clk* clock. The correction is achieved by updating of both the 10-bit Fine Counter value and the 27-bit Base Time Counter value, i.e. correcting respectively the fractional part and the integer part of the phase error of the 625µs reference timing event internal counter.

After termination of the deep sleep period, the 625µs reference timing event generator has to be corrected based on the actual time spent in the low power mode m_{Sleep} (@T_{osc}) reported in DEEPSLSTAT–DEEPSLDUR.

$$m_{Sleep} = T_{OSC} * DEEPSLDUR$$

The integer part of the 625µs reference timing, K, covers an entire Base Time Counter range (i.e. between 0 and 2^{27}-1) has to be written in the BASETIMECNTCORR field. K corresponds to the Base Time Counter correction value to apply after Deep Sleep duration considering a 32kHz clock has been used. So formula hereafter applies.

$$K = floor\left(\frac{m_{Sleep}}{625\mu s}\right)$$

The fractional part of the 625µs reference counter, R has to be written in FINECNTCORR–FINECNTCORR field. R covers an entire 625µs Fine Counter range (i.e. between 0 and 624). As 10-bit Fine Counter is a down counter, R formula hereafter applies.

$$R = 624 - \text{int}\left(m_{Sleep} - K \times 625\mu s\right)$$

Note the definition of the function *int(x)* used in the equation corresponds to the *integer* function that provides the integer part of a number.

In any case of Deep Sleep termination, the RW-BLE Software must compensate both BASTIMECNT and FINECNT counters using the FINECNTCORR and BASETIMECNTCORR fields. Those fields are calculated by the RW-BLE Software, and applied by the RW-BLE Core as described above:

1. On Deep Sleep termination, RW-BLE Software reads DEEPSLSTAT-DEEPSLDUR value
2. RW-BLE Software determines BASETIMECNTCORR and FINECNTCORR and stores it in registers
3. RW-BLE Software requires the correction to apply writing a 1 in DEEPSLCNTL-DEEP_SLEEP_CORR_EN

Base Time Counter and Fine Counter correction is applied when Fine Counter equals zero.

Base Time counter is corrected applying the following formula:

$$\text{BASTIMECNT}_{\text{New}} = \text{BASETIMECNT}_{\text{Current}} + 1 + \text{BASETIMECNTCORR}$$

Fine counter correction is done directly loading FINECNTCORR when Fine Counter equals zero.

Figure 2-8 shows the 625µs base time correction mechanism.

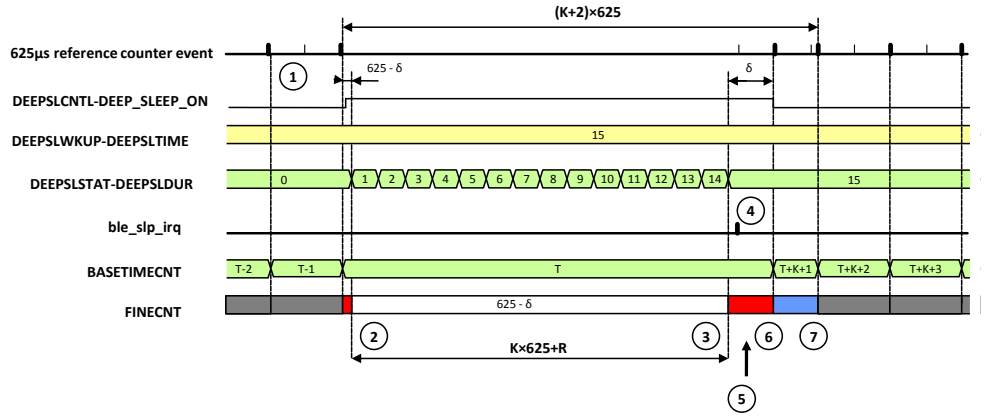

Figure 2-8 – 625µs Base Time Correction Mechanism

So the following events happen:

1. RW-BLE Software requires the system to go in Deep Sleep
2. RW-BLE Core goes to Deep Sleep mode and freezes Base Time Counter to T, and Fine Counter to 625-δ
3. After DEEPSLDUR *low_power_clock* cycles, or a wake up request, RW-BLE Core wakes-up, and Fine counter resumes
4. *ble_slp_irq* is generated once *master1_gclk* is enabled, and RW-BLE Software reads DEEPSLTIME and calculates K and R values
5. RW-BLE Software programs BASETIMECNTCORR=K and FINECNTCORR=R, and requires for compensation
6. Once fine counter equals zero, then Base Time Counter and Fine Counter are updated.
7. 625µs base time reference is corrected and realigned.

Please note that for safety reasons, pre-fetch mechanism must be disabled when DEEPSLCNTL-DEEP_SLEEP_ON equals 1. RW-BLE Software must take some margin in order to cope with this feature and avoid RW-BLE Core to miss some programmed Exchange Table entries.

Furthermore, in case of DEEPSLPWKUP-DEEPSLTIME is lower or equal to T_{WOSC} (or T_{WEXT}): *osc_en* output must be maintained high after DEESLCNTL-DEEP_SLEEP_ON request

Same rule applies in case of DEEPSLPWKUP-DEEPSLTIME is lower or equal to T_{WRM}: *radio_en* output must be maintained high after DEESLCNTL-DEEP_SLEEP_ON request.

Furthermore, if a Sleep Mode is required too early after Wake-up (i.e. before timer correction applies), and then RW-BLE Core cannot ensure 625µs base time reference accuracy.

2.6 Bus Interface

The bus interface converts the signals from the external processor bus to the internal format. Depending on *RW_BLE_XBAR_BUS* configuration parameter, it is either dedicated to the AHB bus of the ARM processor, or it is dedicated to the X-Bar bus Cortus APS3 Processor. Note that using X-Bar bus forces data bus width to 32-bits.

It is the only processor–dependent part of the RW-BLE Core.

2.6.1 AHB Bus Interface

Figure 2-9 shows the AHB (AMBA Advanced High–performance Bus, see [2]) timing diagram for details.

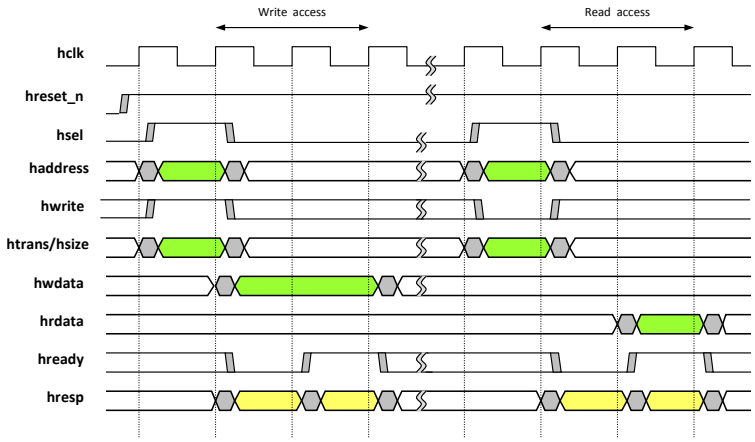

Figure 2-9 – AHB Access Timing Diagram

2.6.2 X-Bar Bus Interface

Figure 2-10 shows the X-Bar (see [3]) timing diagram for details.

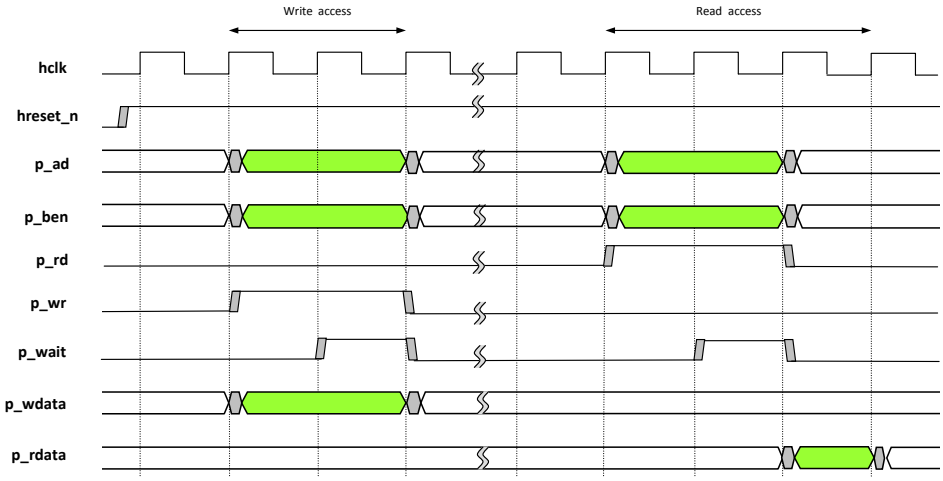

Figure 2-10 – X-Bar Access Timing Diagram

Note that *hclk* and *hreset_n* are reused in the X-Bar Interface block in order to ease synthesis script writing.

2.6.3 Exchange Memory vs. Register Access Selection

The bus interface selects either the Exchange Memory or the registers, depending on the MSB bit of the address of the access. Addresses ranges are configurable, together with Exchange Memory size. Table 2-6 below provides an example with *RW_BLE_ADDRESS_WIDTH* = 16:

Start address	End Address	Block
0x00000	0x0FFFF	RW-BLE Core registers (aligned on 32-bits words)
0x10000	0x1FFFF	Exchange Memory

Table 2-6 – Exchange Memory vs. Register Mapping table

In that case, only *haddress* or *p_ad* is wired to RW-BLE Core. A check on bit 16 (together with *hsel* and *hready* for complete decoding and synchronization in case of AHB, and together with *p_ben* and *p_wait* for X-Bar complete decoding and synchronization) determines whether an incoming access is routed to Exchange Memory (*haddress'* or *p_ad's* MSB equal to 1) or to the registers block (*haddress'* or *p_ad*'s MSB equal to 0).

The RW-BLE Core supports only 16–bit and 32–bit read/write access to register.

The RW-BLE Core supports 8–bit, 16–bit and 32–bit read/write access to Exchange Memory.

2.7 Registers

The registers are used to control the operation of the RW-BLE Core, and to give status indications to the software. They are 32–bit wide and are memory mapped on the core's address range (which also includes the Exchange Memory). This ensures that accesses to pointers or Bluetooth clock are non–breakable and that the register would not change or be used during the access.

Unused bits in the registers are not implemented. They are always read as 0 and any value written to them is ignored.

The registers are accessible by the processor in 16–bit or 32–bit accesses. When a 16–bit write is performed, the other bits of the registers are left untouched, regardless of the actual value of these bits on the input data bus.

2.8 Memory Controller

Several blocks need to access the Exchange Memory at different times and a Memory Controller is required to prioritize the accesses to the RAM. If *RW_BLE_CPU_HIGHEST_PRIORITY* is defined, then the processor accesses must have the highest priority. However, in order to prevent the processor from blocking the access to the Exchange Memory for a long time, this highest priority is granted only on its first access in case of burst access which can block the internal RW-BLE Core blocks to access the Exchange Memory. It is however possible to set the CPU to the lowest priority for any access type by not setting *RW_BLE_CPU_HIGHEST_PRIORITY* define parameter.

Figure 2-11 shows the internal blocks connected to the Memory Controller that need to access to the exchange memory.

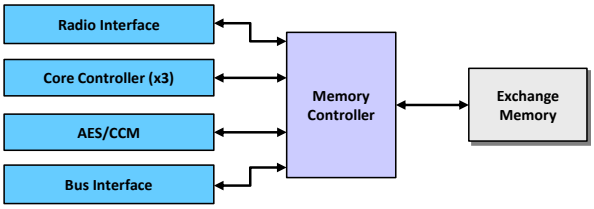

Figure 2-11 – Memory Controller's connections

The different ports of the memory controller are detailed in Table 2-7.

Port	Access size	Priority	Comment
Bus Interface	8–, 16– or 32–bit	Highest	Highest priority on first, or onto a single, access (when *RW_BLE_CPU_HIGHEST_PRIORITY* is set)
Radio Controller	8–bit		Radio Programming (channel, Tx Power, RSSI read, etc...)
Core Controller (x3)	8–, or 16–bit		Serializer / Deserializer, Bit Streaming, state machines, White List Search Engine and Resolving Address List Search Engine Note Event Scheduler and Resolving Address List Search engine have their own interface. Other blocks are time-sharing their respective accesses
AES-CCM	8–, or 16–bit		Encryption / Decryption
Bus Interface	8–, 16– or 32–bit	Lowest	Lowest priority on sub-sequent access (burst case)

Table 2-7 – Ports of the Memory Controller

Note that Core Controller has two interfaces. One is dedicated to the Event Scheduler, and the second one is shared in between Event Controller and Packet Controller.

Access sizes to information stored in the exchange memory (i.e. Exchange Table, Control Structure, Tx/Rx Descriptor, RAL Structure) may vary as described in Table 2-8, Table 2-9, Table 2-10, and Table 2-11 (see sections 3.2, 3.3, and 3.4 for further details on the different structure organization within the Exchange Memory).

ET field	Access size (bits)
$EXTAB_n$	2*16

Table 2-8 – Access sizes to parameters stored in Exchange Table

CS Field	Access size (bits)	CS Field	Access size (bits)
CXCNTL	8	RXMAXTIME	16
FRCNTL	8	PEER_RALPTR	16
FCNTOFFSET	16	SK0	16
LINKCNTL	8	SK1	16
LINKLBL	8	SK2	16
SYNCWL	16	SK3	16
SYNCWH	16	SK4	16
CRCINIT0	16	SK5	16
CRCINIT1	8	SK6	16
FLTPOL	8	SK7	16
RALCNTL	8	IV0	16
HOPCNTL	2*8	IV1	16
TXRXCNTL	2*8	IV2	16
RXWINCNTL	16	IV3	16
TXDESCPTR	16	TXWINOFFSET	16
WINOFFSET	16	TXCCMPKTCNT0	16
MINEVTIME	16	TXCCMPKTCNT1	16
ADV_BD_ADDR0	16	TXCCMPKTCNT2	8
ADV_BD_ADDR1	16	RXCCMPKTCNT3	16
ADV_BD_ADDR2	16	RXCCMPKTCNT4	16
ADV_BD_ADDR3	8	RXCCMPKTCNT5	8
MAXEVTIME	16	SLVFRXDLY	16
CONNINTERVAL	16	BTCNTRXSYNC0	16
CHMAP0	16	BTCNTRXSYNC1	16
CHMAP1	16	FCNTRXSYNC	16
CHMAP2	2*8	TXRXDESCCNT	2*8
RXMAXBUF	8	DMPRIOCNTL	2*8

Table 2-9 – Access sizes to parameters stored in Control Structure

TxDESC field	Access size (bits)
TXPTR	16
TXPHCE	2*8
TXPHADV	2*8
TXDATPTR	16

RxDESC field	Access size (bits)
RXPTR	16
RXSTATCE	2*8
RXPHCE	2*8
RXPHADV	2*8
RXCHASS	2*8
RXDATAPTR	16
RXRALPTR	16

Table 2-10 – Access sizes to parameters stored in Tx/Rx Descriptor

Note that each Descriptor's Data pointer has an associated memory space that is constrained up to 255 bytes (which is the longest possible length of a BLE v4.2 packet).

RAL field	Access size (bits)
RAL_INFO	16
RAL_PEER_IRK	8*16
RAL_PEER_RPA	3*16
RAL_PEER_ID	3*16
RAL_LOCAL_IRK	8*16
RAL_LOCAL_RPA	3*16

Table 2-11 – Access sizes to parameters stored in RAL Structure

2.9 Exchange Memory

2.9.1 Exchange Memory Content

The Exchange Memory is a synchronous, single cycle access (if no wait sates inserted) embedded memory containing an Exchange Table, Control Structures, Tx/Rx Descriptors and its associated Data Buffers, and RAL Structure.

The Exchange Table is used to indicate the addresses of the different Control Structures in the Exchange Memory. The Exchange Table pointer is provided through ET_CURRENTRXDESCPTR-ETPTR. The Control Structures contain all the parameters required to control the events and their timings, and link to the Tx Descriptors through CS-TXDESCPTR field. Rx Descriptor pointer is provided through ET_CURRENTRXDESCPTR-CURRENTRXDESCPTR. Tx/Rx Descriptors describe the packets exchanged during the event. A set of Tx/Rx Descriptors can describe either a finite chained list, or a ring buffer. The Tx/Rx descriptors contain the next Tx/Rx Descriptor pointer value, the status of the current Tx/Rx packet, and the Data Buffers used for data to be transmitted or received. Each Descriptor embeds a Data pointer that indicates the location of its associated Data Buffer.
Note that Tx/Rx Descriptors memory spaces can be doubled in order to use automatic toggling between different link layer connections. This relaxes RW-BLE Software timing constraints required in order to ensure Tx/Rx buffers are free before next use, and avoid any buffers overlap during use.

Table 2-12 describes the Exchange Memory content: data intends to cover advertising, scanning and link layer connection modes.

Field name	Size (in bytes)	Description
Exchange Table	64	Scheduling entry of the RW-BLE Core, read each 625µs.
Control Structure	80 / 82 [(1)]	A single Control Structure contains all control parameters of advertising, scanning, and connection events. Different control structures are needed to handle efficiently all the event types
Tx Descriptor	6	Define next Tx Descriptor pointer value, as well as transmitted packet header content (according CS-FORMAT) and gives status about transmission of this packet (done or pending)
Tx Data buffer	Up to 251	Sized exactly to the amount of data to be transmitted, as per the packet payload size.
Rx Descriptor	10	Define next Rx Descriptor pointer value, as well as received packet header content (according to CS-FORMAT) and gives status about reception of this buffer (done or free)
Rx Data buffer	Up to 251	Sized to the maximum possible amount of received data, as per the different packet payload size.
RAL Structure	52	Resolving Address List Structure used during RPA resolution/renewal Note there may be several structures embedded in the Exchange Memory

Table 2-12 – Exchange Memory Content

[(1)]: Control Structure size depends on the configuration: 76 bytes size stands for BLE only configuration, while 78 bytes size stands for Dual Mode configuration.

Table 2-13 give an example of the Exchange Memory mapping using *RW_BLE_ADDRESS_WIDTH* set to 16.

Address		Content	Size
Begin	**End**		
0x10000'H	0x1003F'H	Exchange Table	64
0x10040'H	0x1006F'H	Frequency Index Table for radio programming (see section 2.14.6)	40 x *SWORD*
0x10070'H	0x1FFFF'H	Available for Control Structures, Descriptors, Data Buffers, White List, Resolving Address List and Radio Initialization Buffer.	Variable

Table 2-13 – Exchange Memory mapping

Please note:

1. Exchange Table pointer is defined by ET_CURRENTRXDESCPTR-ETPTR register (i.e @0x0000 in the example above)
2. A specific radio control area at a variable location in the Exchange Memory may be required to program the radio. A segment of 32 bytes is reserved to this purpose in the Exchange Memory. See section 2.14.6 for a description of this mechanism. See also Radio User Manuals for SPI software access memory space definition.
3. *SWORD* is the size of the frequency word in bytes. For ranges shown here, *SWORD* = 1 byte, and tables bases are word aligned. Note *SWORD* size can be different and depends on the selected radio. Frequency Table pointer is set defined through RADIOCNTL3 (i.e @0x0040 in the example above)
4. For White List / Resolving Address List management and device filtering, please refer to section 3.7.2 for further details.

2.9.2 Interface Timing Diagrams

Figure 2-12 shows Exchange Memory timing diagram for write access (left) and read access (right), with both having one *hclk* clock cycle wait state inserted.

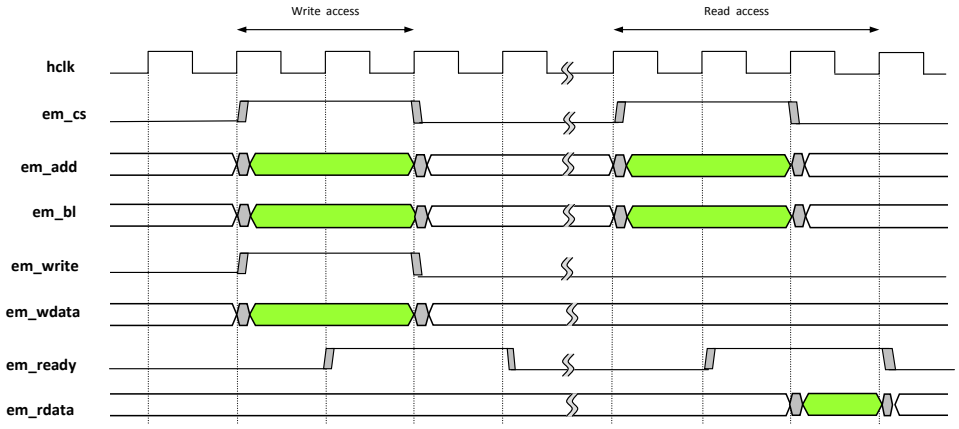

Figure 2-12 – Exchange Memory Interface Timing Diagram

2.9.3 Exchange Memory Access Profiles

2.9.3.1 General considerations

Exchange Memory accesses can be done in a 8-bit, 16-bit or 32-bit format depending on its data bus width (See Table 2-1) from the Bus interface block (i.e. the Processor), and in an 8-bit, or 16-bit format depending on which internal block is performing the access (AES-CCM, Event Scheduler, Event Controller or Packet Controller). The arbitration between all these blocks is controlled by the Exchange Memory Controller (refer to section 2.8 for further details).

The Event Scheduler performs Exchange Table read, and checks whether it is valid entry (See section 2.12.2 for further details). If the Exchange Table entry is valid, then the Event Scheduler enables the Event Controller that pre-fetches the Control Structure in order to handle correctly the advertising / scanning / initiating / connection events. The Event Controller determines which event type is required by the software, and reads the Tx/Rx Descriptor next

pointer value, status fields, header fields and associated Data pointer. Then it enables the Packet Controller in order to perform packet processing (bit-streaming management, Tx/Rx enable, Received packet, error reporting status, etc…).

Note that Data buffers can be accessed either by the Packet Controller, or the AES-CCM engine, depending on the connection context. (See section 2.11 for Bit Stream details).

At the end of each reception, Tx/Rx Descriptors status fields are updated by the Event Controller: Rx status fields provide the status of the received packet, and Tx status field determines whether a retransmission is required.

At the end of the event (i.e. no more packet to process), the Event Controller updates the necessary Control Structure fields required by the software for the next event, and unlocks the Event scheduler.

On top of these basics mechanism, the Event Scheduler continues to read Exchange Table entries in order to determine whether the current event in process must be stopped. This mechanism prevent from any overlapping issue in between interlaced events. This mechanism is active only when an event is in process.

In addition, during Advertising events, it is possible for the Event Controller to enable either White List Search engine or Resolving Address List engine in order to perform address resolution operations. In both case, the engines need to access the Exchange Memory (i.e either White List or RAL Structures)

Please refer to section 2.12 for Core Controller details.

Please refer to section 3 for HW/SW interface details.

Please refer to section 3.4.5 for data flow details.

2.9.3.2 Event Overlapping Prevention

The Core Controller is in charge of determining whether a pre-fetched Exchange Table entry can be processed. It is mandatory, during all events under process to perform the pre-fetched Exchange Table entry in order to prevent from Event overlapping.

Additionally, it is possible, for timing drift compensation reasons or master / slave connections interleaving, for two consecutive entries to be programmed with valid pointers. In this case, the RW-BLE Core ensures the new programmed event to start only if the system can process it: i.e. RW-BLE Core needs to be ready for processing a new event, and needs to have enough time to prepare the new event. Some particular care must be taken in this case.

Basically, prior to enable of the Packet Controller, the Event Controller must read all necessary Control Structure fields as well as launching the frequency hopping calculation, and ensures hop channel is ready. If it is not the case, then the event must be discarded.

Please refer to sections 3.2 and 3.5 for further details.

2.9.3.3 Exchange Table anticipated pre-fetch mechanism basics

The Event Scheduler reads the Exchange Table using the anticipated pre-fetch mechanism as shown in Figure 2-13 (Event with no delay) and Figure 2-14 (Delayed Event).

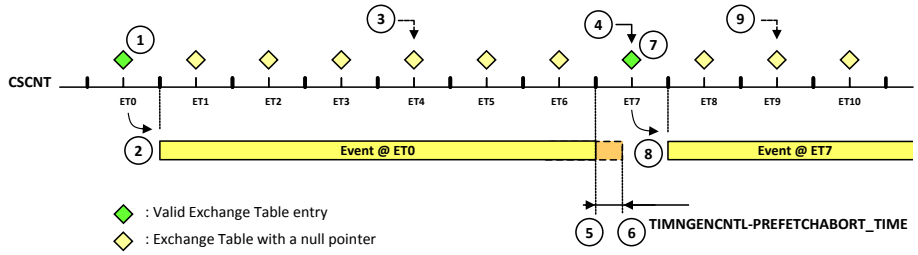

Figure 2-13 – Anticipated pre-fetch mechanism / Event with no delay

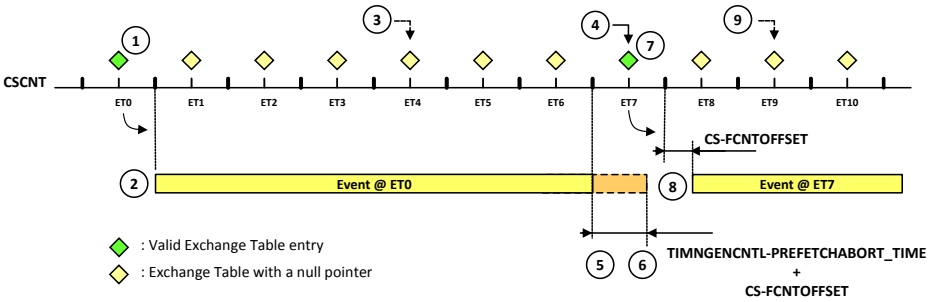

Figure 2-14 – Anticipated pre-fetch mechanism / Delayed event

The following mechanism applies:

1. The Event Scheduler performs Exchange Table pre-fetch, reading Exchange Table entries ET0
2. As ET0 is a valid entry, then the programmed event is processed
3. During the current event processing, the Event Scheduler continues to read exchange entries on each 625µs timing reference boundary.
4. At 625µs timing reference boundary in between ET6 and ET7, the Event Scheduler reads a new event is ready to be processed, CS-FCNTOFFSET is read to determine whether the event is delayed or not
5. The Event Scheduler stats EventAbortCount that counts up to (CS-FCNTOFFSET) + (TIMINGGENCNTL-PREFETCHABORT_TIME).
6. Once EventAbortCount elapses, if Event Controller and Packet Controller FSMs are in idle mode then normal operation resumes (i.e the system has already stopped properly and is ready for next event processing), else an immediate abort of the current event in process is required.
7. The Event Scheduler performs Exchange Table pre-fetch, reading Exchange Table entries ET7. (Note this can happen before step 6 depending on CS-FCNTOFFSET value)
8. As ET7 is a valid entry, Corresponding control Structure is read then the programmed event is processed.
9. During the current event processing, the Event Scheduler continues to read exchange entries on each 625µs timing reference boundary.

In case TIMGENCNTL-PREFETCH_TIME and TIMGENCNTL-PREFETCHABORT_TIME are not set properly, leading to the impossibility to start a programmed event because the current event is not finished on time, then an error is generated: either ERRORTYPESTAT-EVT_SCHDL_ENTRY_ERROR, ERRORTYPESTAT-EVT_SCHDL_APFM_ERROR or

ERRORTYPESTAT-EVT_CNTL_APFM_ERROR is set, depending on the location within the Core controller (issue happening in Event Scheduler or in Event Controller). Please refer to section 2.12 for counters implementation details.

The anticipated pre-fetch mechanism rules are summarized in Table 2-14.

ET_n	ET_{n+1}	Event Scheduler action
Valid entry	null	Process the event normally
Valid entry	non-null	Start the event and requires abort if necessary
null	Valid entry/null	No action

Table 2-14 – Anticipated pre-fetch mechanism rules

Note that TIMGENCNTL-PREFETCHABORT_TIME counter need to expire before TIMGENCNTL-PREFETCH_TIME instant occurs. Figure 2-15 illustrates prefetch abort and prefetch instant programming margins. It is mandatory to have prefetch abort time counter to elapse after ET prefetch instant for proper behavior of the RW-BLE Core.

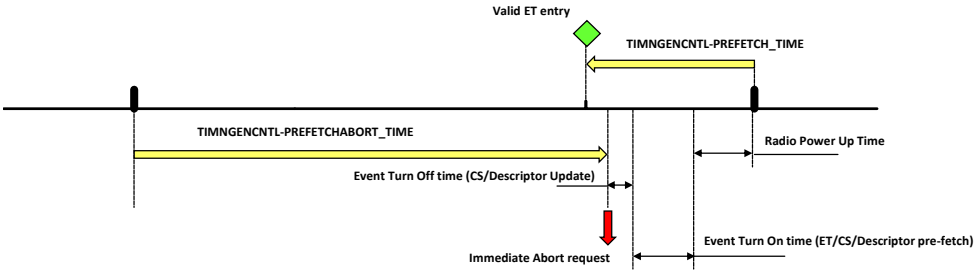

Figure 2-15 – Prefetch abort time and prefetch instant programming margin

2.9.3.4 Exchange Memory access for Broadcaster, Advertiser and Master Devices

This section describes the Exchange Memory access profiles in case the RW-BLE Core is required to transmit first. Note that depending on the programmed Event, several options are possible such as Tx once, or one Tx / Rx sequence, or performing several Tx / Rx sequences.

Figure 2-16 shows an example of the repartition in time of the Exchange Memory accesses during packet processing in Advertising mode, and Master connect mode for non-encrypted packets. In this case, AES-CCM is disabled and does not access to the Tx/Rx Data Buffers. Please note that pre-fetch time equals 312.5µs in this example. Note also that CS-FCNTOFFSET is considered as set to 0x0.

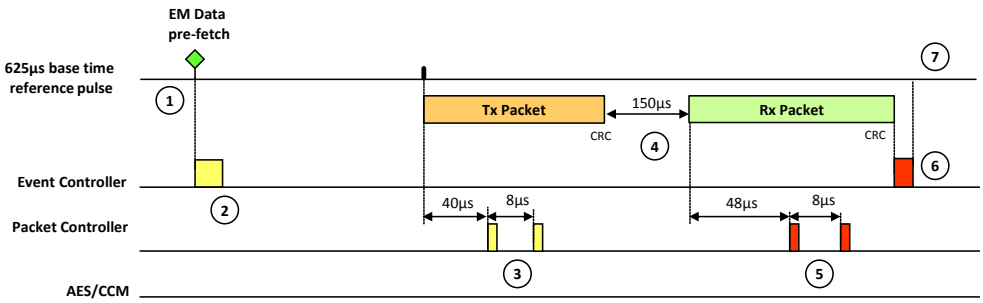

Figure 2-16 – Exchange Memory accesses for non-encrypted packets / Master or Advertiser

The following mechanism applies:

1. Event Scheduler performs Exchange Table pre-fetch, and enables Event Controller
2. Event Controller reads Control Structure content (i.e. common settings are read, like Tx power, channel index, synchronization word, Rx window size, etc… and encryption mode) and Tx/Rx Descriptors next pointer, Tx/Rx Data Pointer, status field, and header field. It enables Packet Controller in Tx mode.
3. Packet Controller is enabled in Tx mode and builds the packet to transmit. It reads Tx Data buffers.
4. Event Controller waits for 150µs before starting Rx processing. In parallel, it enables Packet Controller in Rx Mode, which performs Rx window opening.
5. Once Synchronization Word is detected, the packet reception starts, and each time a byte is received, it is written in the Rx Data buffers by the Packet Controller.
6. At the end of the reception, the Packet controller provides Rx status fields (i.e. Rx NESN, Rx SN, Rx MD, Rx packet type, Rx packet length, CRC error status, etc…) to the Event Controller.
7. Event Controller determines whether a new Tx/Rx frame needs to be sent according to Rx status bit. The Event Controller then updates Tx/Rx Descriptors status field. In case the event must be closed, Event Controller saves specific fields for next event (i.e. channel index, etc…).
8. If there are no more data to send, Event Controller un-locks the Event Scheduler. The event is closed.

Figure 2-17 shows an example of the repartition in time of the Exchange Memory accesses during packet processing in case in Master connect mode for encrypted packets. In this case AES-CCM is enabled and accesses the data buffers before or after the Packet Controller, according on Tx/Rx mode. Please note that pre-fetch time equals 312.5µs in this example. Note also that CS-FCNTOFFSET is considered as set to 0x0.

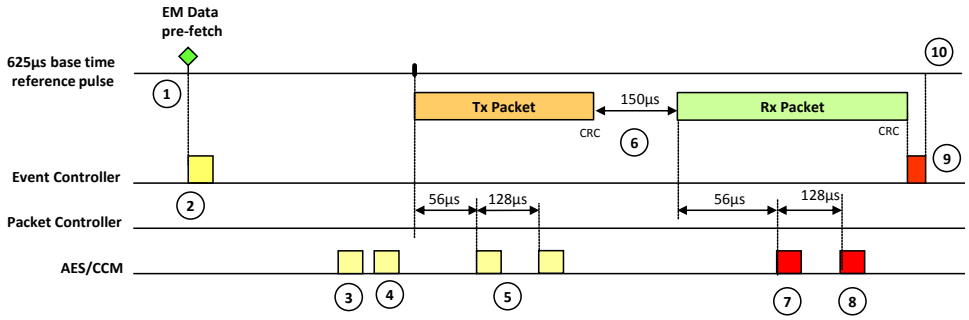

Figure 2-17 – Exchange Memory accesses for encrypted packets / Master

The following mechanism applies:

1. Event Scheduler performs Exchange Table pre-fetch, and enables Event Controller
2. Event Controller reads Control Structure content (i.e. common settings are read, like Tx power, channel index, synchronization word, Rx window size, etc… and encryption mode) and Tx/Rx Descriptors next pointer, Tx/Rx Data pointer, status field, and packet header field.
3. Packet Controller is enabled in Tx mode and requires AES-CCM to perform MIC calculation and Encryption onto the first 16 bytes block.

4. First 16 bytes block of encrypted data is provided to the Packet Controller for transmission, and second block encryption is started
5. Packet Controller is enabled in Tx mode and builds the packet to transmit. It uses encrypted Tx Data and Tx MIC provided by the AES-CCM. It requires the AES-CCM to start another 16 bytes block encryption each time an encrypted block has been sent. This is performed till the end of the packet construction.
6. Event Controller waits for 150µs before starting Rx processing. In parallel, it enables Packet Controller in Rx Mode, which performs Rx window opening.
7. Once Synchronization Word is detected, the packet reception starts. First 16 bytes block is received and is provided to the AES-CCM which performs decryption, and saves decrypted data in the Rx Data buffers.
8. Each time a 16 bytes block is received, it is provided to the AES-CCM for decryption. Decrypted data are written in the Rx Data buffers
9. At the end of the reception, the Packet controller provides Rx status fields (i.e. Rx NESN, Rx SN, Rx MD, Rx packet type, Rx packet length, CRC error status, etc…) to the Event Controller. Event Controller determines whether a new Tx/Rx frame needs to be sent according to Rx status bit. The Event Controller then updates Tx/Rx Descriptors status field. In case the event must be closed, Event Controller saves specific fields for next event (i.e. channel index, etc…).
10. Event Controller determines whether a new Tx/Rx frame needs to be sent according to Rx status bit. The Event Controller then updates Tx/Rx Descriptors status field. In case the event must be closed, Event Controller saves specific fields for next event (i.e. channel index, etc…). If there are no more data to send, Event Controller un-locks the Event Scheduler. The event is closed.

On each CS-RXTHR number of received packet, an Rx interrupt is generated by the Event Controller. It allows the software to read the corresponding Rx data buffers, to check the Tx Descriptors status field, to free the Tx Data Buffers, and to update next Tx data buffers (if needed). The software can then prepare the RW-BLE Core for low power mode till next event (if needed), and stores the necessary values required for next event (e.g., CCM Tx/Rx packet counters, channel index, NESN, SN, etc…). An End of Event interrupt is then generated when the Event Scheduler close the event.

In case of a master device or an advertiser device, the Packet Controller must use the default Rx window size value as defined by RWBLECNTL-RXWINSZDEF register: this information is provided by the Event Controller.

2.9.3.5 Exchange Memory access for Broadcast Scanner, Scanner, Initiator and Slave Devices

This section describes the Exchange Memory access profiles in case the RW-BLE Core is required to receive first. Note that depending on the programmed Event, several options are possible such as Rx once, or one Rx / Tx sequence, or performing several Rx / Tx sequences.

Figure 2-18 shows an example of the repartition in time of the Exchange Memory accesses during packet processing in Scanning mode, Initiating mode and Slave connect mode for non-encrypted packets. In this case, AES-CCM is disabled and does not access to the data buffers. Please note that pre-fetch time equals 312.5µs in this example. Note also that CS-FCNTOFFSET is considered as set to 0x0

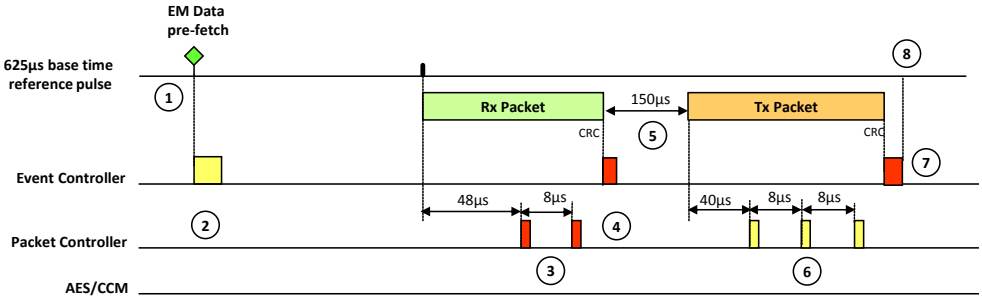

Figure 2-18 – Exchange Memory accesses for non-encrypted packets / Slave or Scanner or Initiator

The following mechanism applies:

1. Event Scheduler performs Exchange Table pre-fetch, and enables Event Controller
2. Event Controller reads Control Structure content (i.e. common settings are read, like Tx power, channel index, synchronization word, Rx window size, etc… and encryption mode) and Tx/Rx Descriptors next pointer, Tx/Rx Data pointer, and status field. It enables the Packet Controller in Rx mode which performs Rx window opening.
3. Once Synchronization Word is detected, the packet reception starts, and each time a byte is received, it is written in the Rx Data buffers by the Packet Controller.
4. At the end of the reception, the Event Controller updates the Rx Descriptor status fields and determines whether current Tx packet need to be retransmitted, or not (using then TxDESC-NEXTPTR pointer), and then reads the corresponding Tx Descriptor Header field. Event Controller updates Control Structure protocol fields and status.
5. Event Controller waits for 150µs before starting Tx processing. At the end of the 150µs inter-frame space, the Packet Controller is enabled in Tx mode.
6. Packet Controller builds the packet to transmit. It reads Tx Data buffers.
7. If there are no more data to send, or if no synchronization word is detected, the event is closed. Event Controller saves specific fields for next event (i.e. channel index, etc…).
8. Event Controller un-locks the Event Scheduler.

Figure 2-19 shows an example the repartition in time of the Exchange Memory accesses during packet processing in Slave connect mode for encrypted packets. In this case AES-CCM is enabled and accesses the data buffers before or after the Packet Controller. Please note that pre-fetch time equals 312.5µs in this example. Note also that CS-FCNTOFFSET is considered as set to 0x0.

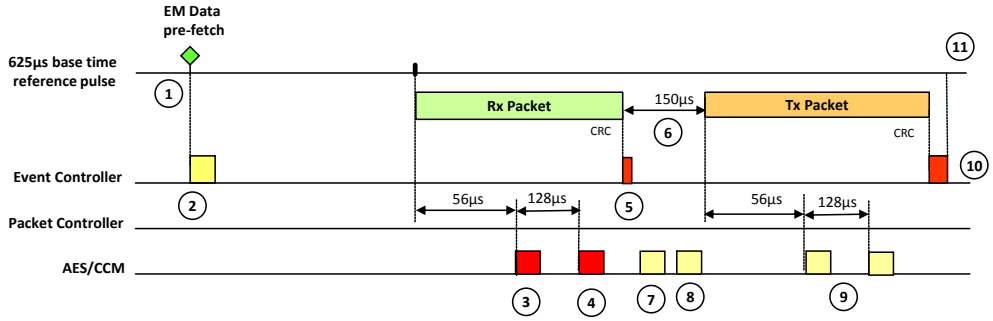

Figure 2-19 – Exchange Memory accesses for encrypted packets / Slave

The following mechanism applies:

1. Event Scheduler performs Exchange Table pre-fetch, and enables Event Controller
2. Event Controller reads Control Structure content (i.e. common settings are read, like Tx power, channel index, synchronization word, Rx window size, etc… and encryption mode) and Tx/Rx Descriptors next pointer, Tx/Rx Data pointer and status field. It enables Packet Controller in Rx mode which performs Rx window opening.
3. Once Synchronization Word is detected, the packet reception starts, and each time a 16 bytes block is received, it is provided to the AES-CCM for decryption.
4. Each time a 16 bytes block is received for decryption, it is written in the Rx Data buffers by the AES-CCM once decryption is performed
5. At the end of the reception, the Event Controller updates the Rx Descriptor status fields and determines whether current Tx packet need to be retransmitted, or not (using then TxDESC-NEXTPTR pointer), and then reads the corresponding Tx Descriptor Header field. Event Controller updates Control Structure protocol fields and status
6. Event Controller waits for 150µs before starting Tx processing. At the end of the 150µs inter-frame space, the Packet Controller is enabled in Tx mode.
7. AES-CCM reads first Tx Data Buffer 16 bytes block and start AES-CCM encryption and MIC calculation.
8. First 16 bytes block of encrypted data is provided to the Packet Controller for transmission, and second block encryption is started
9. Packet Controller is enabled in Tx mode and builds the packet to transmit. It uses encrypted Tx Data and Tx MIC provided by the AES-CCM. It requires the AES-CCM to start another 16 bytes block encryption each time an encrypted block has been sent. This is performed till the end of the packet construction.
10. If there are no more data to send, or if no synchronization word is detected, the event is closed. Event Controller saves specific fields for next event (i.e. channel index, etc…).
11. Event Controller un-locks the Event Scheduler.

On each CS-RXTHR number of received packet, an Rx interrupt is generated by the Event Controller. It allows the software to read the corresponding Rx data buffers, to check the Tx Descriptors status field, to free the Tx Data Buffers, and to update next Tx data buffers (if needed). The software can then prepare the RW-BLE Core for low power mode till next event (if needed), and stores the necessary values required for next event (e.g. CCM Tx/Rx packet counters, channel index, NESN, SN, etc…). An End of Event interrupt is then generated when the Event Scheduler close the event.

In case of a slave device or an initiator device, or a scanner device, the Packet Controller must first use CS-RXWINSZ as Rx window size, and then on the second packet reception, the default Rx window size value, as defined by RWBLECNTL-RXWINSZDEF register, must be used: this information is provided by the Event Controller. It allows a device to open a first wide synchronization window, and to ensure a first packet reception when the device is slave. Then the window size is reduced to a smaller and more optimal value as the device is considered as synchronized.

2.10 Interrupt generator

The interrupt generator creates the following interrupts, from the highest to the lowest priority:

- *ble_error_irq*: Error interrupt, generated when undesired behavior or bad programming occurs in the RW-BLE Core. Error status in reported in ERRORTYPESTAT register.
- *ble_cscnt_irq*: 625µs base time reference interrupt, available in active modes
- *ble_rx_irq*: Receipt interrupt at the end of each received packets, or when received packets number equals CS-RXTHR value.
- *ble_event_irq*: End of Advertising / Scanning / Connection events interrupt. Generated on normal termination, or on anticipated pre-fetch mechanism request.
- *ble_slp_irq*: End of Sleep mode interrupt (see section 2.5.3).
- *ble_crypt_irq*: Encryption engine interrupt, generated either when AES-128 ciphering/deciphering process is finished
- *ble_grosstgtim_irq*: Gross Target Timer interrupt generated when Gross Target timer expired. Timer resolution is 10ms.
- *ble_finetgtim_irq*: Fine Target Timer interrupt generated when Fine Target timer expired. Timer resolution is 625µs.
- *ble_sw_irq*: SW triggered interrupt: generated on SW request through register access (i.e RWBLECNTL-SWINT_REQ)

Each Interrupt can be masked. Each Interrupt supports software acknowledgement via the INTACK register. Each interrupt has its own output to the chip's interrupt controller. Interrupt can be defined either as edge or triggered level interrupt using *RW_BLE_INT_MODE_LEVEL* or *RW_BLE_INT_MODE_PULSE* parameter.

Exception is done for *ble_event_irq* that has two status/acknowledgment fields in register (see sections 3.10 and 3.12.1 for further details).

During event, 625µs base time reference interrupt can be automatically masked when INTCNTL-CSCNTDEVMSK is set, then only Receipt and End of Events interrupts apply.

Please refer to section 3.10 for further details.

2.11 Bit Stream Processing

2.11.1 General Packet Format

Bluetooth Low Energy packets have a common format, as described in Figure 2-20, and contain:

1. A 8-bit preamble

2. A 32-bit Access Address
3. A PDU composed of 1-Byte Header + a 1-Byte Length fields + N Payload data Bytes. A 4-bytes MIC may be appended to the payload in case of encrypted connection with non-null packet length.
4. A 3-byte CRC

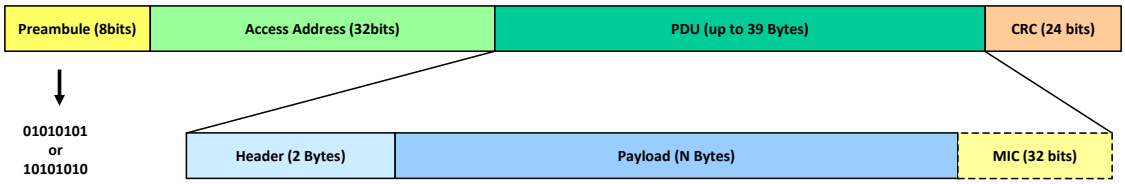

Figure 2-20 – Bluetooth Low Energy Packet / General Format

Depending on the context (test mode / advertising / scanning / initiating / link layer packets), Synchronization words and PDU's Header do not have the same contents.

Please refer to section 3.7 for Advertising Channel packets content details.

Please refer to section 3.8 for Data Channel packets content details.

2.11.2 Data Path Description

The packet processing is under control of the RW-BLE Core's Packet Controller that contains the state machine required to perform all packet–related activities. This controller also performs accesses to the Exchange Memory. Note that some of these accesses must be performed at the highest possible speed, in order to avoid blocking the processor bus for a long time. See section 2.8 for more details.

The bit stream processing is performed for packet transmission and reception:

- For transmission: collecting of different parts of the packets, encryption and encrypted MIC appending, serialization of the transmitted data, CRC calculation, and whitening computation.
- For reception: correlation of the synchronization word, window generation and synchronization detection, de–whitening, CRC checking, de-serialization, decryption and encrypted MIC checking, and storage into the Exchange Memory.

Initialization values of encryption / decryption are defined by the RW-BLE Software and are given in the control structure.

Initialization value of CRC is defined by the RW-BLE Software and is given in the control structure.

Initialization value of the Whitening is defined by the channel index, with LSB forces to 1 in anyway.

Figure 2-21 gives an overview of the bit stream processing in the RW-BLE Core.

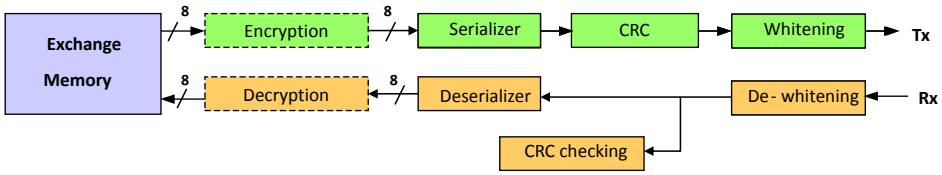

Figure 2-21 – Bit stream processing in the RW-BLE Core

Note: AES-CCM Encryption / decryption blocks are not directly part of the bit streaming path, as a parallel processing is required in order to process Tx/Rx packet encryption/decryption on-time.

Please refer to section 2.11.3 for further details.

Encryption applies on PDU payload data and MIC, when encryption mode is enabled, and when the packet has a non-null length. Encryption / decryption is performed by AES-CCM block

CRC applies on complete PDU.

Whitening applies on PDU and CRC.

2.11.3 Encryption and Decryption

2.11.3.1 Overview

In Bluetooth Low Energy Specifications (See [1]) AES-CCM method for encrypted Link Layer data channel is used. It is also used as a hash function for RPA resolution and generation by Resolving Address List engine. At Host level, AES-128 is used as a toolbox for key generation.

Hence RW-BLE Core supports three flavors:

1. AES-128 ciphering or deciphering performed on RW-BLE Software request: it uses register access to enable the AES-128 (allowing the RW-BLE Software Host to access AES-128 block) for authentication / security purpose.
2. AES-128 ciphering performed in real time under Resolving Address List engine control for RPA generation and resolution.
3. AES-CCM performed in real time for encrypted Link Layer data channel: it is launched by the Event Controller, and has higher priority regarding Software request on AES-128 use.

2.11.3.2 AES-128 basics

AES-128 ciphering / deciphering method is described in FIPS-197 (See [6]).

AES-128 uses a four 32-bit width words key (i.e. key length is 128 bits), a four 32-bit width ciphering block (i.e. named State array) and a defined number of ciphering round to perform that equals 10.

For both encryption and decryption, the AES-128 algorithm uses a key expansion procedure (that generates ten 128-bit cipher keys from the input key) and a round functions composed of four different byte-oriented transformations that are:

1. Byte substitution using a pre-defined table
2. Shifting rows of the State array by different offset

3. Mixing the data within each column of the State array
4. Adding a Round Key to the State array

Please refer to [6] for additional details about these transformations and their inverse.

2.11.3.3 AES-128 on software request

AES-128 can be used by the RW-BLE Software for authentication purpose. The mechanism is the following:

RW-BLE Software stores the block to cipher/decipher, at AESPTR[15:0] address in the exchange memory. The block size to cipher/decipher is always considered to equal 16 bytes. RW-BLE Software defines the input key setting its value in AESKEY<> registers.

Once the settings done, RW-BLE Software starts AES-128 use by setting the cipher/decipher mode in AESCNTL-AES_MODE, and writing a 1 in AESCNTL.AES_START.

On normal termination, the RW-BLE Software receives a *ble_crypt_irq*. When done the RW-BLE Software can find ciphered/deciphered data at AESPTR+16 address as shown in Figure 2-22 (considering byte address memory).

When Encrypted Link Layer event has to be processed, it is possible that Event controller requires AES-CCM to run at the same time a SW request is on-going. In this case the current ciphering/deciphering process is stopped as real time AES-CCM processing has higher priority. When real time encryption/decryption process ends, the AES-128 FSM restarts the ciphering/deciphering process till normal termination, and then AESCNTL.AES_START is reset.

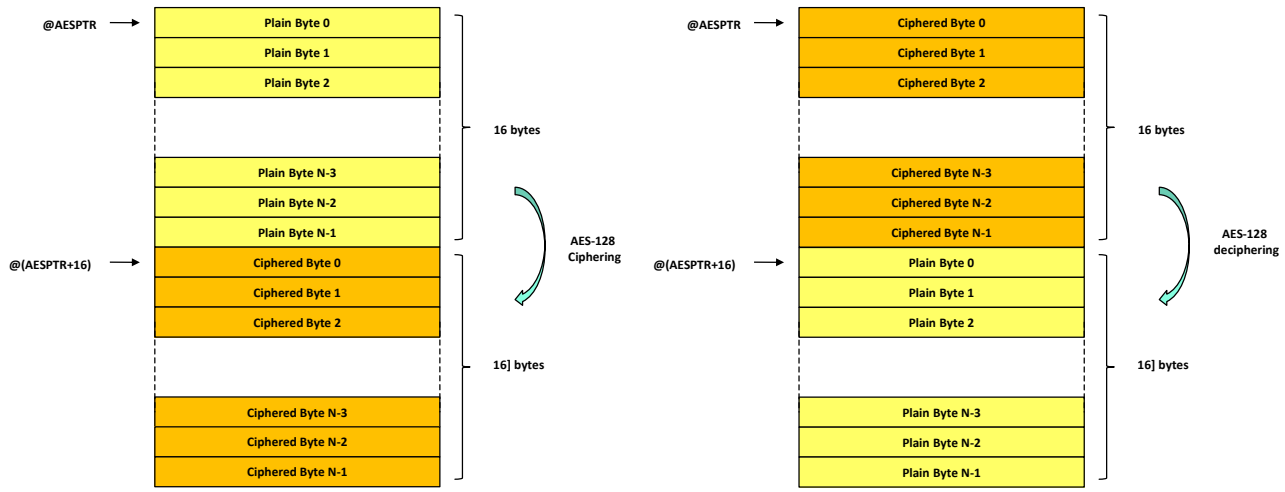

Figure 2-22 – Memory Mapping for AES-128 Software Use

2.11.3.4 AES-128 under Resolving Address List control

AES-128 can be used as a hash function by the Resolving Address List engine for RPA resolution, or generation. The mechanism is the following:

On Resolving Address List engine request, "*data*" and "*key*" are provided to the AES-128, and a start pulse kicks the ciphering process. Once hash operation is performed, the AES-128 returns the resulting "*cipher*" value as well as a done pulse. AES-128 performs the following calculation.

$$cipher = hash(data, key)$$

The mechanism has the highest priority over SW-driven AES-128 operation. As it is performed in real-time, and during non-connected mode, there is no risk of collision with AES-CCM encryption / decryption operations.

2.11.3.5 Real time AES-CCM

2.11.3.5.1 General Rules

Encryption and Decryption are performed by the CCM block. CCM algorithm is defined by the Bluetooth Low Energy Specification (See [1]). This block complies with the NIST 800-38C that defines the CCM algorithm (See [5]). The CCM algorithm is using AES-128 ciphering toolbox as described in FIPS-197 (See [6]).

AES-CCM block uses the CS-SK, CS-IV, CS-TXCCMPKTCNT, and CS-RXCCMPKTCNT, in order to generate or check MIC that is appended to the payload, and to perform payload Encryption / Decryption.

MIC is checked / generated from data channel PDU header LSB part and data channel PDU payload part, except that NESN, SN and MD bits of the header LSB part are masked to 0.

Encryption / Decryption are performed onto data channel PDU payload and MIC.

2.11.3.5.2 CCM Structure

CCM is composed of several AES blocks named as A and B-blocks.

B-blocks are used for MIC generation process / MIC check. A-blocks are used for encryption / decryption.

Figure 2-23 shows CCM B block structure.

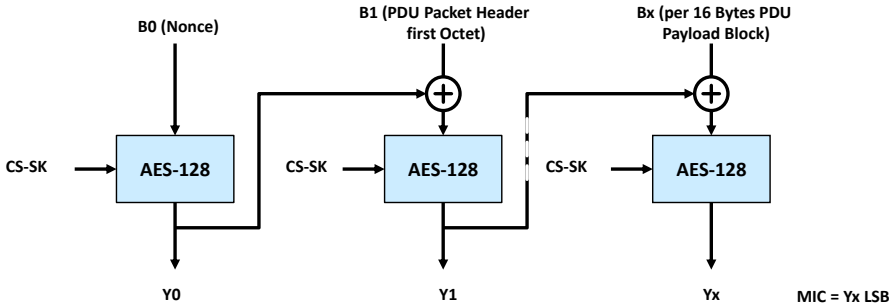

Figure 2-23 – CCM B-Block structure

The MIC equals to the 4 LSBs of Yx.

Figure 2-24 shows CCM A block structure.

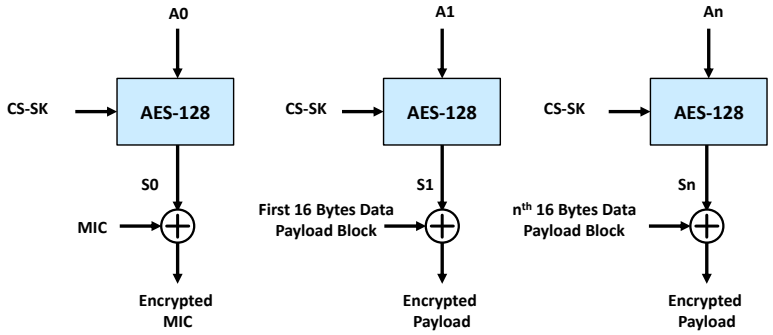

Figure 2-24 – CCM A-Blocks structure

Encrypted data are calculated through a XOR operation between 128-bit plain data and S0…Sn values

2.11.3.5.3 Encryption Mechanism

Encryption mechanism is performing MIC calculation, and payload/MIC encryption. The following steps are described in [1]:

1. MIC Calculation:
    a. Block B0 uses Nonce
    b. Block B1 uses payload header's first octet with SN, NESN and MD masked to 0
    c. Block Bx uses x^{th} 16 bytes block of payload data if needed (with 0 padding if needed)
    d. MIC is stored locally in the RW-BLE Core
2. Data Encryption:
    a. Block A1 encrypts the first 16 octets of the payload
    b. Block An encrypts the n^{th} 16 byte of the payload if needed (with 0 padding if needed).
    c. Block A0 encrypts MIC and saves it in the Exchange Memory
    d. Encrypted data are saved in the Exchange Memory

2.11.3.5.4 Decryption Mechanism

Decryption mechanism performs payload/MIC decryption, and MIC checking. The following steps are described in [1]:

1. Data Decryption:
    a. Block An decrypts the n^{th} 16 byte of the payload if needed (with 0 padding if needed).
    b. Block A1 decrypts the first 16 octets of the payload.
    c. Block A0 decrypts MIC and stored it locally in the RW-BLE Core
    d. Decrypted data are stored in the Exchange Memory
2. MIC Checking

a. Block B0 uses Nonce
b. Block B1 uses payload header's first octet with SN, NESN and MD masked to 0
c. Block Bx uses x^{th} 16 bytes block of payload data if needed (with 0 padding if needed)
d. MIC value is compared with decrypted MIC

2.11.3.5.5 CCM Nonce

CCM Nonce ensures each encrypted packets is unique. CCM Nonce format is described in Table 2-15.

Position	Field Name	Comment
103:40	CS-IV[63:0]	Initialization vector
39	Direction	Direction Bit
38:0	CS-<TX/RX>CCMPKTCNT[38:0]	Tx/Rx CCM Packet Counter

Table 2-15 – CCM Nonce Format

CS-IV is the initialization vectors.

The Direction bit is defined as equal to 1 for master transmission, and as equal to 0 for slave transmission.

CS-<TX/RX>CCMPKTCNT are reset to 0 on the first encrypted packet exchange (initialized by SW).

CS-<TX/RX>CCMPKTCNT are incremented for all reception of an encrypted packet that is authenticated, and that causes a SN update (performed by HW).

CCM Nonce is used by A0 and B0 blocks only.

2.11.3.5.6 B-Block data format

Table 2-16 shows block B0 data format.

Position	Field Name / Value	Comment
127:120	Length	Payload Length LSB
119:112	0x00	As per Bluetooth specification (See [1])
111:8	CCM Nonce	CCM Nonce as defined in Table 2-15
7:0	0x49	Flags as per the CCM specification (See [5])

Table 2-16 – Block B0 format

Table 2-17 shows block B1 data format.

Position	Field Name / Value	Comment
127:24	Padding	Used to pad the block with 0
23:16	Payload header's first octet	NESN, SN and MD must be masked to 0
15:8	0x01	As per Bluetooth specification (See [1])
7:0	0x00	As per Bluetooth specification (See [1])

Table 2-17 – Block B1 format

Table 2-18 shows block Bx data format. Please note that in case of a payload that is bigger or equal than 128 bits, then padding part is removed from the Bx block format, and only payload data are present in the corresponding Bx block.

Position	Field Name / Value	Comment
127:k	Padding	Used to pad the block with 0
k-1:0	Payload data block to encrypt	Payload data

Table 2-18 – Block Bx format

2.11.3.5.7 A-Block data format

Table 2-19 shows block Ax data format.

Position	Field Name / Value	Comment
127: 0	LSB of counter i	Counter i counts A block
119:112	MSB of counter i	Counter i counts A block
111:8	CCM Nonce	CCM Nonce as defined in Table 2-15
7:0	0x01	Flags as per CCM specification

Table 2-19 – Block Ax format

A0 is always used to encrypt the MIC.

A1 is always used to encrypt the first 16 octet of the payload data.

Ax is used to encrypt the remaining octets of the payload data if needed.

A-Blocks payload data input is padded with 0 to match the 128-bit data width on need basis.

2.11.4 Serialization and De-serialization

Accesses to the Exchange Memory data buffers performed by either the Packet Controller or the AES-CCM are always done in an 8-bit format. Serializer and deserializer are controlled by the Packet Controller. The Serializer either reads a byte into the Exchange Memory, or gets it from the AES-CCM and sends it LSB first in the bit stream processing chain. The Deserializer gathers 8 bits coming from the bit stream processing chain and provides it to either the Exchange Memory or the AES-CCM.

Figure 2-25 shows the ordering of bits and the fetching processes.

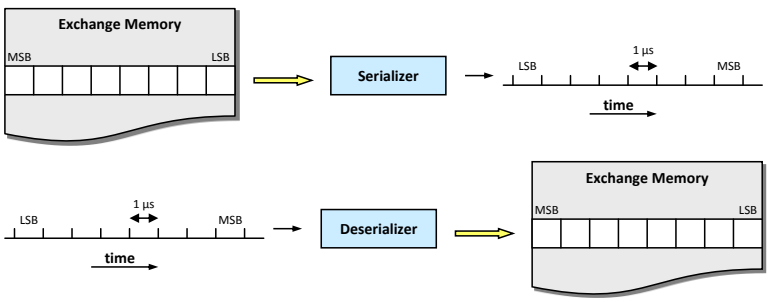

Figure 2-25 – Serialization and deserialization

2.11.5 CRC

CRC is defined by the following polynomial:

$$p(x)=1+x+x^3+x^4+x^6+x^9+x^{10}+x^{24}$$

CRC is a 24 bits field calculated over the PDU field for all packet types. If the packet is encrypted, then the CRC must be calculated after encryption over encrypted PDU and MIC. CRC is transmitted MSB first. In reception, CRC check is performed, and the status is reported in the control structure. CRC initialization value is defined by CS-CRCINIT.

2.11.6 Whitening

Whitening is used to avoid long sequence of 0 and 1. Whitening polynomial is defined as:

$$p(x)=1+x^4+x^7$$

Whitening is performed over PDU and CRC fields. Whitening and De-whitening are performed using the same polynomial. Whitening initialization value depends on the channel index for each advertising / connection event. It is defined as:

Whitening_Init_Value[6:0]={Channel_Index[0:5],'1'}

2.11.7 Bit stream timing

Figure 2-26 shows the timing of the transmission path.

The minimum delay is 1 clock cycle from serialization to the radio.

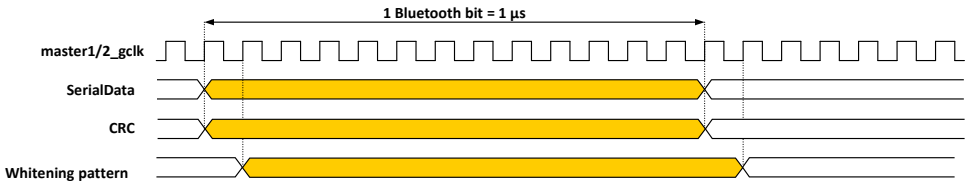

Figure 2-26 – Timing of the Tx bit stream / Example with *master1/2_gclk* = 13MHz

Figure 2-27 describes the timing of the reception.

RxData enters the bit stream processing after having been resynchronized with the clock extracted from the bit stream.

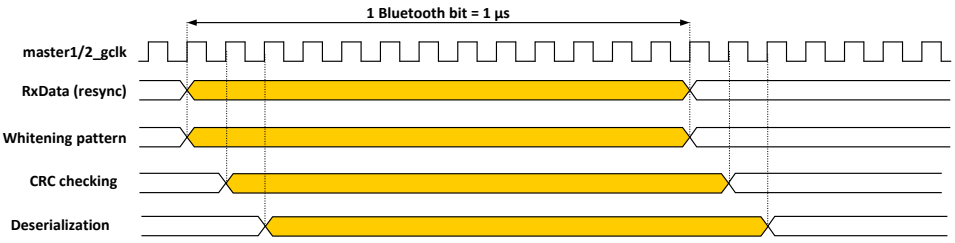

Figure 2-27 – Timing of the Rx bit stream / Example with *master1/2_g*clk = 13MHz

In reception, the timing for the bit stream processing comes from the external device reference clock (i.e. *ExternalRefClk*). As this clock can have different origins depending on the implementation of the radio chip (clock recovery included or not), it is necessary to be able to adapt this clock to always get a similar enable signal for the bit stream processing. Figure 2-28 shows how the external reference clock is used to generate an enable used to clock data in the bit stream processing. This enable is not used to gate a clock but to enable data shifting in. This enable is propagated beside the data all along the data path, and it is generated once the synchronization word has been detected by the correlator block (Refer to section 2.14.4).

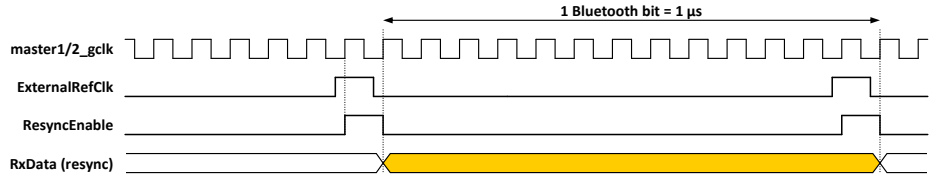

Figure 2-28 – Clocking of the bit stream processing in Rx / Example with *master1/2_g*clk = 13MHz

2.11.8 Bad reception

Upon bad reception of a packet, the packet processing is interrupted as fast as possible and the radio is switched off to save power.

Figure 2-29 shows the conditions that could terminate the packet processing prematurely.

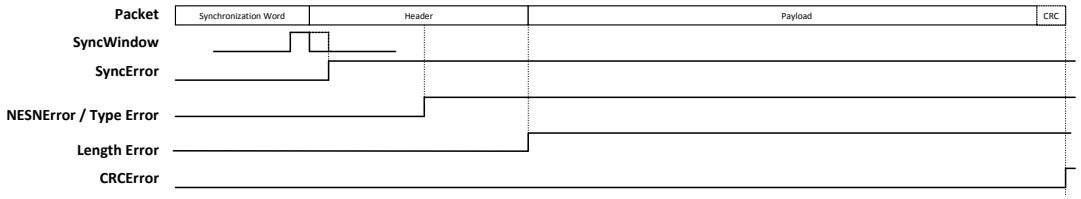

Figure 2-29 – Stopping the reception processing

Note that on top of this mechanism, any MIC error detection leads to device disconnection as defined in [1]. If on reception the length is lower than 5 bytes when the link is encrypted, then a length error is generated.

Note also that NESN error is reported only for LL data and LL control packets: i.e. NESN is not used by advertising, scan request, scan response and connection request packets. NESN error does not affect the processing, the error is indicated in the Rx Descriptor but the packet is still stored into the Exchange Memory. NESN error indicates the system whether a re–transmission of the previous Tx packet is required, and it is used to perform TxDESC-TXDONE

update after each reception. As well, Type Error is reported only when using advertising channels: this bit is used for pre-filtering when receiving packets. It stops the receipt in order to save power.

Please refer to Table 3-5 and Table 3-6 for further details.

Reception is stopped as well if it overflows the maximum length of the indicated TYPE in the packet header. Table 2-20 summarizes Header and Payload byte length to consider for length error testing.

Packet Type	Maximum payload length (in bytes)		
	Header (Bytes)	Payload (Bytes)	Total
ADV_IND	2	37	42
ADV_DIRECT_IND	2	12	17
ADV_NOCONN_IND	2	37	42
ADV_SCAN_IND	2	37	42
SCAN_REQ	2	12	17
SCAN_RESP	2	37	42
CONNECT_REQ	2	34	39
LL_DATA (not encrypted)	2	251	32
LL_CONTROL (not encrypted)	2	251	32
LL_DATA (encrypted)	2	255	36
LL_CONTROL (encrypted)	2	255	36

Table 2-20 – Maximum Payload length

2.12 Core Controller

The RW-BLE Core's Core Controller block contains various counters and state machines used to sequence the system. It is also in charge of the generation of advertising, scanning, initiating and connection events, and it controls the bit stream processing in transmit mode. It also generates the necessary Rx synchronization window in receipt that allows performing synchronization word correlation, and received packet detection.

In reception mode, the RW-BLE Core's Core Controller controls the bit stream processing to align processing with the detected synchronization word. A clock is extracted from the received data stream and used to clock in the data from the radio.

The Core Controller is composed of four main blocks:

- The Event Scheduler
- The Event Controller
- The Packet Controller
- The White List Search engine
- The Resolving Address List engine

The Event Scheduler is in charge of Exchange Table pre-fetch in order to determine whether an event must be started. Additionally, during event processing, the Event Scheduler performs Exchange Table anticipated pre-fetch in order to determine whether the current event must be stopped.

The Event Controller ensures the read/write access to the Control Structure and the Tx/Rx Descriptors pointer, and header field in order to feed the Packet Controller with relevant protocol data. The Event Controller also updates the Descriptor status fields. In case of encrypted link, the Event Controller is also in charge to setup the AES-CCM engine.

The Packet Controller is in charge of the bit stream processing controls, the packet built, packet recovery, and accesses the Tx/Rx data buffers when AES-CCM is not used. In case AES-CCM is used, then it accesses the Tx/Rx buffer in place of the Packet Controller. It is also in charge of the Tx/Rx Enable/Disable control of the radio, as well as real time control of the Bit Streaming (i.e CRC/Whitening).

The White List Search engine is in charge of device parsing within the White List. On Event Controller request White List engine looks for a known device in the available device list.

The Resolving Address List engine is in charge of Private Address resolution and/or generation. On Packet Controller request, the Resolving Address List engine process device Private Address resolution, and can generate a new Resolvable Private Address using AES-128 as a hash function.

2.12.1 Counters

There are several counters used to control the bit stream processing. These counters are described in Table 2-21 below. Note that all the counters do not stand into the Core Controller, but are mentioned here below for better understanding.

Counter Name	Location	Maximum Range	Comments
MicroSecCount	Timing Generator	[0:*clk_sel*-1]	Counts the number of clock cycles per microsecond (down-counter)
FineCount	Timing Generator	[0:624]	Counts the number of µs within a 625µs slot reference (down-counter)
BaseTimeCount	Timing Generator	[0:2^{27}-1]	Generates the 625µs base time reference used to determine which Exchange Table is currently used
PreStartCount	Event Scheduler	[0:PREFETCH_TIME-Tx/RxPowerUp-1]	Determines *FCOCount* counter start event.
FCOCount	Event Scheduler	[0:623]	Counts for the CS-FCNTOFFSET delay before radio power up instant, and *StartCount* is enabled. It allows Events to be delayed by up to 624µs.
StartCount	Packet Controller	[0:Tx/RxPowerUp-1]	Counts the power up sequence of the radio
PayloadCount	Packet Generator	[0:37]	Counts down the number of bytes in the payload. It is initialized with the value coming from the Control Structure or received in the payload header
FieldCount	Packet Generator	[0:31]	Counts the bits in each field of the packet. Its end of count is used to clock *PayloadCount*
IFSCount	Event Controller	[0:149]	Counts the Inter Frame Space (IFS) timing that must be equivalent to 150µs
AdvIntCount	Event Controller	[0:1499]	Counts the time interval in between advertising packets. Must be less than or equal 1.5ms
EventAbortCount	Event Scheduler	[0:PREFETCHABORT_TIME-1]	Used for anticipated pre-fetch mechanism. Determines the instant at which *NextFCOCount* starts.
NextFCOCount	Event Scheduler	[0:CS-FCNTOFFSET-1]	Used for anticipated pre-fetch mechanism. When elapses, defines the instant at which an event immediate abort is required

Table 2-21 – RW-BLE Core principal Counters

Please note that in case Dual Mode is supported, the Event Controller and the Packet Controller can be required to stop their respective current processing according to the current priority level (i.e. CS-CURRENTPRIO[4:0]) that is compared to external Bluetooth BR/EDR controller priority level (Please refer to section 3.6 for Dual Mode priority management details).

2.12.2 Event Scheduler

Event Scheduler reads Exchange Table each 625µs, when pre-fetch time occurs (defined by TIMGENCNTL-PREFETCH_TIME). If Exchange Table content is a valid entry (i.e ET-STATUS="00", ET-MODE=0x2, and ET-CSPTR≠0x0000), then the Event Controller is enabled. The Event Scheduler updates the current Exchange Table entry status field in order to indicate it has been read (i.e ET-STATUS = "01"). The Event Scheduler then waits for the end of the event. Once the end of the event is indicated, the Event Controller is disabled, and generates an end of event interrupt. Then the Exchange Table pre-fetch mechanism is resumed.

Event Scheduler Main FSM is detailed in Figure 2-30.

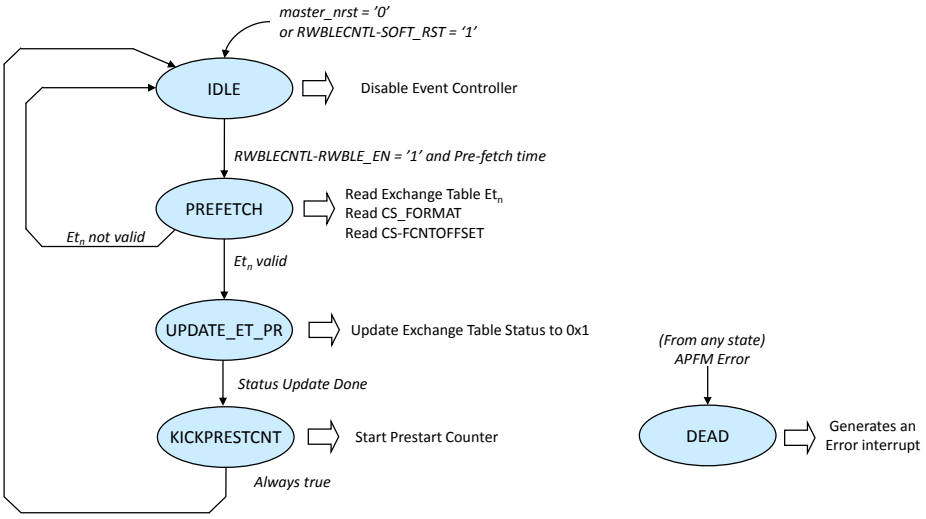

Figure 2-30 – Event Scheduler Main FSM

Furthermore, during the current event in process, the Event Scheduler performs anticipated pre-fetch mechanism each 625µs reference timing boundary. If a valid entry is found, then the Event Scheduler requires the Event Controller to stop its current activity in order to prepare for next programmed event, preventing from any event overlapping. In case the Event Controller cannot be stopped in time, and error interruption is generated and either ERRORTYPESTAT-EVT_SCHDL_ENTRY_ERROR or ERRORTYPESTAT-EVT_SCHDL_APFM_ERROR field is set.

Event Scheduler Anticipated pre-fetch FSM is detailed in Figure 2-31.

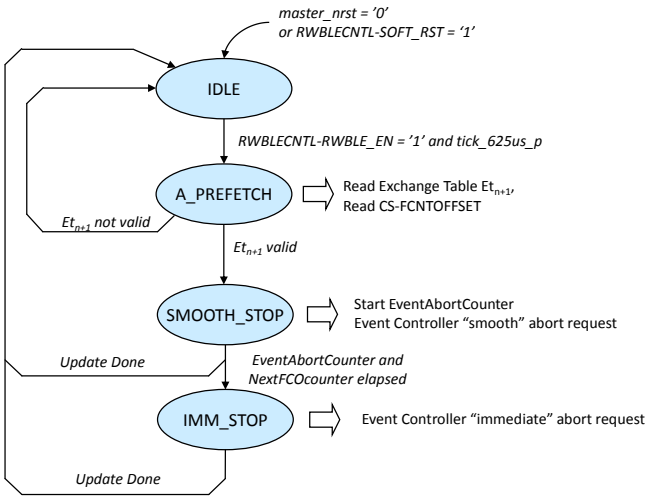

Figure 2-31 – Event Scheduler Anticipated pre-fetch FSM

For a detailed description of the Exchange Table content, please refer to section 3.2

During SMOOTH_STOP state, the Event Controller is required to stop the current event during IFS transition, after a proper sequence (e.g Tx then Rx in master, Rx then Tx in slave, etc...).

On IMM_STOP state, the Event Controller is required to stop immediately the current event.

2.12.3 Event Controller

When the Event Controller is enabled, it reads the Control Structure content, and Tx/Rx Descriptors pointer and status field, Tx Descriptor's header field. Once this is read, it enables the Packet Controller according to CS-FORMAT value. Each time a packet is received, Event Controller updates the Tx/Rx Descriptor status field. Each time CS-RXTHR number of Rx packet has been performed, the Event Controller generates an Rx interrupt. The Event Controller is also in charge of driving the AES-CCM encryption/decryption when encrypted link is required by the system during connection events. Once event processing is finished, the Event Controller indicates it to the Event Scheduler, and waits for next enable.

Please note that in case Dual Mode is supported, the Event Controller may be required to stop its current processing according to current priority level (i.e. CS-CURRENTPRIO[4:0]) compared to external Bluetooth BR/EDR controller (Please refer to section 3.6 for Dual Mode priority handling management details).

In case the event is discarded before it starts, it is mandatory to perform frequency hopping calculation in order preserve channel alignment.

In case the Event Controller is not able to finish Control Structure pre-fetch before the Tx/Rx Power u instant, then an error interruption is generated and ERRORTYPESTAT-EVT_CNTL_APFM_ERROR field is set.

Event Controller is composed of a main FSM that drives five sub-FSMs that are enabled according to CS-FORMAT values. Once enabled, each sub-FSM controls the Packet Controller. The sub-FSMs are dedicated to:

- Master Connection
- Slave Connection
- Advertising (Low Duty Cycle and High Duty Cycle)

- Scanning / Initiating
- RF Test Modes

Event Controller main FSM is detailed in Figure 2-32.

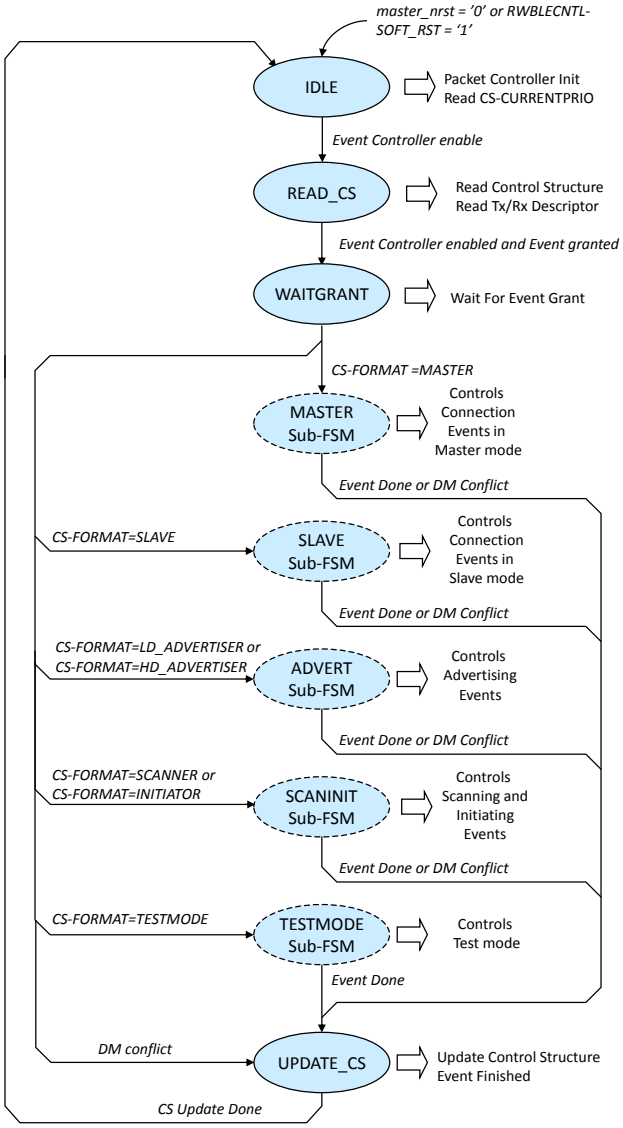

Figure 2-32 – Event Controller Main FSM

Event Controller master connect sub-FSM is detailed in Figure 2-33.

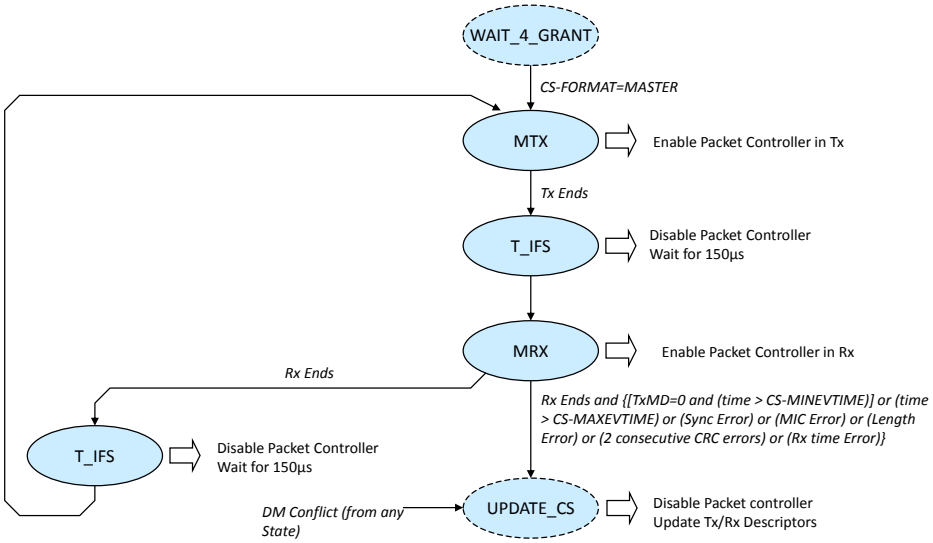

Figure 2-33 – Event Controller "Master Connect" sub-FSM

Event Controller slave connect sub-FSM is detailed in Figure 2-34.

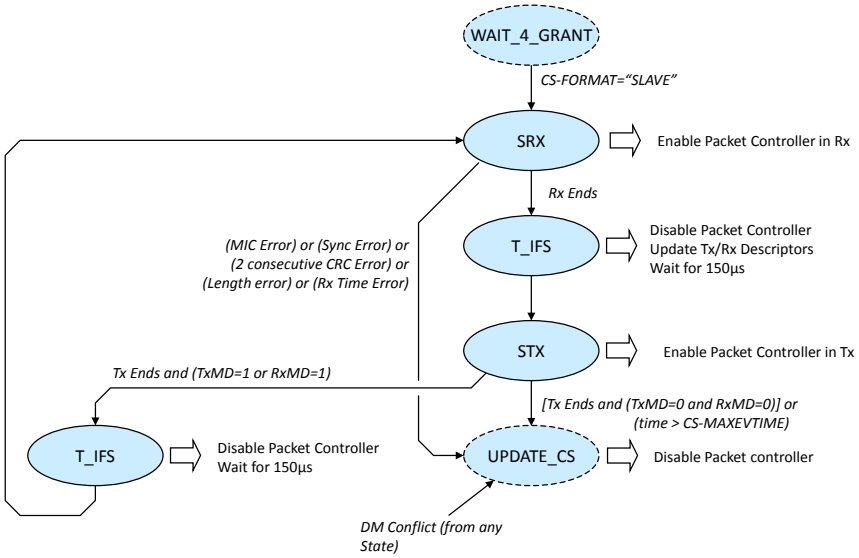

Figure 2-34 – Event Controller "Slave Connect" sub-FSM

Event Controller advertiser FSM is detailed in Figure 2-35.

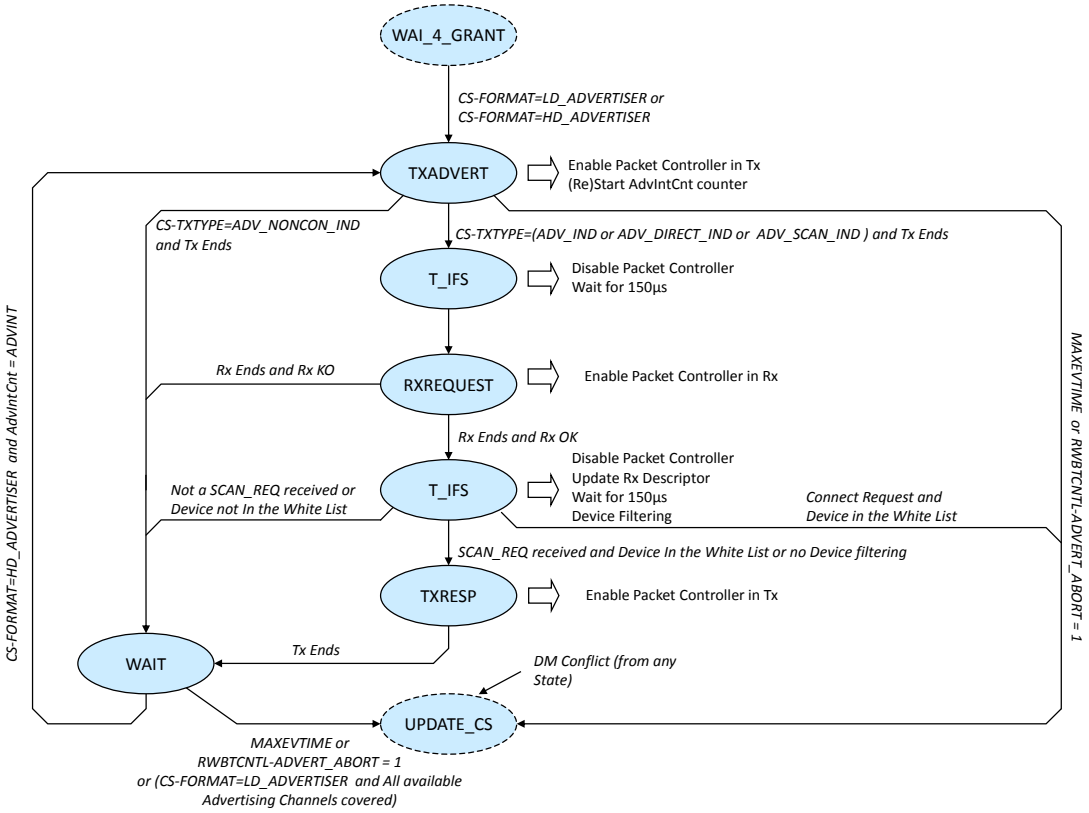

Figure 2-35 – Event Controller "Advertising" sub-FSM

Event Controller scanning/initiating sub-FSM is detailed in Figure 2-36.

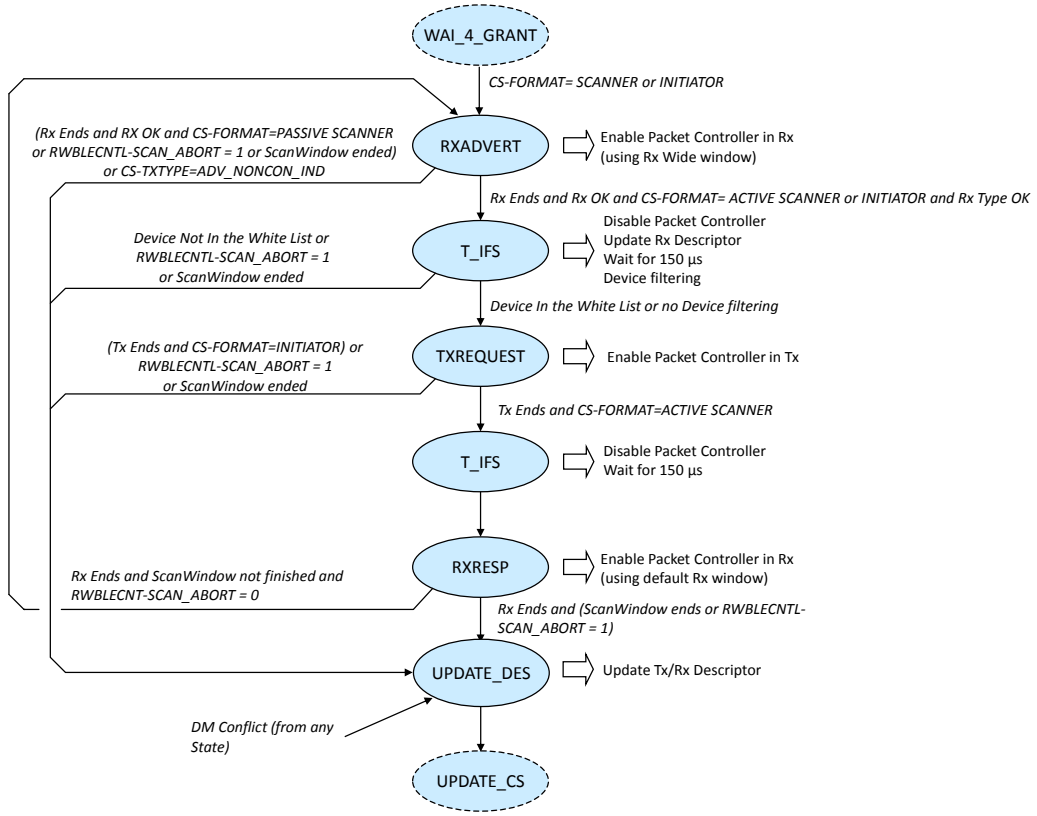

Figure 2-36 – Event Controller "Scanning / Initiating" sub-FSM

Note that Scanning/Initiating window is limited by CS-MAXEVTIME.

Event Controller RF test modes sub-FSM is detailed in Figure 2-37

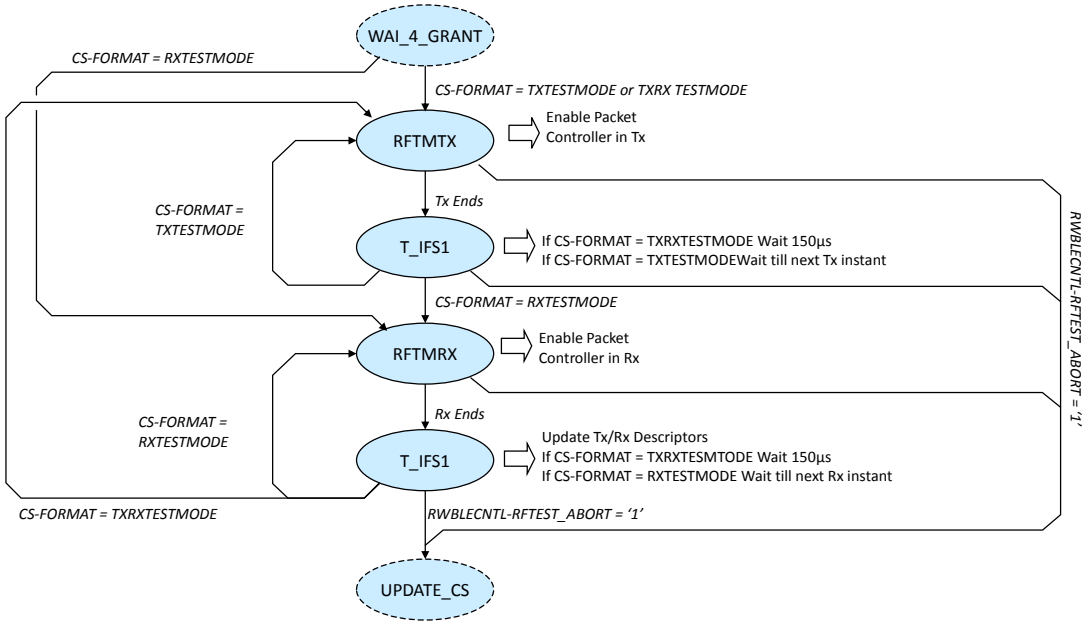

Figure 2-37 – Event Controller "RF Test Modes" sub-FSM

For a detailed description of the Control Structure content, please refer to section 3.3.

Note that Tx/Rx CCM packet counters are re-used for Tx/Rx RF testing and PER estimate purposes, and as such:

- if RFTESTCNTL-TXPKTCNTEN is set, then CS-TXCCMPKTCNT is reset to 0x0 when RF Tx Test Mode start, and incremented each packet sent. This counter is reported in RFCNLTTXSTAT-TXPKTCNT field over 32 bits only
- if RFTESTCNTL-RXPKTCNTEN is set, then CS-RXCCMPKTCNT is reset to 0x0 when RF Rx Test Mode start, and incremented each correctly received packet (no errors). This counter is reported in RFCNLRTXSTAT-RXPKTCNT field over 32 bits only

Note also that

- in Tx Direct Test Mode (i.e CS-FORMAT = TXTESTMODE), Tx interval depends on Tx packet length defined by TxDESC-TXADVLEN
- in Rx Direct Test Mode (i.e CS-FORMAT = RXTESTMODE), Rx interval depends on decoded Rx packet length.

Tx and Rx interval values to be used are defined in

Tx/Rx Packet Length	Tx/Rx instant
[0:37] bytes	625µs
[38:115] bytes	1250µs
[116:193] bytes	1875µs
[194:255] bytes	2500µs

Table 2-22 – Tx/Rx Direct Test Mode intervals

2.12.4 Packet Controller

Packet Controller is in charge of transmitted packet construction, and received packet recovery. The Packet Controller has only access to the Tx and Rx Data Buffers, plus the Control Structure fields related to Packet Header. The Packet Controller is in charge of the control of the CRC and whitening blocks. It is also in charge of starting the Resolving Address List Search engine.

Packet Controller FSM is detailed in Figure 2-38.

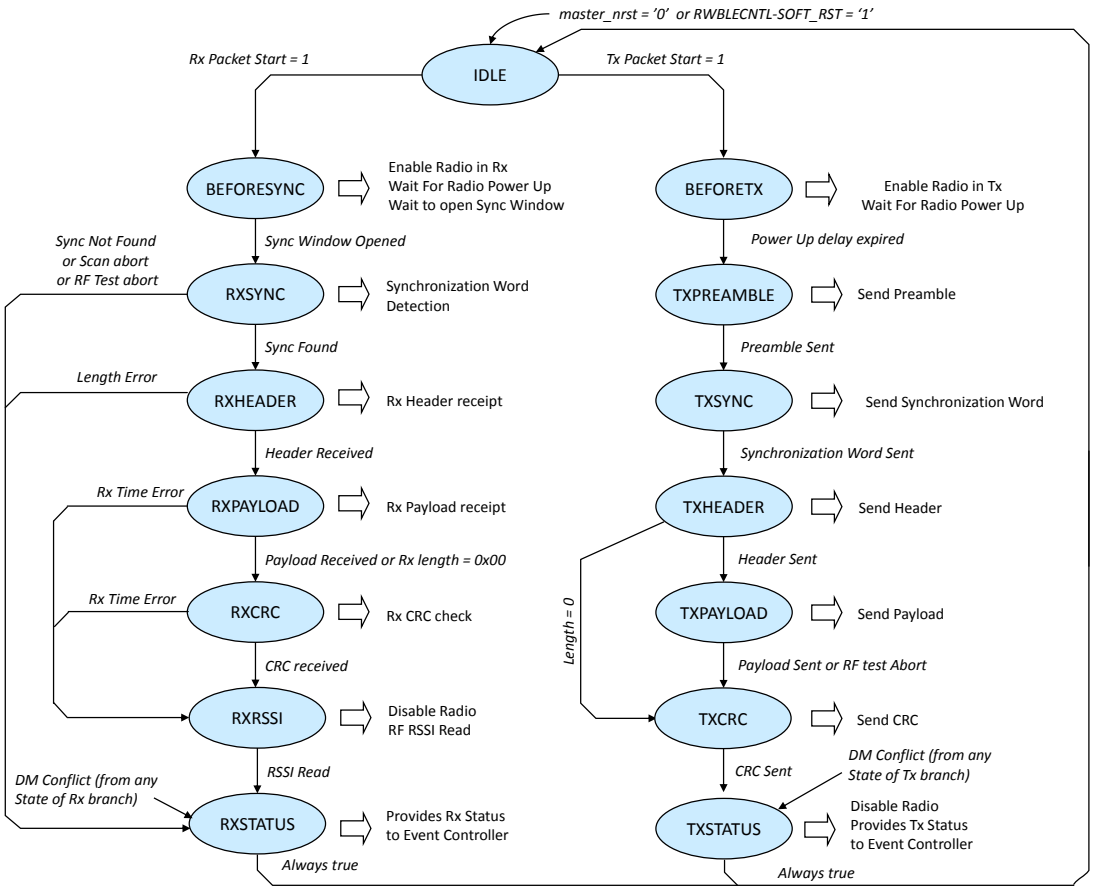

Figure 2-38 – Packet Controller FSM

2.12.5 White List Search Engine

White List Search engine is in charge of parsing either Public White List or Private White List. It is under controller of the Event Controller. It is enabled according to the Device Filtering Policy, the Privacy Mode and the Local and Peer Address Types.

White List Search engine is used when

- CS-RAL_EN = 0
- CS-RAL_EN = 1 and when Resolving Address List Identity address search is not successful

White List Search engine FSM is detailed in Figure 2-39.

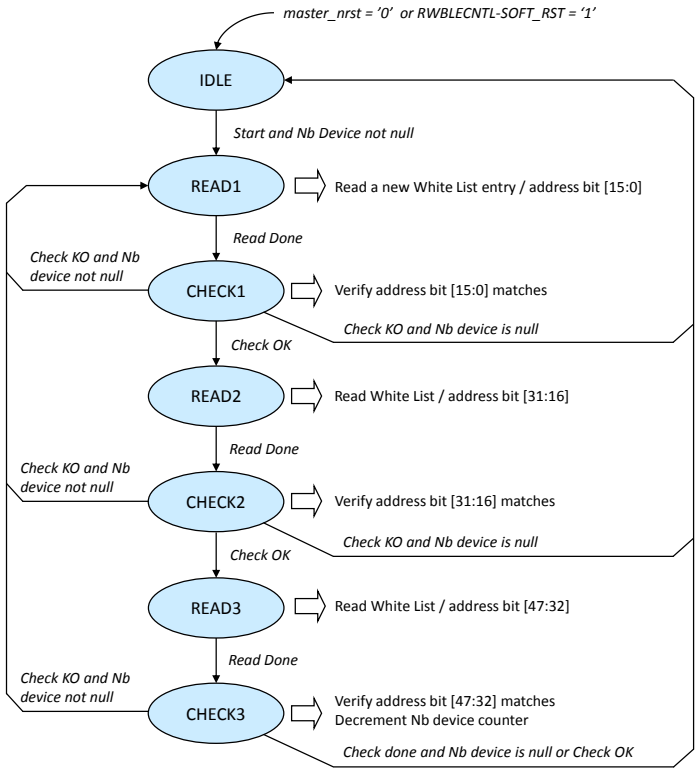

Figure 2-39 – White List Search Engine FSM

2.12.6 Resolving Address List Engine

Resolving Address List engine is in charge of parsing the resolving List. It is under controller of the Event Controller for RPA renewal and Packet controller for Address resolution. It is enabled when the Enhanced Privacy Mode is enabled. Resolving Address List Search engine is used when CS-RAL_EN = 1.

The Resolving Address List FSM performs several tasks (if RAL-ENTRY_VALID=1 and RAL_CONNECTED=0) with a priority order as defined in Table 2-23.

Priority Level	Tasks	Valid CS-FORMAT / Valid Packet type	Comments
Highest	Peer RPA resolution	<LD/HD>_ADVERTISER and <SCAN/CONNECT>_REQ reception <PASSIVE/ACTIVE>_SCANNER and ADV_<>_IND reception INITIATOR and ADV_IND/ADV_DIRECT_IND reception	performed if RAL-PEER_IRK_VALID = 1 else skipped
	Peer RPA comparison	<LD/HD>_ADVERTISER and <SCAN/CONNECT>_REQ reception <PASSIVE/ACTIVE>_SCANNER and ADV_<>_IND reception INITIATOR and ADV_IND/ADV_DIRECT_IND reception	performed if RAL-PEER_RPA_VALID = 1 else skipped
	Peer RPA renewal	<LD/HD>_ADVERTISER and ADV_DIRECT_IND transmission	performed if RAL-PEER_RPA_RENEW = 1 else skipped Performed onto a single RAL entry selection
	Peer ID comparison	<LD/HD>_ADVERTISER and <SCAN/CONNECT>_REQ reception <PASSIVE/ACTIVE>_SCANNER and ADV_<>_IND reception INITIATOR and ADV_IND/ADV_DIRECT_IND reception	performed if Peer Address is not a RPA then compare RAL_PEER_ID_TYPE with TxADD, and compare PEER_ID with received AdvA,
	Local RPA resolution	<PASSIVE/ACTIVE>_SCANNER and ADV_<>_IND reception INITIATOR and ADV_IND/ADV_DIRECT_IND reception	performed if RAL-LOCAL_IRK=1 else skipped
	Local RPA Comparison	<PASSIVE/ACTIVE>_SCANNER and ADV_<>_IND reception INITIATOR and ADV_IND/ADV_DIRECT_IND reception	performed if RAL-LOCAL_RPA_VALID=1 else skipped

	Local RPA renewal	<LD/HD>_ADVERTISER and ADV_<>_IND transmission ACTIVE_SCANNER and SCAN_REQ transmission INITIATOR and CONNECT_REQ transmission	performed if CS-LOCAL_RPA_SEL=1 and RAL-LOCAL_RPA_RENEW=1 and RAL-LOCAL_RPA_VALID=1 else skipped
Lowest	Read Local RPA	<LD/HD>_ADVERTISER and ADV_<>_IND transmission ACTIVE_SCANNER and SCAN_REQ transmission INITIATOR and CONNECT_REQ transmission	performed if if CS-LOCAL_RPA_SEL=1 and RAL-LOCAL_RPA_RENEW=0 and RAL-LOCAL_RPA_VALID=1 else use BDADDR register

Table 2-23 – Resolving Address List FSM Action List and priority

In addition, any Privacy error detection prevents any reply to the received packet, while it is reported in a Rx Descriptor. Privacy error happens when:

- Peer RPA is found with RAL-CONNECTED bit set.
- Peer device Identity is received in Tx<AdvA/ScanA/InitA> field, as well as in RAL structure, but RAL-PEER_IRK_VALID is set

Note that Peer and Local RPA random number generation used for RPA renewal are managed through LFSRs. Initialization values are provided by the RW-BLE Core using RAL_<LOCAL/PEER>_RND registers (See section 3.12.12). LFSRs are initialized by the RW-BLE Software, and updated on each Event Controller start (whatever the CS-FORMAT). Updated values are reported in RAL_<LOCAL/PEER>_RND respective registers

Each LFSR uses the same polynomial that is:

$$p(x) = 1+ x^2+ x^3+ x^5+ x^7+ x^8+ x^9+ x^{10}+ x^{11}+ x^{15}$$

Local and Peer random number generation are respectively built by appending 2 MSB bits which equal to “01” to the LFSR content (See [1]).

Resolving Address List engine FSM is detailed in Figure 2-40:

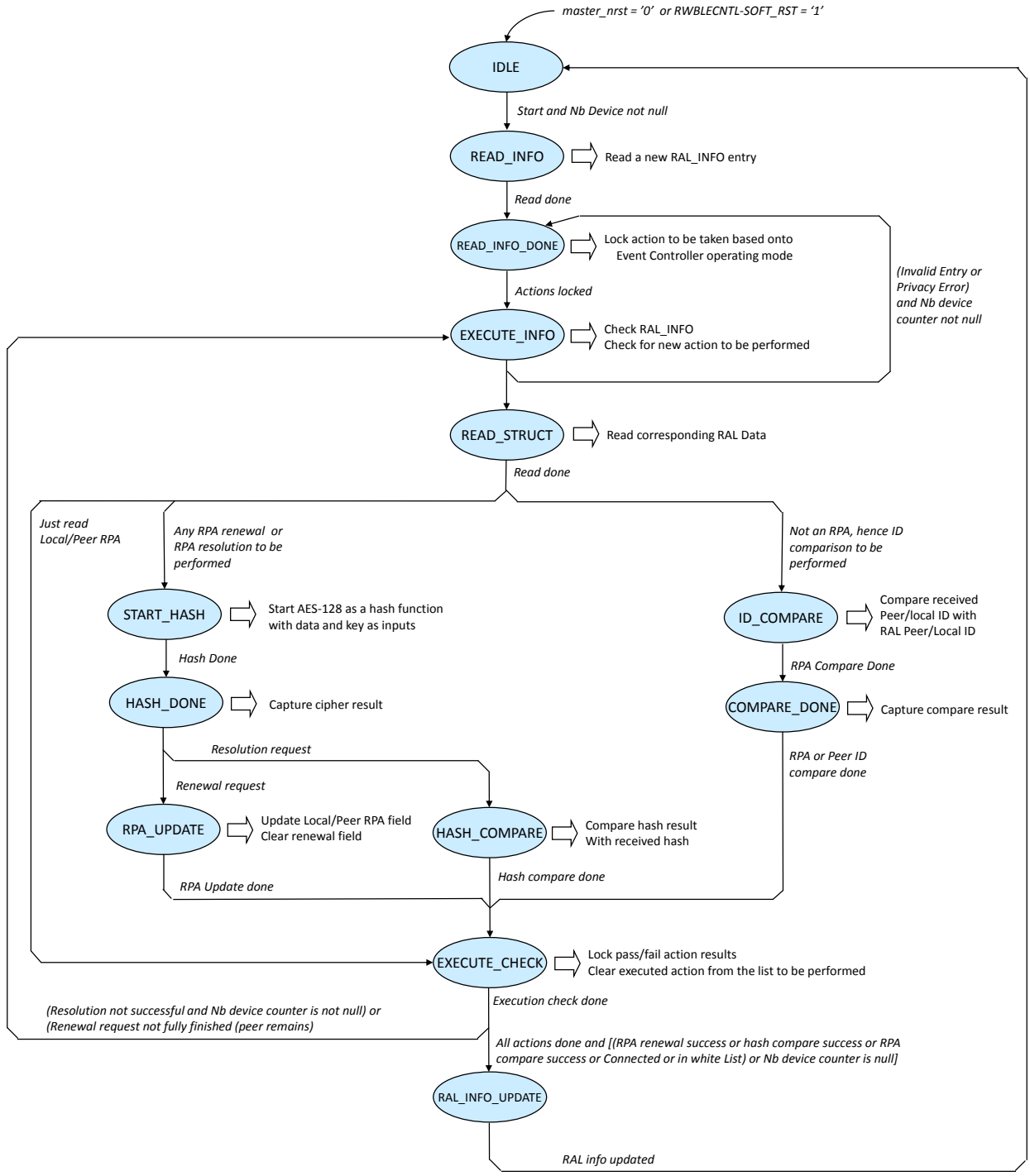

Figure 2-40 – Resolving Address List FSM

2.12.7 Event management

The start of an event is determined by reading the content of the Exchange Table when CSBitCount=TIMGENCNTL-PREFETCH_TIME. Depending on the type of device (i.e. Master, Slave, Broadcaster, Advertiser, Scanner, Initiator), the

round trip delay offset (defined by RADIOPWRUPDN-RTRIP_DELAY) is applied differently (Please refer to the radio User Manual documents).

2.12.7.1 Advertiser and Master Device event management

In the case of a Master or Advertiser device, the Tx packet is started first, as shown in Figure 2-41. Then comes the Rx Packet as depicted in Figure 2-43 (Synchronization word detected) or Figure 2-44 (Synchronization Word not detected). Please note that pre-fetch instant equals 312.5µs in these figures.

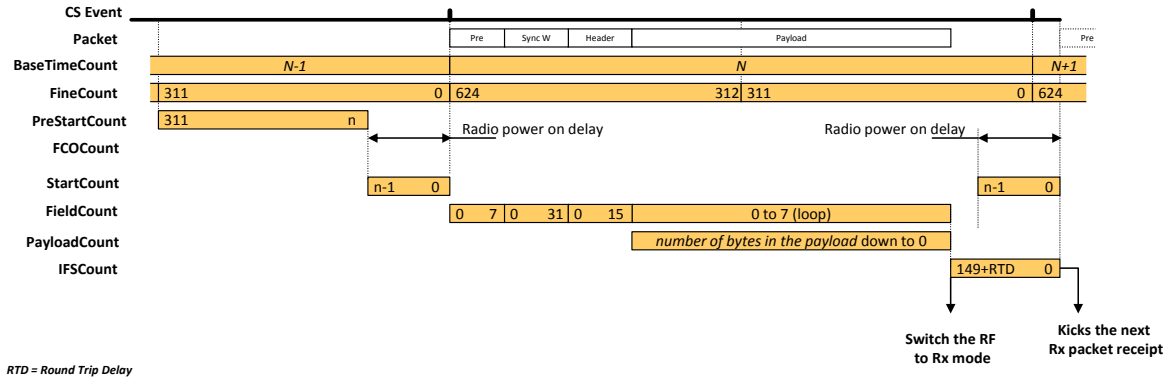

Figure 2-41 – Start of a Tx packet – Master or Advertiser Device

The *n* value is programmed using RADIOPWRUPDN register; it defines the maximum length of the power up sequence. The actual values at which radio events are triggered depend on the type of RF chip controlled by the RW-BLE Core and are hard–coded in the Radio Controller. These values are compared with StartCount to start specific actions.

It is also possible, given certain conditions, to delay the start of the first Tx packet of the event (e.g. Dual Mode implementation with BR/EDR controller a slave device). This can be done through the mean of CS-FCNTOFFSET usage, as shown in Figure 2-42.

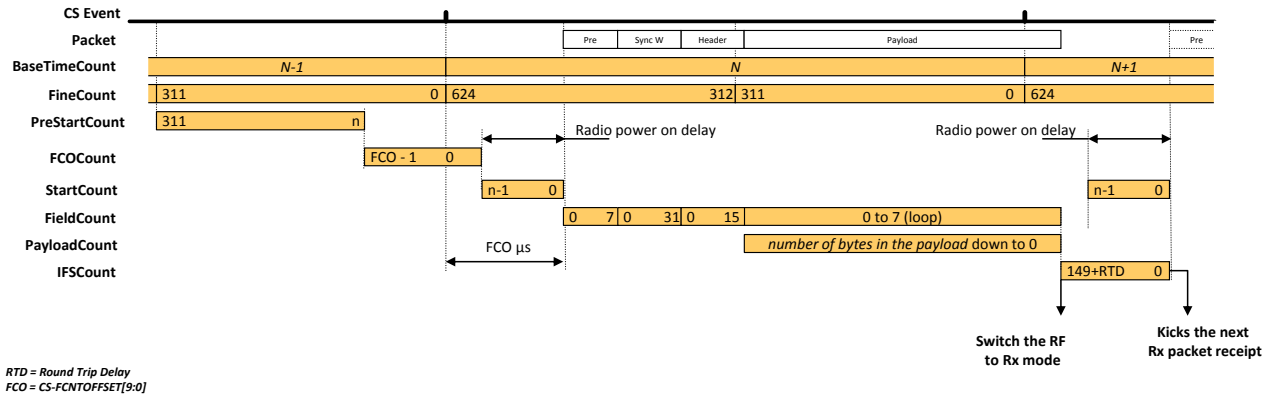

Figure 2-42 – Start of a Tx packet with delay – Master or Advertiser Device

The Rx packets sequencing is slightly different than for the Tx packets.

Figure 2-43 shows the case where synchronization word is detected.

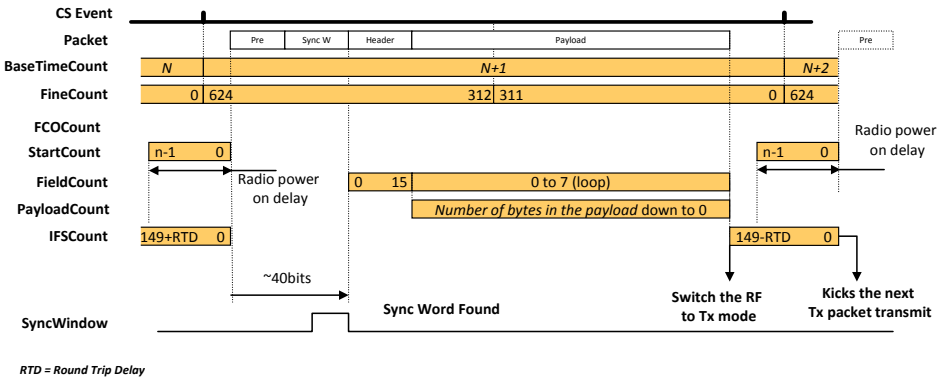

Figure 2-43 – Start of an Rx packet – Master or Advertiser Device with Packet Detected

Figure 2-44 shows the case of a window that is closed without having detected any synchronization.

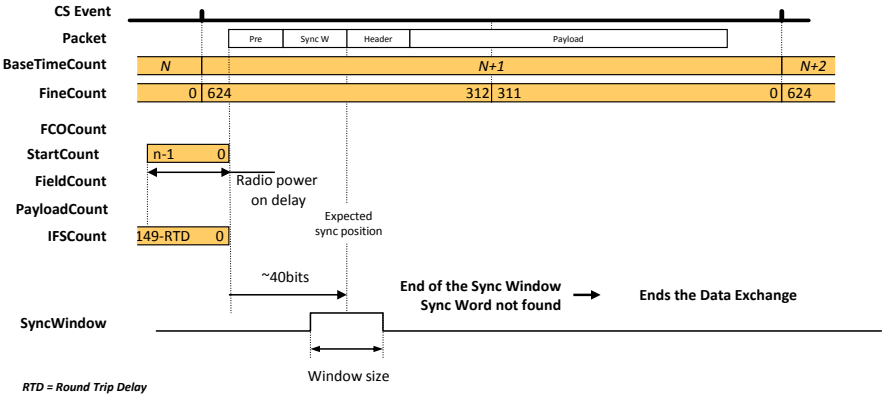

Figure 2-44 – Start of an Rx packet – Master or Advertiser Device with Packet Not Detected

2.12.7.2 Slave, Scanner or Initiator Device event management

In the case of a Scanner or Initiator device, the Rx packet is started first, as shown in Figure 2-45. In case of a Slave device, the Rx Packet is started first, but a delayed Synchronization window opening can be added in order to cope with synchronization window in [RWBLECNTL-RXWINSZDF:625µs] range, as shown in Figure 2-46. The synchronization mechanism is performed on each received packet in within the events. Please note that pre-fetch instant equals 312.5µs in these figures.

The synchronization window width mainly depends on the connection event interval time defined as per connection request. As connection event interval maximal time can be 2000s (i.e. 500×4s), the synchronization window width can be as big as ±1600 times 625µs base time reference (i.e. requires to program CS-RXWIDE = 1 and CS-RXWINSZ = 3200) on each event first packet receipt (on slave side). Once the first packet is received and as the internal counter has been updated (i.e. connection successfully established), the synchronization window can be reduced ideally down to ±0.3µs that is programmed using RWBLECNTL-RXWINSZDEF register. However, due to round trip delay that varies in

between two different radios, it is recommended to use a bigger synchronization window size. If the synchronization word detection is not performed during synchronization window, the event is closed.

Figure 2-45 shows Rx packet receipt for a slave device when a wide synchronization window is opened.

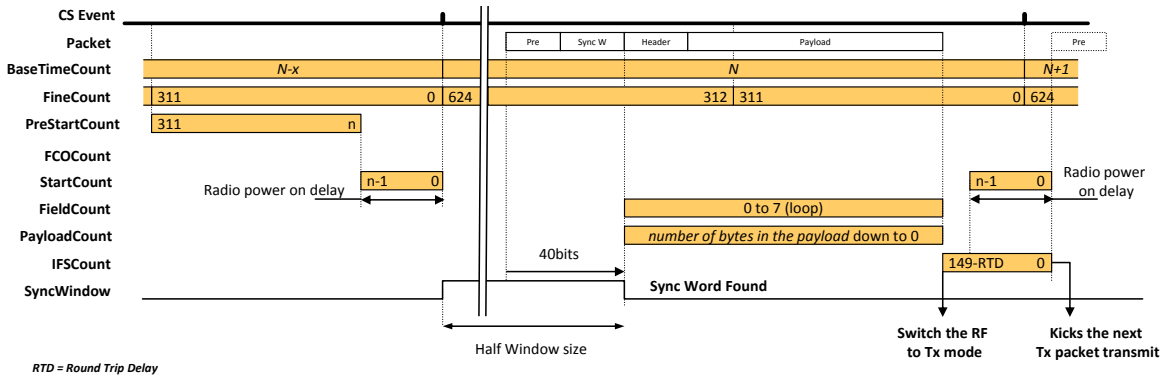

Figure 2-45 – Start of an Rx packet without delay – Slave, Scanner or Initiator Device

Figure 2-46 shows Rx packet receipt for a slave device when a bigger synchronization window than RWBLECNTL-RXWINSZDEF value is opened. The size of the Rx window must be compensated by anticipating the power-up of the Radio. In this case, an offset is added to the instant at which the Radio is supposed to be powered on.

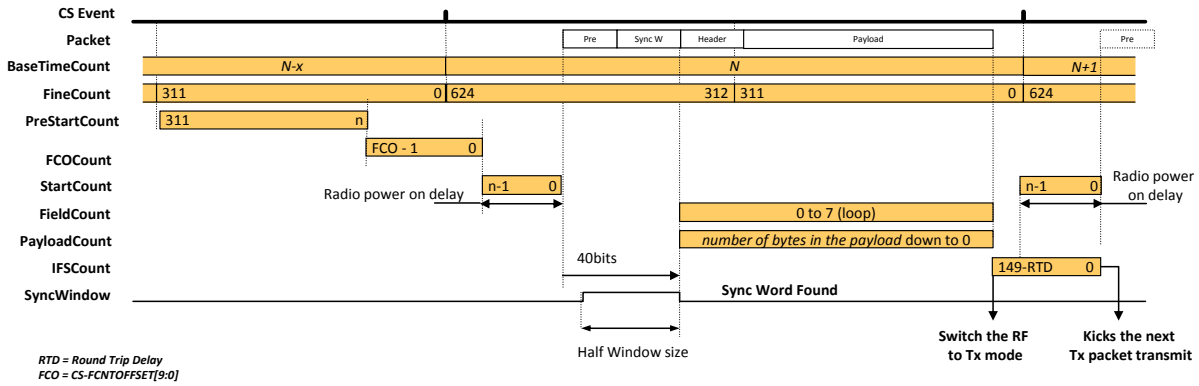

Figure 2-46 – Start of an Rx packet with delay – Slave, Scanner or Initiator Device

When the CS-RXWINSZ is lower than 625µs, and when it is bigger than RWBLECNTL-RXWINSZDEF, the pre-fetch instant must be anticipated by 625µs. FCNTOFFSET value must be set in the Control Structure, and it is determined according to the following formula:

$$FCNTOFFSET = 624 - \left(FCNTRXSYNC + 40 + \frac{RXWINSZ}{2} \right) \bmod 625$$

Note that if the result in parenthesis is bigger than 625, then the pre-fetch time must be anticipated by twice 625µs.

Note that the value 40 corresponds to Packet Preamble + Access Address duration (i.e 8+32 = 40µs).

Figure 2-47 shows the 1st Tx packet transmission for a slave device.

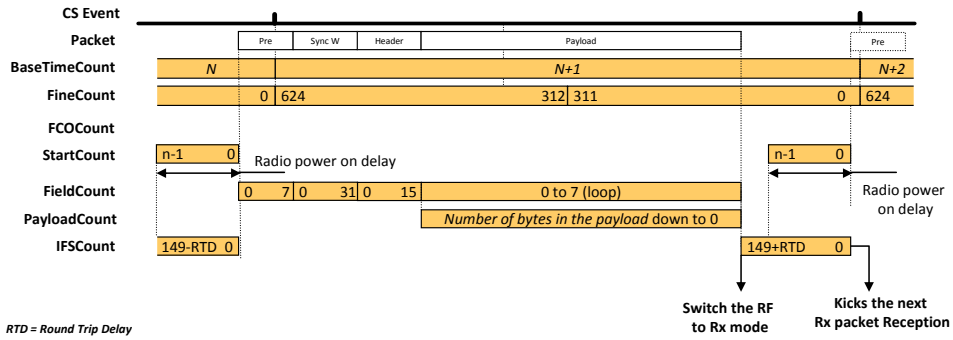

Figure 2-47 – Start of a Tx packet – Slave, Scanner or Initiator Device

Note the round trip delay compensation between two packets depends on the radio used with the RW-BLE Core and is implemented in RADIOPWRUPDN-RTRIP_DELAY register. It is equal to the receive delay plus the transmit delay in the RF chip.

2.12.7.3 Anticipated pre-fetch mechanism management

Anticipated pre-fetch mechanism allows the RW-BLE Core to prevent from event overlapping. The mechanism requires the current event to be stopped earlier enough to ensure next event proper Control Structure pre-fetch and execution. This mechanism is enabled only when conditions described in Table 2-14 are met.
Figure 2-48 and Figure 2-49 illustrates anticipated pre-fetch counters behavior respectively with events with no delay, and delayed events.

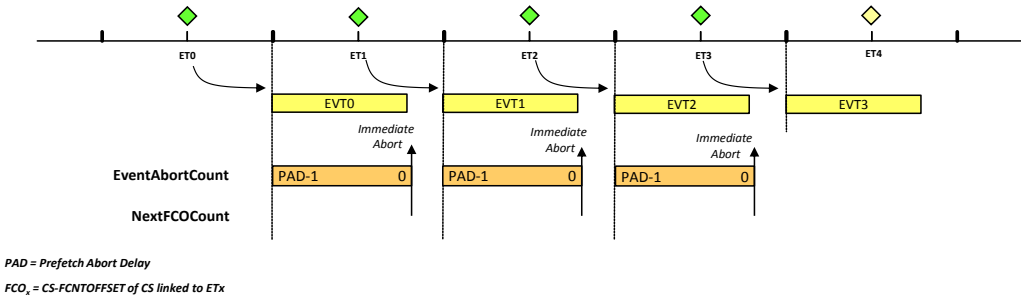

Figure 2-48 – Anticipated pre-fetch counters / Event with no delay

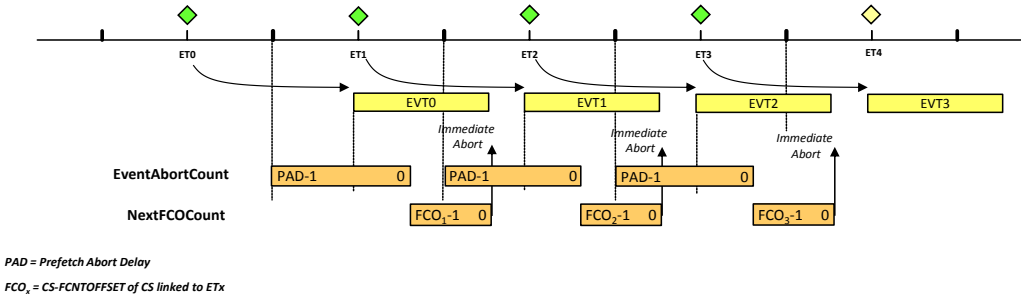

Figure 2-49 – Anticipated pre-fetch counters / Delayed Event

2.13 Frequency selection

The frequency selection block calculates automatically the frequency to be used for the packet, depending on the parameters given by the software in the CS–FORMAT field. It supports three main frequency hopping schemes composed by:

1. 37 data channels (index in [0:36] range)
2. 3 advertising channels (index in [37:39] range)

Table 2-24 details how to select in between the three hopping frequency schemes.

CS-FORMAT	Channel Type
Master Connect	Data Channel
Slave Connect	Data Channel
Advertiser	Advertising Channel
Passive Scanner	Advertising Channel
Active Scanner	Advertising Channel
Initiator	Advertising Channel
Tx Test Mode	n/a (1)
Tx Test Mode	n/a (1)
Tx/Rx Test Mode	n/a (1)

Table 2-24 – Frequency Hopping Scheme Selection

(1): In test mode, the CS-CH_IDX is used blindly till abort is required

The selected frequency is used to program the radio before any transmission or reception. Depending on the operating mode, the hopping scheme is adapted as described in the Bluetooth Low Energy specifications: advertising, scanning request / response, connection request, and master or slave connection are supported, as described in [1].

Table 2-25 provides correspondence between channel type, channel index, and frequency.

Frequency (MHz)	Channel Type	Channel Index	Channel Number
2402	*Advertising Channel*	*37*	*0*
2404	Data Channel	0	1
2406	Data Channel	1	2
…	Data Channel	…	…
2424	Data Channel	10	11

2426	*Advertising Channel*	*38*	*12*
2428	Data Channel	11	13
…	Data Channel	…	…
2478	Data Channel	36	38
2480	*Advertising Channel*	*39*	*39*

Table 2-25 – Bluetooth Low Energy Channel Index and Physical Channel Correspondence

2.13.1 Advertising Channel Frequency Hopping

Advertising channel frequency hopping is performed according to ADVCHMAP register, which determines advertising channels availability. Advertising channels must be used in ascending order (i.e. from index 37 up to 39). The Host ensures at least one advertising channel must be available during advertising event.

2.13.2 Data Channel Frequency Hopping

On any event, the RW-BLE Core must use the channel index provided in the control structure with CS-CH_IDX. Link Layer channel frequency hopping process is run at the beginning of an event in order to determine the Channel to select. Once the event is closed, CS-CH_IDX must be updated for its use on the next event. This processing applies whenever a programmed event is discarded by Dual Mode Priority arbitration mechanism.

Data channel frequency hopping is performed according to CS-LLCHMAP, which determine Link Layer channels availability. For each Link Layer, the hopping sequence can be modified so as to allow coexistence with non-Bluetooth Low Energy devices operating in the ISM band, as described in [1]. The method consists on assessing what channels are not suitable for transmission (most likely due to interference from non-Bluetooth Low Energy device sources), and removing them from the Connection hopping sequence. This task is handled by the master device of the Bluetooth Low Energy network that decides which channel has to be removed from the channel list.

Frequency hopping scheme is based onto a basic algorithm, and a remapping algorithm.

Basic algorithm is defined as:

$$NewEventChannel = (\text{CS-CH\_IDX} + \text{CS-HOPINT}) \bmod 37$$

If the *NewEventChannel* is not in the LLCHMAP available channel list, then the remapping algorithm is executed. Remapping algorithm is defined as:

$$RemapIndex = (NewEventChannel) \bmod (\text{NBCHGOOD})$$

RemapIndex is then use to remap the data channel using it as a pointer onto a table composed with only good channels in ascending order.

Each connection event must start with *NewEventChannel = 0,* and CS-CH_IDX=0.

2.13.3 Channel Assessment

The Channel Assessment algorithm is based on combining RSSI for each channel with packet reception status. For this:

- RSSI is supplied at the end of the synchronization window (for connection formats, and regardless of reception status). RSSI Channel assessment information is provided through RxDESC-RXRSSI.
- Current Bluetooth Low Energy used channel index is reported to RxDESC-USED_CH_IDX (after frequency hopping calculation). This memory access is handled by Core Controller block, as all other Control Structure and Descriptors status updates. Depending on the RW-BLE Core state (i.e. idle, advertising, master connected, slave connected), the Core Controller uses either LLCHMAP, or ADVCHMAP.

RxDESC-USED_CH_IDX actually holds the radio device dependent programming word used by the Radio Controller. Each radio device features its own relationship between programming words and Bluetooth Low Energy channels. Current Software implementation relies on Bluetooth Low Energy channel being equal to programming word plus a device dependent offset. RSSI channel assessment is performed essentially for all data channels available. The RW-BLE Software is in charge to collect RSSI data and update channel mapping table.

2.14 Radio Controller

2.14.1 General Consideration

The RW-BLE Core is able to support several different radio chips, by selecting a Radio Controller block dedicated to the desired one.

Figure 2-50 shows the block diagram of the block.

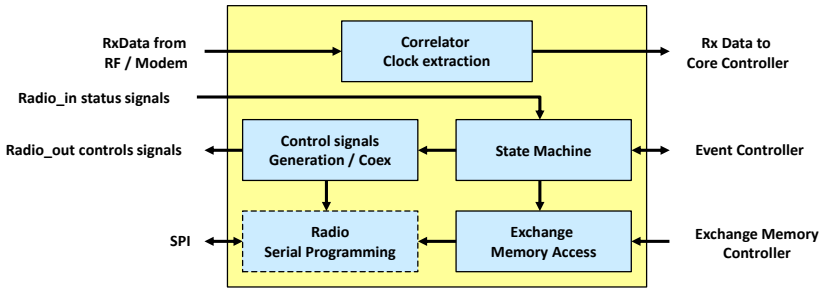

Figure 2-50 – Radio Controller Basic Block Diagram

This block always has the same interface to the internal side of the RW-BLE Core, whereas the external pins depend on the selected radio chip.

As the Radio Controller is dedicated to a specific radio, it contains all the elements that can be specific to the radio: control of the frequency (i.e. channel selection), sending (and filtering if necessary) of the transmitted data stream, clock recovery, DC compensation, synchronization word correlation, serial interface, etc… A more detailed description of the supported radios and corresponding interfaces can be found in their respective user manual.

Please note that unused outputs of *radio_out* must be ignored, as well as unused inputs of *radio_in* must be tied to logic 0. It is up to the designer to take this into account during RW-BLE Core implementation, according to the targeted radio(s). It is up to the user to refer to the dedicated User Manual documents(s).

Note also the *radio_in* / *radio_out* mapping is generic, and could be updated later according to the introduction of each new Bluetooth Low Energy RF device.

Please refer to supported Radio User Manual for specific details about *radio_in*/*radio_out* mapping.

2.14.2 Generic RF interface

Table 2-26 details RivieraWaves' generic radio controller interface that is available when *RW_BLE_EXTRC_INST* is set.

RW-BLE Core Signal	Comments
radio_out[2:0]	Tx Data / Bit 2 and 1 are forced to 0 during Tx
radio_out[3]	Tx Data qualifier
radio_out[5:4]	Rate / Forced to 01
radio_out[6]	Transmit Enable
radio_out[7]	Receipt Enable
radio_out[15:8]	Transmit Power
radio_out[16]	Access Address detection (*sync_p*)
radio_out[17]	Reserved / Forced at 1
radio_out[18]	Reserved / Forced at 0
radio_out[19]	RSSI request indicator
radio_out[27:20]	Bluetooth Channel
radio_out[30:28]	Reserved / Forced at 0
radio_out[31]	Optimal sampling train pulses
radio_out[39:32]	Tx/Rx Packet length in bytes
radio_out[40]	Tx/Rx Packet length valid
radio_out[41]	Event in process
radio_out[47:42]	Reserved / Forced at 0
radio_in[2:0]	Rx Data / Bit 2 is reserved, Bit 1 can be used depending on RADIOCNTL0-DPCORR_EN value as well as connected RF capabilities, else it has to be tied at 0
radio_in[3]	Reserved
radio_in[11:4]	RSSI Value input
radio_in[12]	RSSI Value ready indicator
radio_in[13]	RF Busy indicator
radio_in[47:14]	Reserved

Table 2-26 – RivieraWaves' Generic Radio Interface

Note that reserved *radio_in* input must be tied to 0.

Note also the radio interface behavior depends on each individual radio, and this table is subject to updates. Please refer to related User Manual documents for further details.

2.14.3 Interface Timings

Figure 2-51 shows RivieraWaves' generic radio controller interface transmission timing diagram. In this case, *radio_out[2:1]* is forced to 0x0.

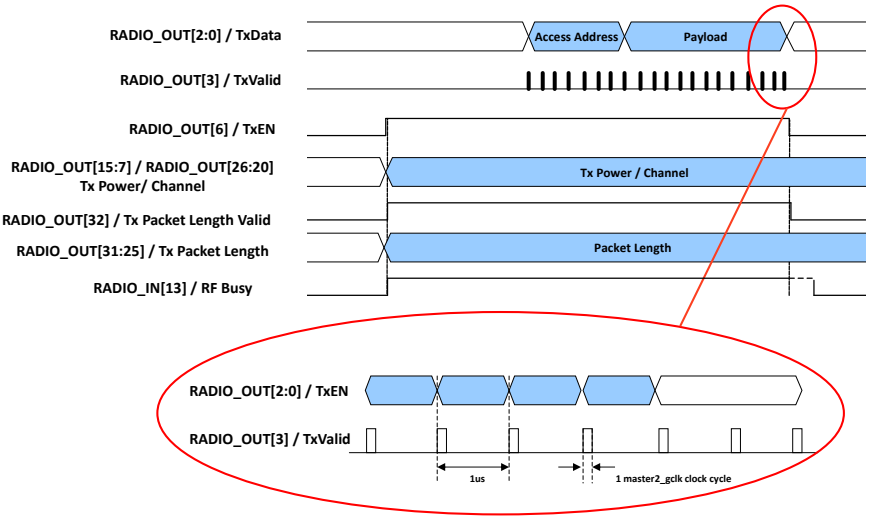

Figure 2-51 – Transmission Timing Diagram

Figure 2-52 shows RivieraWaves' generic radio interface reception timing diagram. In this case, *radio_in[2]* is not used by the RW-BLE Core.

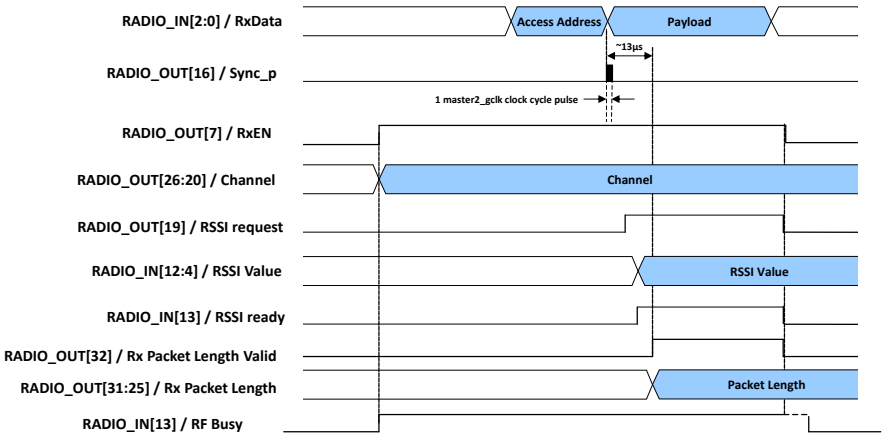

Figure 2-52 – Reception Timing Diagram

Note that Radio Controller correlator performs clock recovery and Rx data sampling decision. Correlator supports also a dual path used for DC-compensated received data.

Please refer to section 2.14.4 for further correlator details.

2.14.4 Correlation and Clock extraction

In order to synchronize the incoming bit stream, a clock must be extracted from the received data flow. This extraction can be performed either in the radio part or in the baseband part, depending on the selected radio.

RivieraWaves implementation considers it is not performed in the radio, hence the baseband performs also clock extraction.

The received data stream is resynchronized in the Radio Controller, in order to align the bits on the RW-BLE Core's internal bit timing. It is very important to resynchronize the sync pulse as well, since a timing mismatch between these events would result in the loss or doubling of one received bit. The clock extraction is performed by determining the middle position of received bits.

The only known information is the edge's position between bits of different value. As the synchronization word is not fixed, and there are few chances to get any of the eight preamble bits, this detection must be done on an undefined pattern, hence CS-SYNCWORD existence.

Synchronization Word detection, as well as data stream resynchronization is performed using the Correlator. It is used to recognize a given synchronization word, even when some errors occurred during the transmission or the reception. RWBLECNTL-SYNCERR indicates the maximum number of errors allowed to trigger the synchronization detector. Note that RWBLECNTL-SYNCERR cannot be used during normal operation and is dedicated to testing and debug only.

This correlation is simply a XOR of the content of the shift register filled with received data and the expected synchronization word. The number of 1 in the result of this operation is counted to determine the number of different bits. If this sum is greater than RWBLECNTL-SYNCERR, no sync is detected. In the other case, a Correlation Window is detected, *sync_p* synchronization pulse is sent to the RW-BLE Core's state machines to start the packet processing, and optimal sampling point is calculated. In order to get a more precise sampling point, the shift register over-samples the Rx data stream.

Correlator block is able to handle a dual GFSK data path stream with fixed DC offset compensation delay, as shown in Figure 2-53.

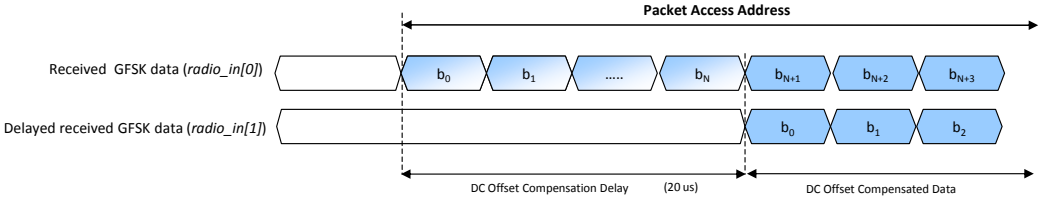

Figure 2-53 – Dual path Correlator GFSK data stream

Figure 2-54 shows the correlator structure.

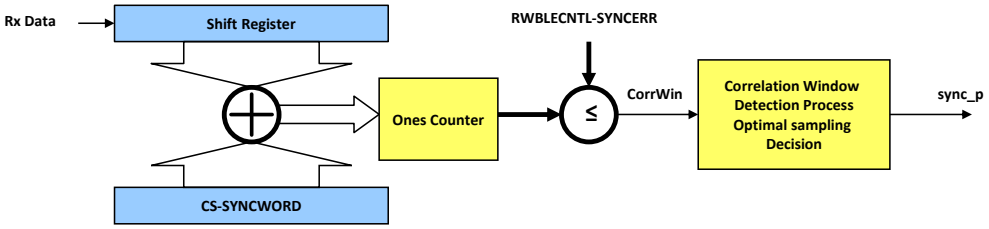

Figure 2-54 – Correlator Structure

2.14.5 Control signal generation

The radio component is controlled by real–time signals toggling before and after the packet to switch on or off the data paths. The radio power sequence is defined by a dedicated counter (*StartCount*, described in Section 2.12.1), and all actions are linked to the value of this counter.

Note that for each radio chip, there exists a dedicated user manual, in which all the relevant constants and connection information are detailed.

2.14.6 Programming Interface

A programming interface (in most of the cases serial programming interface / SPI) is used to program the radio component. Note all the controls are provided through *radio_out* as described in Table 2-26. Specific bits in RADIOCNTL0/1 registers are dedicated to the control of the SPI, as well as debugging controls allowing the RW-BLE Software to pass command to RF through SPI real frame (i.e. in between events). Please refer to User Manuals for further details.

Radio programming occurs at initialization time and before every advertising or connection event to select the correct frequency and other parameters dependent on the RF chip.

Before each event, the frequency is sent to the radio. This is done in the Radio Controller and depends on the selected radio. The frequency tables, stored in the Exchange Memory, contain the frequency words to be sent to the radio. The Radio Controller composes the SPI command by adding the command and address words to it. Frequency tables are located in a fixed location in the Exchange Memory (see section 2.9).

Figure 2-55 describes the default frequency table layout, and its correspondence to BR/EDR channel mapping (Please refer to for [1] details)

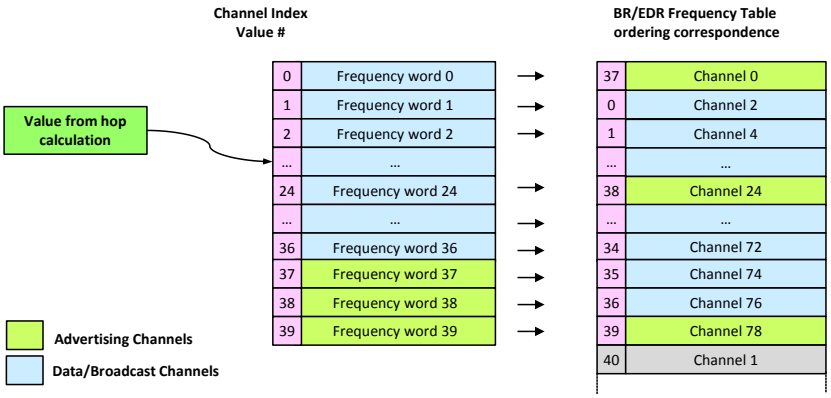

Figure 2-55 – Mapping of hop selection to frequency word

Frequency tables for frequency hopping are described in sections 2.13.1 and 2.13.2.

Please note that both advertising and broadcast channels are excluded from frequency hopping mechanism, and that both advertising and broadcast channels are used in ascendant order, according to their availability.

2.15 WLAN Coexistence

RW-BLE Core includes the following provisions for Coexistence with WLAN devices:

- ✓ Ability to report real time BLE Tx and Rx activity to a collocated WLAN device
- ✓ Ability to abort BLE Tx and/or Rx activity when real time WLAN Rx activity is reported

- ✓ Ability to abort BLE Tx and/or Rx activity when real time WLAN Tx activity is reported
- ✓ A Bluetooth Low Energy real time synchronization signal

Coexistence signals and the corresponding WLAN mechanism principles are described in the next sections.

2.15.1 WLAN Coexistence basics

In a WLAN – Bluetooth Low Energy coexistence scenario, RW-BLE Core Radio Controller might be requested to abort transmission and/or reception by means of either the *wlan_rx* or *wlan_tx* signals, so as to prevent mutual interfering with a collocated WLAN device. These requests are generated outside the RW-BLE Core, and are asynchronous to its internal time base. RW-BLE Core might however refuse this request by setting CS-DNABORT. This allows RW-BLE Core to preserve some critical events (such as dealing with control packets), as well as some connection establishment critical phases.

Note that it is left to the user the possibility to program whether *wlan_tx* and/or *wlan_rx* will have effect on the current event by programming WLANCNTL-WLAN<TX/RX>MSK[1:0] fields (please refer to section 3.12.11 for programming details). Note that the default mode is set with *wlan_rx* that impacts BLE transmission only, and with *wlan_tx* has no impact (neither Tx nor Rx).

Furthermore, the radio controller produces a Bluetooth Low Energy Tx active (*ble_tx*) signal (indicating that the radio is ramping up/down in Tx, or currently transmitting), a Bluetooth Low Energy Rx active (*ble_rx*) signal (indicating that the radio is ramping up/down in Rx or currently receiving). Both *ble_tx* and *ble_rx* generation are controlled by CS-TXBSY_EN and CS-RXBSY_EN respectively (refer to Table 3-2). Two different modes are possible, including or excluding Tx/Rx Power up delays.

Figure 2-56 shows WLAN coexistence interface timing diagram including Tx/Rx Power up delay (i.e WLC<TX/RX>PRIOMODE = “00”): this is the default mode

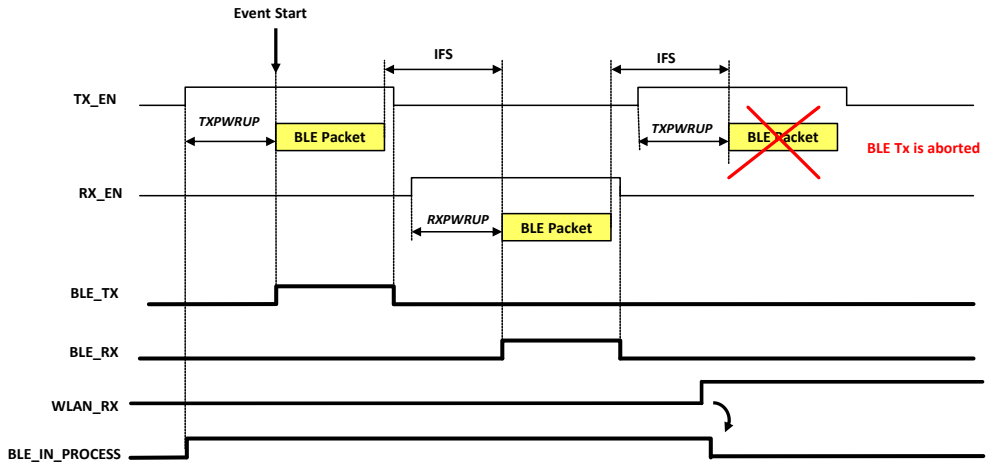

Figure 2-56 – WLAN Coexistence Timing / WLC<TX/RX>PRIOMODE=”00”

Figure 2-57 shows WLAN coexistence interface timing diagram including Tx/Rx Power up delays (i.e WLC<TX/RX>PRIOMODE = “01”)

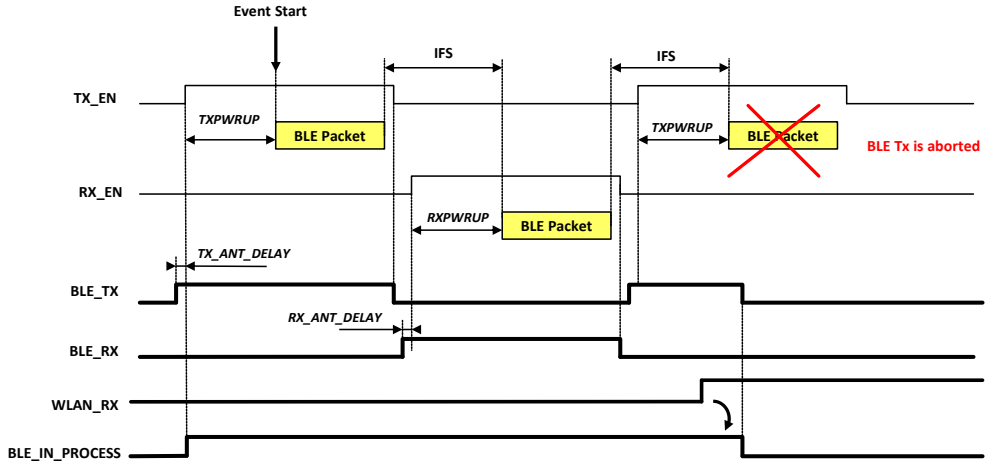

Figure 2-57 – WLAN Coexistence Timing / WLC<TX/RX>PRIOMODE="01"

Additionally, a 3rd mode can be used: WLAN Coexistence interface conveys Tx or Rx priority indicator on *ble_tx* or *ble_rx* respectively, and this whenever CS-TXBSY_EN and CS-RXBSY_EN are set to 0. This behavior is possible when WLC<TX/RX>PRIOMODE="10". Additionally, the priority level can be selected through WLCP<TX/RX>THR field, (i.e. if *ble_pti[3:0]* output value is greater than WLCP<TX/RX>THR, then the Bluetooth Low Energy priority is considered as high on either Tx or Rx, and must be provided to the Coexistence interface. (Please refer to section 2.15.2 for details).

Figure 2-58 illustrates BLE priority conveyed on both Tx and Rx packet start (Case of a device with WLC<TX/RX>PRIOMODE set to "10").

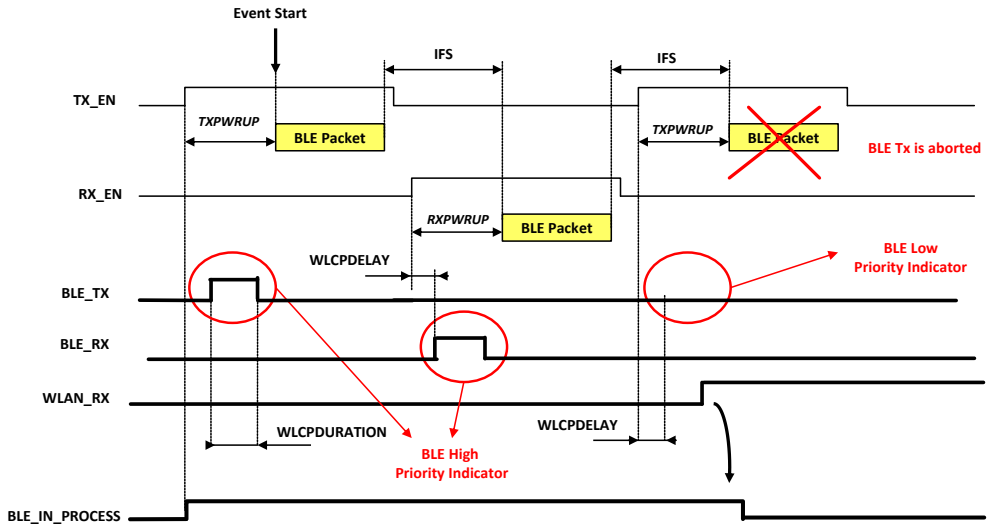

Figure 2-58 – WLAN Coexistence Timing / WLC<TX/RX>PRIOMODE="10"

Note that WLCTXPRIOMODE and WLCRXPRIOMODE can be set independently.

2.15.2 Packet Traffic Arbiter Interface

In order to allow an efficient packet traffic arbitration between Bluetooth Low Energy and any external devices, the *ble_pti[3:0]* output needs to be refresh each BLE Packet (i.e. on each *ble_tx* or *ble_rx* rising edge).

The message events to be reported are listed below, according to their respective index:

- 0: Initiating (Connection Request response)
- 1: LLCP
- 2: Data Channel Transmission
- 3: Initiating (Scanning)
- 4: Active Scanning Mode
- 5: Connectable Advertising Mode
- 6: Non-Connectable Advertising Mode
- 7: Passive Scanning
- 15: Default Priority output value

The priority level must be provided at *ble_pti[3:0]* output. All the necessary information can be found in either Control Structure, or in registers.

Table 2-27 describes the priority assignment according to each message type.

Message type	Priority	Comments
Initiator (Connection Request Response)	15	Applies to CONNECT_REQ transmit and CONNECT_REQ attempt of reception (Advertising or Initiating mode)
LLCP Packets	13	Applies to all LLCP Tx packets, regardless whether the device is Master or Slave
Data Channel transmission	10	Applies to Master and Slave device
Active Scanning mode, Initiating (Scan)	9	Applies when device is receiving in Scanning or Initiating mode
Connectable Advertising mode	8	Applies when device transmit ADV_DIRECT_IND or ADV_IND packets
Non-Connectable Advertising mode	4	Applies when device transmit ADV_NONCONN_IND, ADV_SCAN_IND packets
Passive Scanning mode	3	Applies to Passive Scanning modes
Unknown Message / Default Priority Value	3	Default priority, applies to other BLE message not defined above

Table 2-27 – Bluetooth Low Energy Messages Priority Default Assign

ble_pti[3:0] output drags out the current event priority that depends on event context, the CS-PTI[3:0] fields and the BLEMPRIO<0/1> register settings. On each Control Structure read, the RW-BLE Core determines the event type that is required to be issued, as well as all necessary fields indicating which packets to be exchanged during the current programmed event (e.g, LLCP, Advertising, Scanning, etc..), and some context indications that can be driven on need-basis using CS-PTI[4:0]

According to these values, the RW-BLE Core determines which index has to be used by the priority table in order to point onto the corresponding BLEMPRIO<0/1> register field values to drag out to *ble_pti* [3:0].

In case the RW-BLE Software need to override the register programmed priority values as defined in Table 2-27, it can set CS-PTI to a higher priority index value, matching to corresponding BLEMPRIO register that is routed to *ble_pti*[3:0]

output. *ble_pti*[3:0] output is also reset at end of each event. *ble_pti*[3:0] output is updated on each internal Tx/Rx enable rising edge, and reset on each internal Tx/Rx enable falling edge.

Figure 2-59 illustrates this mechanism.

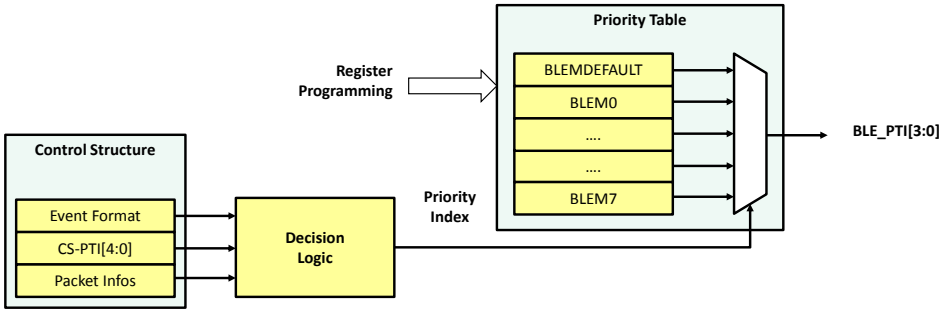

Figure 2-59 – WLAN Coexistence Priority output programming principle

Figure 2-60 shows *ble_pti[3:0]* timing diagrams.

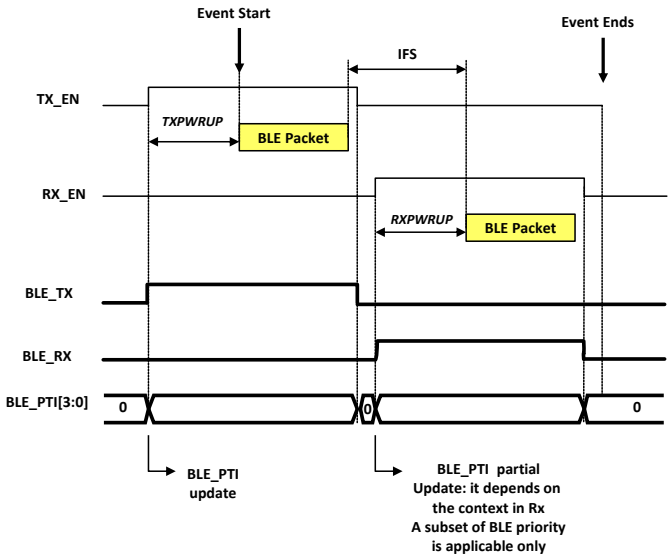

Figure 2-60 – BLE_PTI Timing diagrams / Example with default WLC<TX/RX>PRIOMODE values

2.16 Validation and test

To be able to validate the functioning of the RW-BLE Core in FPGA or once implemented into ASICs, a diagnostic port must be provided at the outputs of the chip. This port may be multiplexed with other functional connectors (GPIO, for example…) to avoid increasing the pin count for a test–only feature. The diagnostic port is multiplexed internally and a selection of the signals to drive on the outputs is given by the DIAGCNTL register. The validation output width is configurable (Refer to DIAGCNTL register controls). The byte–wide observable signal groups are described in Table 2-28. As specified in section 3.12.3, every group can be routed to either the upper and/or lower byte when the output is 16 bit wide.

Connector name	Value of DIAGCNTL							
	0x00	**0x01**	**0x02**	**0x03**	**0x04**	**0x05**	**0x06**	**0x07**
DiagOut[0]	evt_schdl_main[0]	evtcntl_sub_fsm[0]	pkt_main_fsm[0]	radcntl_txen	microseccnt[0]	basetimecnt[0]	basetimecnt[8]	timing_gen_fsm[0]
DiagOut[1]	evt_schdl_main[1]	evtcntl_sub_fsm[1]	pkt_main_fsm[1]	radcntl_rxen	microseccnt[1]	basetimecnt[1]	basetimecnt[9]	timing_gen_fsm[1]
DiagOut[2]	evt_schdl_main[2]	evtcntl_sub_fsm[2]	pkt_main_fsm[2]	sync_window	microseccnt[2]	basetimecnt[2]	basetimecnt[10]	timing_gen_fsm[2]
DiagOut[3]	evt_schdl_apfm_cs[0]	evtcntl_sub_fsm[3]	pkt_main_fsm[3]	sync_found_pulse	microseccnt[3]	basetimecnt[3]	basetimecnt[11]	tick_625us_p
DiagOut[4]	evt_schdl_apfm_cs[1]	evtcntl_sub_fsm[4]	pkt_em_sm[0]	event_in_process	microseccnt[4]	basetimecnt[4]	basetimecnt[12]	tick_prefetch_p
DiagOut[5]	evt_schdl_apfm_cs[2]	evtcntl_main_fsm[0]	pkt_em_sm[1]	ble_event_irq	tick_1us_p	basetimecnt[5]	basetimecnt[13]	radio_en [1]
DiagOut[6]	event_in_process	evtcntl_main_fsm[1]	pkt_em_sm[2]	ble_rx_irq	tick_prefetch_p	basetimecnt[6]	basetimecnt[14]	osc_en [1]
DiagOut[7]	tick_preftech_p	evtcntl_main_fsm[2]	pktcntl_rxtxn	ble_error_irq	tick_625us_p	basetimecnt[7]	basetimecnt[15]	wake_up
Connector name	**Value of DIAGCNTL**							
	0x08	**0x09**	**0x0A**	**0x0B**	**0x0C**	**0x0D**	**0x0E**	**0x0F**
DiagOut[0]	master1_gclken [1]	ahb2em_fsm[0]	em_inzone [2]	selected_if[0]	ctl_struct_ptr[0]	ctl_struct_ptr[8]	tx/rxdesc_ptr[0]	tx/rxdesc_ptr[8]
DiagOut[1]	master2_gclken	ahb2em_fsm[1]	businterface_em_req	selected_if[1]	ctl_struct_ptr[1]	ctl_struct_ptr[9]	tx/rxdesc_ptr[1]	tx/rxdesc_ptr[9]
DiagOut[2]	crypt_gclken	ahb2em_fsm[2]	businterface_em_write	selected_if[2]	ctl_struct_ptr[2]	ctl_struct_ptr[10]	tx/rxdesc_ptr[2]	tx/rxdesc_ptr[10]
DiagOut[3]	phase_match_p	em_sel	businterface_em_ack	em_inzone [2]	ctl_struct_ptr[3]	ctl_struct_ptr[11]	tx/rxdesc_ptr[3]	tx/rxdesc_ptr[11]
DiagOut[4]	clk_status	ahb2reg_fsm[0]	reg_inzone	em_write	ctl_struct_ptr[4]	ctl_struct_ptr[12]	tx/rxdesc_ptr[4]	tx/rxdesc_ptr[12]
DiagOut[5]	ble_slp_irq	ahb2reg_fsm[1]	reg_sel	em_cs	ctl_struct_ptr[5]	ctl_struct_ptr[13]	tx/rxdesc_ptr[5]	tx/rxdesc_ptr[13]
DiagOut[6]	ble_sw_irq	ahb2reg_fsm[2]	reg_write	em_ready	ctl_struct_ptr[6]	ctl_struct_ptr[14]	tx/rxdesc_ptr[6]	tx/rxdesc_ptr[14]
DiagOut[7]	master_soft_rst	reg_sel	reg_enable	bi_burst_ongoing [2]	ctl_struct_ptr[7]	em_gclken	tx/rxdesc_ptr[7]	pkcntl_rxtxn
Connector name	**Value of DIAGCNTL**							
	0x10	**0x11**	**0x12**	**0x13**	**0x14**	**0x15**	**0x16**	**0x17 [3]**
DiagOut[0]	tx/rxdata_ptr[0]	tx/rxdata_ptr[8]	rx_data	syncfound_pulse	rxint	evt_cntl_abort	fh_start_p	ccm_cs[0]
DiagOut[1]	tx/rxdata_ptr[1]	tx/rxdata_ptr[9]	rx_data_en	radcntl_busy	up_fetch_done_p	evt_cntl_done_p	whitelist_done	ccm_cs[1]
DiagOut[2]	tx/rxdata_ptr[2]	tx/rxdata_ptr[10]	rx_data_core	pktcntl_done_p	update_cs_req_p	evt_cntl_start_p	in_whitelist	ccm_cs[2]
DiagOut[3]	tx/rxdata_ptr[3]	tx/rxdata_ptr[11]	rx_data_core_en	pktcntl_rxtxn	fetch_cs_req_p	evt_cntl_fetch_p	whitelist_req_p	ccm_cs[3]
DiagOut[4]	tx/rxdata_ptr[4]	tx/rxdata_ptr[12]	tx_data	pktcntl_abort_p	update_rx_req_p	fco_enable	rxdesc_valid	micerr
DiagOut[5]	tx/rxdata_ptr[5]	tx/rxdata_ptr[13]	tx_data_en	pktcntl_start_p	fetch_rx_req_p	prestart_enable	txdesc_error	crypt_int
DiagOut[6]	tx/rxdata_ptr[6]	tx/rxdata_ptr[14]	tx_data_core	evt_cntl_done_p	update_tx_req_p	nextfco_enable	whitelist_error	crypto_start_p
DiagOut[7]	tx/rxdata_ptr[7]	tx/rxdata_ptr[15]	tx_data_core_en	evt_cntl_start_p	fetch_tx_req_p	event_abort_enable	ifs_underrun	swcrypt_error
Connector name	**Value of DIAGCNTL**							
	0x18	**0x19**	**0x1A**	**0x1B**	**0x1C**	**0x1D**	**0x1E [4]**	**0x1F**
DiagOut[0]	txccmpktcnt[0]	tx/rxlen[0]	rxtype[0]	initaddr_match	rxmd	whit_en	ble_pti[0]	ble_tx [4]
DiagOut[1]	txccmpktcnt[1]	tx/rxlen[1]	rxtype[1]	bdaddr_match	txmd	whit_init_pulse	ble_pti[1]	ble_rx [4]
DiagOut[2]	txccmpktcnt[2]	tx/rxlen[2]	rxtype[2]	rxtimeerr	cs_sn	crc_error	ble_pti[2]	ble_sync [4]
DiagOut[3]	txccmpktcnt[3]	tx/rxlen[3]	rxtype[3]	micerr	cs_nesn	crc_check	ble_pti[3]	bletxabort
DiagOut[4]	rxccmpktcnt[0]	tx/rxlen[4]	txtype[0]	crcerr	rxsn	crc_field	ble_rx	blerxabort
DiagOut[5]	rxccmpktcnt[1]	tx/rxlen[5]	txtype[1]	lenerr	rxnesn	crc_en	ble_tx	radcntl_rxen
DiagOut[6]	rxccmpktcnt[2]	tx/rxlen[6]	txtype[2]	typeerr	txsn	crc_init_pulse	wlan_rx	radcntl_txen
DiagOut[7]	rxccmpktcnt[3]	tx/rxlen[7]	txtype[3]	syncerr	txnesn	pktcntl_rxtxn	wlan_tx	event_in_process
Connector name	**Value of DIAGCNTL**							
	0x20	**0x21**	**0x22**	**0x23**	**0x24**	**0x25**	**0x26**	**0x27 [5]**
DiagOut[0]	ble_cscnt_irq	freq_select_fsm[0]	freq_word[0]	corr_window_size[0]	nb_corr_error[0]	rf_rx_en	radcntl_rxen	rp_em_fsm[0]
DiagOut[1]	ble_event_irq	freq_select_fsm[1]	freq_word[1]	corr_window_size[1]	nb_corr_error[1]	rf_tx_en	radcntl_txen	rp_em_fsm[1]
DiagOut[2]	ble_sw_irq	freq_select_fsm[2]	freq_word[2]	corr_window_size[2]	nb_corr_error[2]	rf_rx_data	miso	rp_em_fsm[2]
DiagOut[3]	ble_slp_irq	freq_select_fsm[3]	freq_word[3]	corr_window_size[3]	nb_corr_error[3]	rf_rx_data_valid	mosi	rp_em_fsm[3]
DiagOut[4]	ble_finetgtime_irq	hop_remap_dsb	freq_word[4]	corr_window_size[4]	nb_corr_error[4]	rf_tx_data	sclk	rp_em_fsm[4]
DiagOut[5]	ble_grosstgtime_irq	cs_fh_en	freq_word[5]	corr_window_size[5]	nb_corr_error[5]	rf_tx_data_valid	n_ss	rp_spi_fsm[0]
DiagOut[6]	ble_error_irq	fh_done_p	freq_word[6]	corr_window_size[6]	sync_window	sync_window	event_in_process	rp_spi_fsm[1]
DiagOut[7]	ble_crypt_irq	fh_start_p	freq_word[7]	corr_window	sync_found_pulse	syncfound_pulse_delayed	tick_625_us_p	rp_spi_fsm[2]
Connector name	**Value of DIAGCNTL**							
	0x28 [6]	**0x29 [7]**	**0x2A [7]**	**0x2B [7]**	**0x2C [7]**	**0x2D**	**0x2E**	**0x2F**
DiagOut[0]	extrc_em_fsm[0]	icytrx_em_fsm[0]	icytrx_spi_fsm[0]	radd_cnt[0]	radcntl_rxen	evt_schdl_mgmt[0]	ral_fsm[0]	ral_done_p
DiagOut[1]	extrc_em_fsm[1]	icytrx_em_fsm[1]	icytrx_spi_fsm[1]	radd_cnt[1]	half_full_flag	evt_schdl_mgmt[1]	ral_fsm[1]	ral_req_p
DiagOut[2]	extrc_em_fsm[2]	icytrx_em_fsm[2]	icytrx_spi_fsm[2]	radd_cnt[2]	full_flag	evt_cntl_done_p	ral_fsm[2]	peer_address_match
DiagOut[3]	extrc_em_fsm[3]	icytrx_em_fsm[3]	icytrx_spi_fsm[3]	radd_cnt[3]	empty_flag	evt_cntl_start_p	ral_fsm[3]	local_address_match
DiagOut[4]	extrc_txen	icytrx_em_fsm[4]	spigo	wadd_cnt[0]	rx_data_out	evt_cntl_fetch_p	ral_fsm[4]	rpa_renew_done_p
DiagOut[5]	extrc_rxen	fh_done_p	spicomp	wadd_cnt[1]	fifo_ren	evt_cntl_smooth_abort	in_whlist	rpa_renew_req_p
DiagOut[6]	fh_done_p	radcntl_txen	radcntl_busy	wadd_cnt[2]	rx_data_in	evt_cntl_immediate_abort	connected	rpa_resolution_done_p
DiagOut[7]	sync_p	radcntl_rxen	sync_p	wadd_cnt[3]	fifo_wen	empty_flag	entry_valid	rpa_resolution_req_p
Connector name	**Value of DIAGCNTL**							
	0x30	**0x31**	**0x32**	**0x33**	**0x34**	**0x35**	**0x36**	**0x37**
DiagOut[0]	privacy_error	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[1]	new_ralptr_valid	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[2]	peer_rpa_valid	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[3]	peer_irk_valid	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[4]	peer_rpa_renew	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[5]	local_rpa_valid	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[6]	local_irk_valid	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DiagOut[7]	local_rpa_renew	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Connector name	**Value of DIAGCNTL**							
	0x38	**0x39**	**0x3A**	**0x3B**	**0x3C**	**0x3D**	**0x3E**	**0x3F**
DiagOut[0]	N/A	N/A	N/A	N/A	swprofval[0]	swprofval[8]	swprofval[16]	swprofval[24]
DiagOut[1]	N/A	N/A	N/A	N/A	swprofval[1]	swprofval[9]	swprofval[17]	swprofval[25]
DiagOut[2]	N/A	N/A	N/A	N/A	swprofval[2]	swprofval[10]	swprofval[18]	swprofval[26]
DiagOut[3]	N/A	N/A	N/A	N/A	swprofval[3]	swprofval[11]	swprofval[19]	swprofval[27]
DiagOut[4]	N/A	N/A	N/A	N/A	swprofval[4]	swprofval[12]	swprofval[20]	swprofval[28]
DiagOut[5]	N/A	N/A	N/A	N/A	swprofval[5]	swprofval[13]	swprofval[21]	swprofval[29]
DiagOut[6]	N/A	N/A	N/A	N/A	swprofval[6]	swprofval[14]	swprofval[22]	swprofval[30]
DiagOut[7]	N/A	N/A	N/A	N/A	swprofval[7]	swprofval[15]	swprofval[23]	swprofval[31]

Table 2-28 – Diagnostic port observable signal groups

Note also that depending on the selected configuration, as defined per the parameters listed in Table 2-1, some of the signals described within Table 2-28 may be not present and are forced at '0'. As Example:

(1): Forced to '0' if *RW_BLE_TIMING_GEN_LP_EXTERNAL* is defined

(2): Forced to '0' if *RW_BLE_AHB_TO_EM_PATH* is not defined

(3): Forced to '0' if *RW_BLE_CRYPT_INST* is not defined

(4): Forced to '0' if *RW_BLE_WLAN_COEX_INST* is not defined

(5): Forced to '0' if *RW_BLE_RIPPLE_INST* is not defined

(6): Forced to '0' if *RW_BLE_EXTRC_INST* is not defined

(7): Forced to '0' if *RW_BLE_ICYTRX_INST* is not defined

Some of these signals listed above may be already present at the pins of the chip (e.g. TxData and RxData), hence this list contains only the basic signals required for validation purposes. Additional features can be added later to ease RW-BLE Core debug test.

3 Software control

3.1 General considerations

The RW-BLE Core is controlled by the RW-BLE Software to be able to perform all the tasks expected from a Bluetooth Low Energy device. The communication between the core and the software is depicted in Figure 3-1. The Exchange Memory contains buffers to be transmitted or received by the core. It also contains control parameters to manage the packet format and its features. Registers are used for static control parameters and real–time information. Interrupts are used to synchronize both worlds.

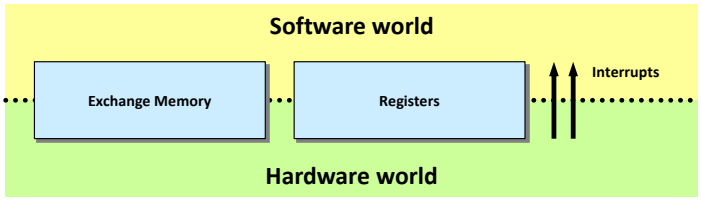

Figure 3-1 – Communication between Hardware and Software

3.2 Exchange table

The main point of communication between the RW-BLE Core and the RW-BLE Software is the Exchange Table. This table, shown in Figure 3-2, contains 15–bit pointers to the Control Structures, which are located in the Exchange Memory.

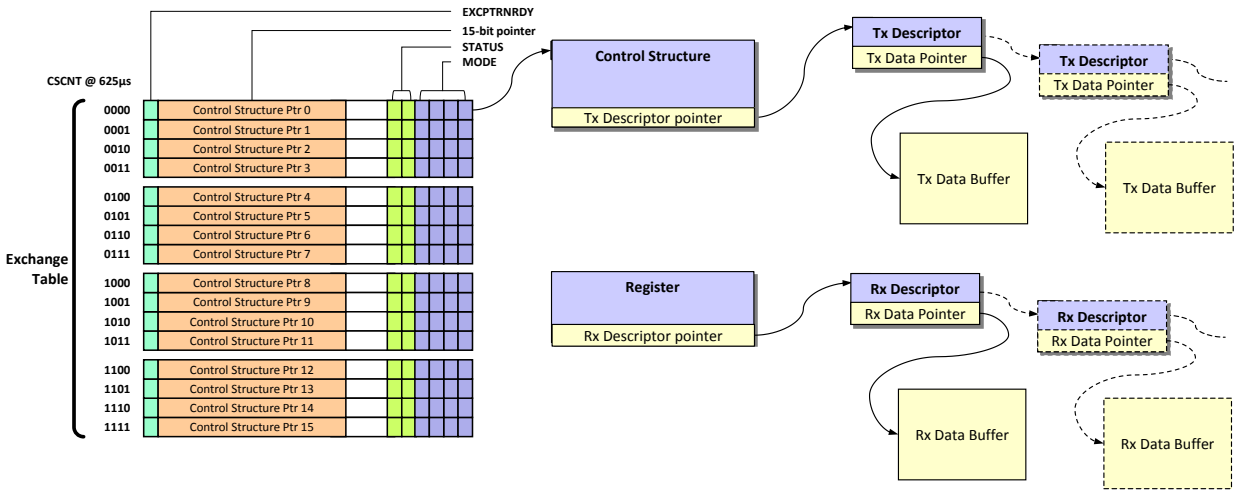

Figure 3-2 – Exchange table, Control Structure, Descriptors and Data Buffers

The Exchange Table has 16 entries, which correspond to either successive advertising, scanning or connection events anchor points. While the software prepares the future entries (i.e. anchor points), the hardware uses one to transmit / receive the packets.

Each Exchange table entry points to a unique control Structure that determines the Event type to be performed. Each control Structure points onto a Tx Descriptor chained list, while a register determines the Rx Descriptor location. Descriptors are chained lists in which the information / status of the current processed Tx/Rx packets are contained. Descriptors contain also the pointer to the next Descriptor of the chained list. Each descriptor has an associated data

pointer, giving the memory location of an associated Data Buffer. Each Data buffer has a maximum length of 37 bytes (See Figure 3-5, Figure 3-6 and Figure 3-7).

The position of the Exchange Table, Control Structure, and Tx/Rx Descriptors is limited to the first 32kBytes of the Exchange Memory, as pointers are 15–bit wide. Pointers to data buffers are 16-bit wide, leading to a maximum Exchange Memory size of 64kBytes.

Exchange Table entries are fetched at TIMGENCNTL-PREFETCH_TIME µs before the event starts. When the device is slave, then a margin must be taken into account regarding the internal 625µs clock drift between master and slave. This is done opening an Rx wide window around the ideal reception time. Then, a slave device must be able to resynchronize its internal timing to its master as explained in section 2.12.7.2. Figure 3-3 shows Control Structures pre-fetch mechanism, with a pre-fetch time that equals 312.5µs.

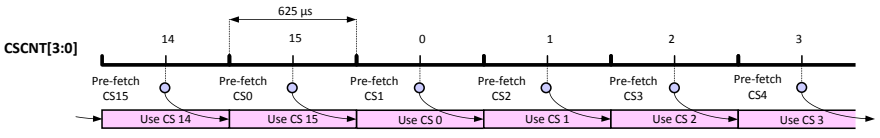

Figure 3-3 – Control Structure pre-fetch example with pre-fetch time set at 312.5µs

The software prepares the Control Structures in advance and the pointer in the Exchange Table is only setup when this operation is completed. The RW-BLE Core accesses the relevant pointers one time, just before each active event, so the software can prepare the later slots and access safely their pointers. However, care must be taken not to modify a Control Structure that is in use by the hardware.

The invalid entries in the Exchange Table are indicated by a null operating mode, or a null pointer (pointer to the 0x0000 address), or the status field being different from “00”. When the RW-BLE Core finds such an address in the table, it does not enable the system, and nothing is done. The 0x0000’H address has been chosen since the Exchange Memory starts at address 0x10000’H (when *RW_BLE_ADDRESS_WIDTH* = 16), which makes all lower addresses invalid.

Furthermore, ET-STATUS field is used to indicate the current status of the programmed pointer to the RW-BLE Software. It indicates if a pointer has to be processed, if a pointer is under process, if a pointer has been processed, or if a pointer has been simply discarded because of event overlapping.

Figure 3-4 lists the Exchange Table content

Address	Access		Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HW	SW																																	
00'H to 3C'H	R/W	R/W	EXTAB$_n$	EXTPTRNRDY$_n$	CSPTR$_n$[14:0]																									STATUS$_n$[1:0]		MODE$_n$[3:0]			

Figure 3-4 – Exchange Table content

Table 3-1 describes Exchange Table control field’s description.

Field Name	Value
MODE$_n$[3:0]	0x0: No mode selected, nothing to be performed 0x1: BR/EDR Mode 0x2: BLE Mode 0x3-0xF: Reserved for future use
STATUS$_n$[1:0]	00: Control Structure Pointer is ready for processing. 01: Control Structure Pointer has been read 10-11: Reserved
CSPTR$_n$[15:0]	Pointer to the associated Control Structure

EXCPTRNRDY$_n$	1: Control Structure pointer is not valid, software update ongoing. 0: Control Structure pointer is valid.

Table 3-1 – Exchange Table control field's description

3.3 Control Structure

The Control Structure, displayed in Figure 3-5, contains all the fields detailed in Table 3-2. These settings relate to advertising, scanning, or connection event, and give status information on the link. The width of Control Structures is set to 16 bits, but as it is implemented into the Exchange Memory, it is accessible with 8–, and 16–bit accesses for internal blocks, and with 8–, 16– and 32–bit accesses for the processor.

Address	Access		Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HW	SW																	
00'H	R	W	CXCNTL/FRCNTL	PTI[3:0]					TXBSY_EN	RXBSY_EN	DNABORT				FORMAT [4:0]				
02'H	R	W	FCNTOFFSET							FCNTOFFSET[9:0]									
04'H	R	W	LINKCNTL/LINKLBL	RXTHR[3:0]						TXCRYPT_EN	RXCRYPT_EN				LINKLBL[4:0]				
06'H	R	W	SYNCWL	SYNCWORD [15:0]															
08'H	R	W	SYNCWH	SYNCWORD [31:16]															
0A'H	R	W	CRCINIT0	CRCINIT[15:0]															
0C'H	R	W	CRCINIT1									CRCINIT[23:16]							
0E'H	R	W	FLTPOL/RALCNTL	FILTER_POLICY[7:0]													LOCAL_RPA_SEL	RAL_MODE	RAL_EN
10'H	R/W	R/I	HOPCNTL	FH_EN			HOPINT[4:0]								CH_IDX[5:0]				
12'H	R/W	R/W	TXRXCNTL	RXBUFF_FULL	LASTEMPTY	SN	NESN	RXBFMICERR				TXPWR [7:0]							
14'H	R	W	RXWINCNTL	RXWIDE		RXWINSZ[13:0]													
16'H	R	W	TXDESCPTR		TXDESCPTR[14:0]														
18'H	R	W	WINOFFSET	WINOFFSET[15:0]															
18'H	R	W	MINEVTIME	MINEVTIME[15:0]															
1A'H	R	W	MAXEVTIME	MAXEVTIME[15:0]															
1C'H	R	W	CONNINTERVAL	CONNINTERVAL[15:0]															
1C'H	R	W	CHMAP0	LLCHMAP[15:0]															
1E'H	R	W	CHMAP1	LLCHMAP[31:16]															
20'H	R	W	CHMAP2			NBCHGOOD[5:0]									LLCHMAP[36:32]				
22'H	R	W	RXMAXBUF									RXMAXBUFF[7:0]							
24'H	R	W	RXMAXTIME				RXMAXTIME[12:0]												
26'H	R	W	SK0	SK [15:0]															
26'H	R	W	ADV_BD_ADDR0	ADV_BD_ADDR[15:0]															
26'H	R	W	PEER_RALPTR	PEER_RALPTR[15:0]															
28'H	R	W	SK1	SK [31:16]															
28'H	R	W	ADV_BD_ADDR1	ADV_BD_ADDR[31:16]															
2A'H	R	W	SK2	SK [47:32]															
2A'H	R	W	ADV_BD_ADDR2	ADV_BD_ADDR[47:32]															
2C'H	R	W	SK3	SK [63:48]															
2C'H	R	W	ADV_BD_ADDR3																ADV_BD_ADDR[48]
2E'H	R	W	SK4	SK [79:64]															
30'H	R	W	SK5	SK [95:80]															
32'H	R	W	SK6	SK [111:96]															
34'H	R	W	SK7	SK [127:112]															
36'H	R	W	IV0	IV[15:0]															
38'H	R	W	IV1	IV[31:16]															
3A'H	R	W	IV2	IV[47:32]															
3C'H	R	W	IV3	IV[63:48]															
3E'H	W	R/I	TXWINOFFSET	TXWINOFFSET[15:0]															
3E'H	R/W	R/I	TXCCMPKTCNT0	TXCCMPKTCNT[15:0]															
40'H	R/W	R/I	TXCCMPKTCNT1	TXCCMPKTCNT[31:16]															
42'H	R/W	R/I	TXCCMPKTCNT2										TXCCMPKTCNT[38:32]						
44'H	R/W	R/I	RXCCMPKTCNT0	RXCCMPKTCNT[15:0]															
46'H	R/W	R/I	RXCCMPKTCNT1	RXCCMPKTCNT[31:16]															
48'H	R/W	R/I	RXCCMPKTCNT2										RXCCMPKTCNT[38:32]						
4A'H	W	R/I	BTCNTSYNC0	BTCNTRXSYNC[15:0]															
4C'H	W	R/I	BTCNTSYNC1						BTCNTRXSYNC[26:16]										
4E'H	W	R/I	FCNTSYNC	EVTRXOK						FCNTRXSYNC[9:0]									
50'H	W	R/I	TXRXDESCCNT	RXDESCCNT[7:0]								TXDESCCNT[7:0]							
52'H	R/W	R/I	DMPRIOCNTL	PRIOINCSTEP[2:0]			MINPRIO[4:0]					CONFLICT			CURRENTPRIO[4:0]				

Figure 3-5 – Control Structure content

When the RW-BLE Core writes a field of the Control Structure, all the unused bits are written to 0. The bits written by the software at unused locations are not read by the core and they can be of any value.

The access authorizations for each field are displayed in the Control Structure description; this indicates the direction of the information flow for the core (HW) or the software (SW). R means that only reading accesses are performed to this location, W means that the corresponding field is written, R/W means that the field is read and modified

afterwards and R/I means that the software must initialize this field before use but normally only reads the field's value.

Table 3-2 details the Control Structure content.

Field name	Description		Comments
PTI[3:0]	If set higher than calculated index found by the Event Controller, then CS-PTI is used to define *ble_pti[3:0]* output according to BLEMPRIO0/1 registers.		Used by the SW to override current register defined priority setting
TXBSY_EN	Enable *ble_tx* signal generation: 0: Do not generate *ble_tx* signal 1: Generate *ble_tx* signal		TX reporting policy (if any) is application driven.
RXBSY_EN	Enable *ble_rx* signal generation: 0: Do not generate *ble_rx* signal 1: Generate *ble_rx* signal		WLAN Coexistence provisions report Rx activity mainly for already established links. Some special link establishment scenarios could enable *ble_rx* generation as well.
DNABORT	Prevent Tx turn off: 0: Accept coexistence Tx turn off request 1: Ignore coexistence Tx turn off request		In WLAN coexistence scenarios, RW-BLE Core might be requested to suddenly turn off an ongoing Tx. If SW sets this flag, the core ignores the request. In some cases, HW has to write this bit.
FORMAT[4:0]	0000x: ***Do not use*** 00010: Master Connect 00011: Slave Connect 00100: Low Duty Cycle Advertiser 00101: High Duty Cycle Advertiser 0011x: ***Do not use*** 01000: Passive Scanner 01001: Active Scanner 0101x: ***Do not use*** 0110x: ***Do not use*** 01110: ***Do not use*** 01111: Initiator 10xxx: ***Do not use*** 110xx: ***Do not use*** 11100: Tx Test Mode 11101: Rx Test Mode 11110: Tx / Rx Test Mode 11111: ***Do not use***	– Master device Slave device Advertiser Advertiser – Scanner Scanner – – – Initiator – – RF Test Mode RF Test Mode RF Test Mode –	Advertising leads to a Slave device (when receiving a connect request) Initiator leads to a Master device (after connect request is sent)
FCNTOFFSET[9:0]	Time offset to apply onto the first Rx Packet. Valid for all devices, except during RF Test modes		Value is in µs
RXTHR[3:0]	Threshold value of Rx packet to receive before generating a *ble_rx_irq*		
TXCRYPT_EN	Enable encryption mode for Tx packet 0: Tx packet not encrypted 1: Tx packet encrypted		Applies only during connection event only
RXCRYPT_EN	Enable encryption mode for Rx packet 0: Rx packet not encrypted 1: Rx packet encrypted		Applies only during connection event only
LINKLBL[4:0]	Link Label. Used by the RW-BLE Software to label a Control Structure. This value is reported in Rx Descriptors when a packet is received		
SYNCWORD[31:0]	Synchronization Word used for Bluetooth Low Energy packet detection		
CRCINIT[23:0]	CRC Initialization value. Defined as per the connection request packet parameter.		Note that for advertising and broadcast packets, the CRC initialization equals to 0x555555.

FILTER_POLICY[7:0]	Bits [1:0] = 00: Allow scan request and connection request from any device Bits [1:0] = 01: Allow scan request from device in the white list only, and connection request from any device Bits [1:0] = 10: Allow scan request from any device, and connection request from device in the white list only Bits [1:0] = 11: Allow scan request and connection request from device in the white list only. Bits [7:2]: Reserved	Advertising Filtering Policy	Device Filter Policy, depends on CS-FORMAT values
	Bit [1:0] = 00: Accepts all advertising packets. Directed advertising packets which are not addressed for this device are ignored Bit [1:0] = 01: Accept advertising packets from device in the white list only. Directed advertising packets which are not addressed for this device are ignored Bit [1:0] = 10: Accepts all undirected advertising packets. Directed advertising packets which are not addressed for this device are ignored. All Directed advertising packet with RPA are reported (even if *InitA* is not resolved). Bit [1:0] = 11: Accept advertising packets from device in the white list only. Directed advertising packets which are not addressed for this device are ignored. All Directed advertising packets with RPA are reported (even if *InitA* is not resolved). Bit [7:2]: Reserved	Scanning Filtering Policy	
	Bit [0] = 0: White List is not used to determine which advertiser to connect, but the specific CS-ADV_BD_ADDR[47:0] or selected specific RAL structure Bit [0] = 1: White List is used to determine which advertiser to connect Bit [7:1]: Reserved	Initiator Filtering Policy	
RAL_EN	Enable Resolvable Address List engine 0: RAL disabled 1: RAL enabled		Used when the device is in Enhanced Privacy mode
RAL_MODE	Valid only if RAL_EN = 1. Resolvable Address List engine operating mode. Valid for Initiating and Advertising modes only. Meaningless in Scanning mode. 0: Use directly CS-ADV_BD_ADDR 1: Use CS-PEER_RALPTR as RAL entry		In Advertising, used for 1st transmission only
LOCAL_RPA_SEL	Valid only if RAL_EN = 1 and CS-RAL_MODE = 1 0: Use PRIV_NPUB and BDADDR[47:0] from registers 1: Use RAL-RAL_LOCAL_RPA if RAL-LOCAL_RPA_VALID = 1, else use PRIV_NPUB and BDADDR[47:0] from registers		Used on first Transmission on any Advertising events to determine which device address to be used
FH_EN	Enable Frequency Hopping mechanism 0: Hopping disabled 1: Hopping enabled		
HOPINT[4:0]	Hopping Interval. This value must be in [5:16] range.		This field is used only when the device is in connected mode.
CH_IDX[5:0]	Channel Index provided by the SW, used for frequency hopping calculation.		At the beginning of the event, this field corresponds to the last unmapped

	- When CS-FH_EN = 1, the hopping algorithm is enabled - When CS-FH_EN = 0, the hopping algorithm is disabled, and CS-CH_IDX is used "as is" by the RW-BLE Core.	channel index. It is then used by the frequency selection lock for frequency hopping calculation. At the end of the event, it is updated with the unmapped channel index. On first connection event, the value must equal 0x0.
RXBUFF_FULL	Indicates whether any event has been closed due to full Rx buffer. In case this bit is set on CS pre-fetch, then there are three cases: 1- The device is master, then fakes reception in order to get NESN bit and determine whether ACK/NAK is required 2- The device is slave, then fakes reception in order to get NESN bit and determine whether ACK/NAK is required, but transmit same SN hence NAK-ing the received packet 3- The device is in any other modes and simply discards the event.	RW-BLE Core updates this field at the end of the event if there are no more Rx Descriptors available
LASTEMPTY	On pre-fetch, indicates the device has to send an empty packet till its acknowledgment, instead of directly using current Tx Descriptor Pointer and send a new Tx packet.	This bit is updated at the end of an event. It is cleared on acknowledgement and a valid Tx Descriptor is available (Slave device), and is set if the last sent packet is a null length packet without a valid ACK (Master device).
SN	Indicates Connection Event Tx Packet initial Sequence Number bit This bit is also used to report the last SN value used by the RW-BLE Core	This field is used when RWBLECNTL-SN_DSB = 0, else TxDESC-SN is used This bit is used "as-is" in case of master device This bit is used as "latest SN" in case of slave device.
NESN	Indicates Connection Event Tx Packet initial Next Sequence Number bit This bit is also used to report the last NESN value used by the RW-BLE Core	This field is used when RWBLECNTL-NESN_DSB = 0, else TxDESC-NESN is used. This bit is used "as-is" in case of master device This bit is used as "latest SN" in case of slave device.
RXBFMICERR	0: Received MIC was correct. 1: Received MIC was not correct	Valid for encrypted link layer connection only, and when CS-RXBUFF_FULL is set
TXPWR[7:0]	Selects the power level to be used by the radio (when supported by the selected radio module).	The exact value depends on the actual radio implementation.
RXWIDE	0: Normal reception window (count in µs) 1: Wide–open mode (count in 625 µs multiple value).	
RXWINSZ[13:0]	Size of the Rx window in bits (normal mode), or in number of 625µs base time reference (wide–open mode).	Maximum size is 16384 x 625µs in wide–open mode (i.e. 10.24s). 0 is not a valid value.
TXDESCPTR[14:0]	Pointer to the Tx Descriptor.	In case of Link Layer Channel, a null pointer allows the devices to send null length packet only In case of device set as Advertiser, Active Scanner or Initiator, a null pointer generates a *ble_error_irq* and set ERRORTYPESTAT.CSTXPTR_ERROR.
WINOFFSET[15:0]	Transmit Window Offset Initialization Value. Used by Event Controller to calculates optimal first connection time. Provided to future slave device through CONNECT_REQ packet.	Applicable only when device is Initiator

	Value is a multiple of 1.250ms.	
ADV_BD_ADDR[48:0]	When the device is set as Initiator with Device filtering Policy set to 0x0, this field must be used in place of the White List	Privacy indicator is set on bit 48, and Advertiser BD_ADD is set on bit [47:0]
PEER_RALPTR[15:0]	When the device is set as Initiator and in enhanced Privacy mode, this field indicates the Peer device's RPA Structure pointer to be used.	Privacy indicator is set to 1.
MINEVTIME[15:0]	Minimum Event Time Counted in number of 625µs base time reference Valid for Master device only	Minimum duration of a connection event. Whenever master's MD bit equals 0, it forces the master to maintain the event at least during this minimum time.
MAXEVTIME[15:0]	Maximum Event Time. Counted in number of 625µs base time reference Valid for any event. If CS-MAXEVTIME equals 0 then the event has no time limitation.	Maximum duration of an event. During connection event, whenever master's MD bit equals 1, it forces a master device to stop the event the maximum event time is reached. In case of slave device, it forces to stop the event during transmission.
CONNINTERVAL[15:0]	Used for Transmit Window Offset calculation to be sent through CONNECT_REQ packet. Corresponds to the required connection interval of the BLE Link. Value is a multiple of 1.250ms	Applicable only when device is Initiator
LLCHMAP[36:0]	Logical Link Channel Map lower word, defined as per the channel connection settings. If LLCHMAP[i] equals: 0: Do not use data channel i. 1: Use data channel i.	Valid for Link Layer Connection only
NBCHGOOD[5:0]	Indicates number of good channels in data channel map 0 is not a valid value: at least 1 channel must be available.	Valid for Link Layer Connection only
RXMAXBUFF[7:0]	Defines the maximum size (in bytes) of the allocated Rx Buffers for Master/Slave connections	If equals 0x00 means BLE v4.1 or earlier version support
RXMAXTIME[12:0]	Defines the reception maximum time (in µs) for Master/Slave connections	If equals 0x00 means no Rx time constraints
SK[127:0]	AES-CCM Session Key	
IV[63:0]	AES-CCM Initialization Vector	
TXWINOFFSET[15:0]	Transmit Window Offset value that is send by the CONNECT_REQ packet.	Applicable only when device is Initiator
TXCCMPKTCNT[38:0]	Tx CCM Packet Counter	Can be also used as Tx packet counter during RF Tx Test mode
RXCCMPKTCNT[38:0]	Rx CCM Packet Counter	Can be also used as correct Rx packet counter during RF Rx Test mode
BTCNTRXSYNC[26:0]	Reports BASETIMECNT value on: first Rx Packet synchronization time when use of Advertising channels last Rx Packet synchronization when use of Link Layer Channels	Value is a multiple of 625µs
FCNTRXSYNC[9:0]	Reports FINETIMECNT value on: first Rx Packet synchronization time when use of Advertising channels last Rx Packet synchronization when use of Link Layer Channels	Note that FINETIMECNT is a down-counter and in order to restore proper value 624-FINETIMECNT calculation may be required. Value is in µs.
EVTRXOK	Indicates First Reception in the event is valid. Used to validate BTCNTRXSYNC and FCTRXSYNC value.	
RXDESCCNT[7:0]	Indicates the number of Rx Descriptor "consumed" (i.e. with RxDESC-RXDONE bit set	
TXDESCCNT[7:0]	Indicates the number of Tx Descriptor "transmitted and "acknowledged" (i.e. with TxDESC-TXDONE bit set, valid only for Master, Slave and Initiator devices)	

PRIOINCSTEP[2:0]	Priority Increment to apply to CS-CURRENTPRIO[4:0] when a conflict is detected	Used only if RW_DM_SUPPORT is set. Please refer to section 2.12.3 and 3.6 for further details.
MINPRIO[4:0]	Minimum Priority level, used to set CS-CURRENTPRIO[4:0] when there's no conflict	Used only if RW_DM_SUPPORT is set. Please refer to section 2.12.3 and 3.6 for further details.
CONFLICT	Indicates whether a conflict with a BR/EDR frame happens This bit is set when the current event has been stopped by a higher BR/EDR frame priority. (See section for details)	Used only if RW_DM_SUPPORT is set. Please refer to section 2.12.3 and 3.6 for further details.
CURRENTPRIO[4:0]	Current event priority level. Updated by the RW-BLE Core either a conflict is detected, or at each end of event.	Used only if RW_DM_SUPPORT is set. Please refer to section 2.12.3 and 3.6 for further details.

Table 3-2 – Control Structure content description

3.4 Descriptors and Payload Buffers

3.4.1 Tx Descriptor

The Tx Descriptor, displayed in Figure 3-6, contains all the fields detailed in Table 3-3. These settings relate to either an advertising event, or a connection event, and give processing status information and acknowledgement status of the current Tx packet. The width of Tx Descriptor is set to 16 bits.

When the RW-BLE Core writes a field in the Tx Descriptor, all the unused bits are written to 0. The bits written by the software at unused locations are not read by the core and they can be of any value. The access authorizations for each field are displayed in the Tx Descriptor description; this indicates the direction of the information flow for the core (HW) or the software (SW). R means that only reading accesses are performed to this location, W means that the corresponding field is written, R/W means that the field is read and modified afterwards and R/I means that the software must initialize this field before use but normally only reads the field's value.

Address	Access		Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HW	SW																	
00'H	R/W	R/I	TXPTR	TXDONE	NEXTPTR[14:0]														
02'H	R	W	TXPHCE	TXLEN[7:0]											TXMD	TXSN	TXNESN	TXLLLID[1:0]	
02'H	R	W	TXPHADV	TXADVLEN[7:0]								TXRXADD	TXTXADD			TXTYPE0[3:0]			
04'H	R	W	TXDATAPTR	TXDATAPTR[15:0]															

Figure 3-6 – Tx Descriptor

Table 3-3 details the Tx Descriptor content.

Field name	Description	Comments
TXDONE	Transmission Done indicator (valid only for Master, Slave and Initiator devices) 0: Tx Packet transmission not done 1: Tx Packet transmission done and successfully acknowledged.	This bit is reset by the RW-BLE Software in order to indicate the HW the packet has to be sent. HW set this bit once the packet has been transmitted and successfully acknowledged.
NEXTPTR[14:0]	Next Tx Descriptor Pointer of the Tx data buffers chained list	Link Layer Connection use case: A pointer set to 0x0000 indicates the end of the chained list: A slave device sends null length packet till the end of the event A master device either sends null length packet or stop the event according to CS-MINEVTIME

		Advertising channel use case: A pointer set to 0x0000 indicates the end of the chained list, the event can be closed.
TXLEN[7:0]	Transmit Packet payload length	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
TXMD	Value of the More Data bit value to transmit	The use of this field depends on CS–FORMAT, and applies in case of connection event only. This bit is used if RWBLECNTL-MD_DSB is set
TXSN	Value of the Current Sequence Number bit to transmit	The use of this field depends on CS–FORMAT, and applies in case of connection event only. This bit is used if RWBLECNTL-SN_DSB is set
TXNESN	Value of the transmitted Next Sequence Number bit	The use of this field depends on CS–FORMAT, and applies in case of connection event only. This bit is used if RWBLECNTL-NESN_DSB is set
TXLLID[1:0]	00: Reserved 01: Continuation of a Data packet 10: Start of a Data packet 11: Control Packet	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
TXADVLEN[7:0]	Tx Advertising events packet length	The use of this field depends on CS–FORMAT, and applies in case of advertising events only.
TXRXADD	Returned BD_ADDR private/public indicator 0: Public 1: Private	The use of this field depends on CS–FORMAT, and applies in case of advertising events only.
TXTXADD	Sent BD_ADDR private/public indicator 0: Public 1: Private	The use of this field depends on CS–FORMAT, and applies in case of advertising events, RF Test modes only.
TXTYPE[3:0]	0x0: Connectable undirected advertising packet type 0x1: Connectable directed advertising packet type 0x2: Non-connectable advertising packet type 0x3: Scan Request packet type 0x4: Scan Response packet type 0x5: Connect Request packet type 0x6: Scannable undirected advertising packet type 0x7–0xF: Reserved for future use	The use of this field depends on CS–FORMAT, and applies in case of advertising event only.
TXDATAPTR[15:0]	Address of the Memory Space used for data to be transmitted	The use of this fields depends on CS–FORMAT, as well as relevant TXADVLEN or TXLEN field value

Table 3-3 – Tx Descriptor content description

3.4.2 Rx Descriptor

The Rx Descriptor, displayed in Figure 3-7, contains all the fields detailed in Table 3-4. These values relate to either an advertising event, or a connection event, and give processing status information and acknowledgement status of the current Rx packet. The width of Rx Descriptor is set to 16 bits.

When the RW-BLE Core writes a field in the Rx Descriptor, all the unused bits are written to 0. The bits written by the software at unused locations are not read by the core and they can be of any value. The access authorizations for

each field are displayed in the Rx Descriptor description; this indicates the direction of the information flow for the core (HW) or the software (SW). R means that only reading accesses are performed to this location, W means that the corresponding field is written, R/W means that the field is read and modified afterwards and R/I means that the software must initialize this field before use but normally only reads the field's value.

Address	Access		Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HW	SW																	
00'H	R/W	R/I	RXPTR	RXDONE	NEXTPTR[14:0]														
02'H	W	R	RXSTATCE	RXLINKLBL[4:0]						PRIV_ERROR	RXTIMEERR	BDADDR_MATCH	NESNERR	SNERR	MICERR	CRCERR	LENERR	TYPEERR	SYNCERR
04'H	W	R	RXPHCE	RXLEN[7:0]											RXMD	RXSN	RXNESN	RXLLLID[1:0]	
04'H	W	R	RXPHADV	RXADVLEN[7:0]								RXRXADD	RXTXADD			RXTYPE[3:0]			
06'H	W	R	RXCHASS			USED_CH_IDX[5:0]						RXRSSI[7:0]							
08'H	R	W	RXDATAPTR	RXDATAPTR[15:0]															
0A'H	W	R	RXRALPTR	RXRALPTR[15:0]															

Figure 3-7 – Rx Descriptor

Table 3-4 details the Rx Descriptor content.

Field name	Description	Comments
RXDONE	Reception Done indicator 0: Rx Packet not received, Rx buffer is free for use 1: Rx Packet received, Rx buffer need to be freed	This bit is reset by the RW-BLE Software in order to indicate the HW an Rx packet can be received in this buffer. HW set this bit once the packet is received.
NEXTPTR[14:0]	Next Rx Descriptor Pointer of the Rx data buffers chained list	Link Layer Connection use case: when null, next Rx packet is not saved in EM, and all incoming Rx packets are NAK-ed in order to force peer device to retransmit till end of the event. Advertising Channels use case: when null, the event is closed.
RXLINKLBL[4:0]	Control Structure label (i.e CS-LINKLBL[4:0]) is copied here on each packet reception	
PRIV_ERROR	Valid only if CS-RAL_EN=1 0: No error 1: Privacy error detected on the received packet	
RXTIMEERR	0: Rx Time was below than CS-RXMAXTIME field. 1: Rx Time was greater than CS-RXMAXTIME field	Valid for Data Channels only. Processed over an effective reception (i.e in between Access Code detection to last CRC bit, regardless of Rx turn-off time
BDADDR_MATCH	0: Returned Bluetooth Device Address does not match with BDADDR register 1: Returned Bluetooth Device Address match with BDADDR register	Valid for Advertiser, active Scanner, and Initiator devices, according to the device filtering policy
CRCERR	0: CRC was correct. 1: CRC was not correct.	
NESNERR	0: Next Sequence Number was correct. 1: Next Sequence Number was not correct.	Valid for link layer connection only
SNERR	0: Current Sequence Number was correct. 1: Current Sequence Number was not correct.	Valid for link layer connection only
MICERR	0: MIC was correct. 1: MIC was not correct.	Valid for encrypted link layer connection only
LENERR	0: Length in the range allowed by the event. 1: Length bigger than maximum allowed by the event.	Valid only for link layer connection when CS-RXMAXBUFF is different from 0x00, and advertising channels
TYPEERR	0: Packet is allowed by the event 1: Packet is not allowed by the event	Valid only for Advertising channels

SYNCERR	0: Sync Word has been detected. 1: Sync Word has not been correct.	
RXLEN[7:0]	Received Packet payload length	The use of this field depends on CS–FORMAT, and applies in case of connection events only. Applies for data/control packets only
RXRXADD	Returned BD_ADDR private/public indicator 0: Public 1: Private	The use of this field depends on CS–FORMAT, and applies in case of advertising events only.
RXTXADD	Sent BD_ADDR private/public indicator 0: Public 1: Private	The use of this field depends on CS–FORMAT, and applies in case of advertising events only.
RXMD	Value of the received More Data bit value	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
RXSN	Value of the received Current Sequence Number bit	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
RXNESN	Value of the received Next Sequence Number bit	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
RXLLID[1:0]	00: Reserved 01: Continuation of a Data packet 10: Start of a Data packet 11: Control Packet	The use of this field depends on CS–FORMAT, and applies in case of connection event only.
RXADVLEN[7:0]	Rx Advertising events packet length	The use of this field depends on CS–FORMAT, and applies in case of advertising events only.
RXTXADD	Received Advertiser / Scanner / Initiator Address privacy indicator 0: Public address 1: Private address	The use of this field depends on CS–FORMAT, and applies in case of advertising event only.
RXTYPE[3:0]	0x0: Connectable undirected advertising packet type 0x1: Connectable directed advertising packet type 0x2: Non-connectable advertising packet type 0x3: Scan Request packet type 0x4: Scan Response packet type 0x5: Connect Request packet type 0x6: Scannable undirected advertising packet type 0x7–0xF: Reserved for future use	The update of this field depends on CS–FORMAT, and applies in case of advertising events only.
USED_CH_IDX[5:0]	Used channel Index in which the RxDESC-RXRSSI value has been captured	
RXRSSI[7:0]	Indicates the received power level (when supported by the selected radio module).	The exact value depends on the actual radio implementation.
RXDATAPTR[15:0]	Address of the Memory Space used for data to be received	The use of this fields depends on CS–FORMAT, as well as relevant received RXADVLEN or RXLEN field value
RXRALPTR[15:0]	Address of the RAL Structure for which RPA resolution is successful.	

Table 3-4 – Rx Descriptor content description

3.4.3 Validity in reception

After reception of a packet, certain status flags of the Rx Descriptor are not meaningful and must be ignored.

Table 3-5 summarizes the link layer connection cases after reception of a packet.

Conditions	Valid fields
Sync not found	SYNCERR
Sync found / Length Error	SYNCERR, LENERR[(1)]
Sync found / Correct Length	SYNCERR, LENERR[(1)], NESNERR, SNERR, CRCERR, RXBFMICERR/MICERR, RXTIMERR[(2)], RXLLID, RXNESN, RXSN, RXMD, RXLEN,

Table 3-5 – Validity of Control Structure's reception fields – Link Layer Packet

[(1)]: LENERR is set only if received length is greater than CS-RXMAXBUF and if CS-RXMAXBUF is not null.

[(2)]: RXTIMEERR is set if received packet duration is greater than CS-RXMAXTIME. Note this stops the event under processing.

Table 3-6 summarizes the advertising channel cases after reception of a packet.

Conditions	Valid fields
Sync not found	SYNCERR
Sync found / Packet type error	SYNCERR, TYPEERR
Sync found / Correct Packet type / Length Error	SYNCERR, TYPEERR, LENERR
Sync found / Correct Packet type / Correct Length	SYNCERR, TYPEERR, LENERR, CRCERR, RXTYPE, TXADD, RXADVLEN, BDADDR_MATCH

Table 3-6 – Validity of Control Structure's reception fields – Advertising Packet

Note that in case of Advertising Channels, RXTIMEERR is meaningless.

3.4.4 Payload Buffers

3.4.4.1 General Considerations

The Control Structure contains the pointers to Tx Descriptor only. Rx Descriptor is set by ET_CURRENTRXDESCPTR-CURRENTRXDESCPTR register. ET_CURRENTRXDESCPTR-CURRENTRXDESCPTR register field is updated at the end of each event in order to be ready on next programmed event. As Rx Descriptors are "ring" chained buffers, the RW-BLE Software is in charge of freeing the used Rx Descriptors before next event starts, or before it is reused within the same event. This action is controlled by the use of CS-RXTHR and *ble_rx_irq* interrupt (please refer to section 3.3 and section 2.10 for details).

One Payload Buffer (See Figure 3-2) is associated to a given descriptor, and one buffer is dedicated either to transmission or to reception. Tx and Rx Descriptors are defining the starting point of the chained list of Tx/Rx packets to be sent during an event, and embeds associated data buffer pointer defining the payload buffer content, hence the capability of handling several Tx/Rx buffer during an event.

Each of these buffers can contain a payload (as a null length packet does not carry data). It is used for all types of packets containing data to be transmitted or received. The Descriptors contains Packet Header information (<Tx/RxDESC>-<TX/RX>PHADV for advertising events, or <Tx/Rx>DESC-<TX/RX>PHCE for connection events), data buffers location, and several processing status about the current Descriptor in use). The Synchronization Word is not stored either, but is driven from the settings of CS–SYNCWL/H.

Payload buffers are used for Link Layer data and control packets, and for advertising packets. The software must ensure that the length of the payload specified in the Tx Descriptor is consistent with the amount of data present in the Tx buffer, and that the Rx buffer is large enough to accept receiving the largest possible packet negotiated with the other Bluetooth Low Energy unit.

3.4.4.2 Tx Descriptor handling

Allowing next Tx Descriptor usage depends on the acknowledgement of the previous transmitted data. It is done according to NESN bit value of last received packet. The number of Tx packets sent during each connection event may vary according to MD bit.

Additionally, the RW-BLE Core is able to manage null length packet generation according to both CS-LASTEMPTY value and Tx Descriptor validity (i.e verifying CS-TXDESCPTR, TxDESC-NEXPTR and TxDESC-TXDONE field).

Figure 3-8 shows Tx buffer selection mechanism, and the NESN check mechanism.

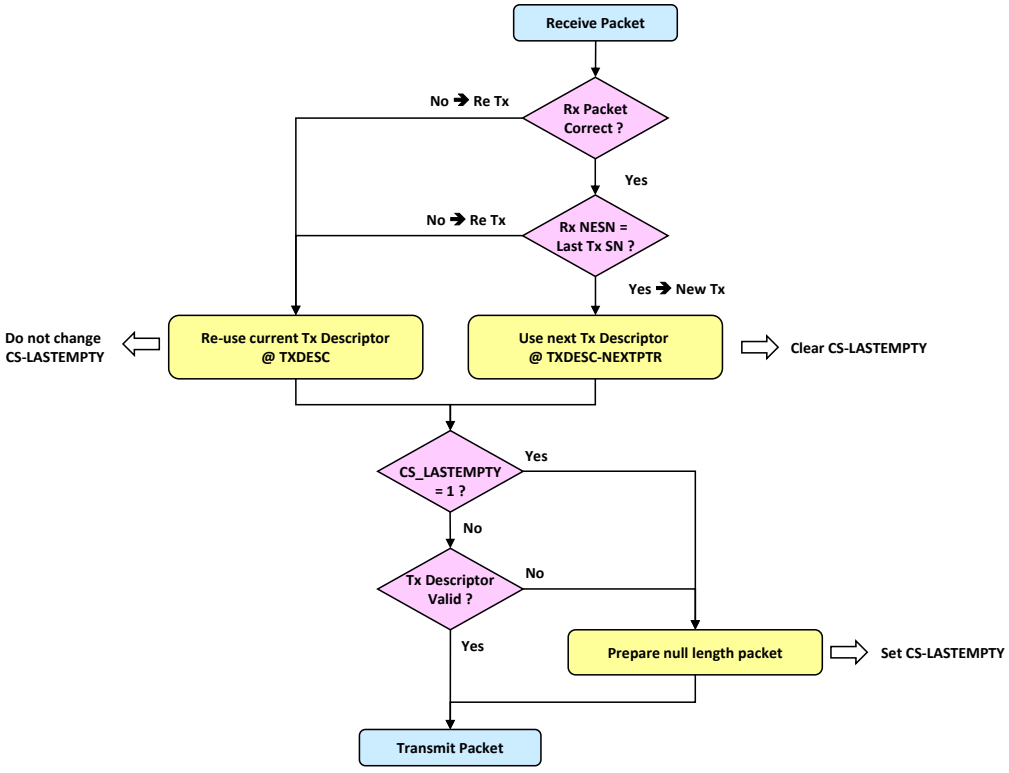

Figure 3-8 – Selection of Descriptors in Tx

Refer to section 3.8.2 for further details regarding SN, NESN and MD bit fields.

3.4.4.3 Rx Descriptor handling

A new Rx Descriptor is used each time a packet is received, whenever the Access Address is detected.

Figure 3-9 shows the Rx buffer mechanism

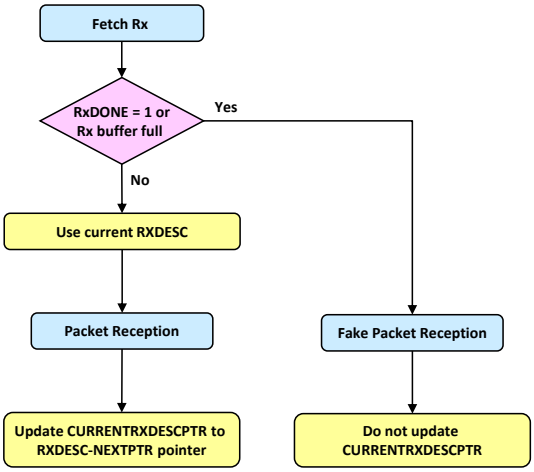

Figure 3-9 – Selection of data buffers in RX

Refer to section 3.8.2 for further details regarding SN, NESN and MD bit fields.

Packet reception information is available from the Radio controller upon correlation, and is sent to the Event Controller block, which uses it for its processing, and later dumps it into the Control Structure and Rx Descriptors among other status information. In case of synchronization detection failure during reception, the event is closed. In case of two consecutive CRC errors detection (that would lead onto retransmission request), then the event is closed. In case of received length error (i.e. bigger than the maximum length allowed by the received packet type), then the event is closed as well, and a CRC error is generated. In case of MIC error, the device must go to standby mode, leading to disconnection.
Additionally, in case the RW-BLE Software cannot free the current Rx Descriptor to be used on time (i.e RxDESC-RXDONE read at logic '1' on pre-fetch indicating the Rx Buffer is used), whenever the device is in slave or master role, then the RW-BLE Core fakes reception, sets CS-RXBUFF_FULL, and replies automatically with a Data Packet having same NESN bit (indicating a retransmission is required). This mechanism enables automatic flow control at Link Layer level.

Additionally, in order to ease connection maintenance, a slave device must set CS-EVTRXOK, and update both CS-FCNTRXSYNC and CS-BTCNTRXSYNC (if Access Address is detected) of its first received packet.

3.4.5 Data Flow

The section hereafter explains how payload data are exchanged from RW-BLE Software, to the Exchange Memory Data Buffers and then to the RF, and vice-versa.

3.4.5.1 Transmission

Figure 3-10 details transmission payload data flow.

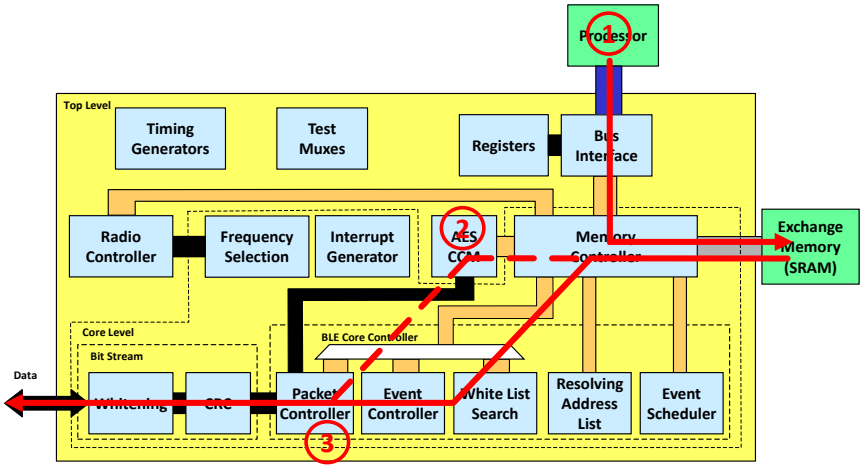

Figure 3-10 – Transmission Data Flow

In transmission the following mechanism applies:

1. The RW-BLE Software prepares the Tx Descriptors and the Tx Data Buffers
2. If the Link is encrypted, AES-CCM performs the Access to the Tx Data Buffer before providing encrypted data to the Packet Controller
3. If the Link is not encrypted, the packet controller reads directly the Tx Data Buffers and sent it over the Bit Stream path to the Radio

3.4.5.2 Reception

Figure 3-11 details reception payload data flow

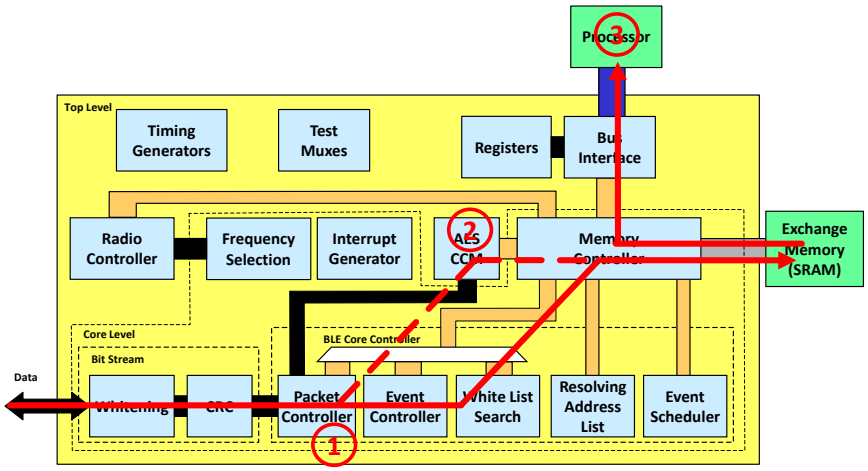

Figure 3-11 – Reception Data Flow

In reception the following mechanism applies:

1. The packet controller receives the data from the radio through the Bit Stream. If the link is not encrypted, Rx Data are written directly to the Rx Data Buffers
2. If the Link is encrypted, the Packet Controller provides Rx Data to the AES-CCM that performs decryption prior to write decrypted data to the Rx Data Buffers.
3. The RW-BLE Software reads data of the Rx Data Buffers and perform Link Layer software task

3.5 Event Programming

Events (advertising / connection / scanning) are programmed by the means of the Control Structure. Depending on the setting of CS-FORMAT, the event is composed differently. Each different event format is detailed in this section. In section 3.3, the description of CS-FORMAT field also gives an indication of the type of device associated with each format.

The RW-BLE Software is responsible of advertising events and connection events programming. The RW-BLE Digital Core's Event Scheduler generates an interrupt on each end of event, as well as when an event is not processed: each exchange table is read and taken into account only after the current event in process is closed (Please refer to section 2.9.3.2 and 2.12.2 for details).

Figure 3-12 shows exchange table pre-fetch for a master device, and how the Core Controller behaves during each connection events. Please note that pre-fetch time equals 312.5µs in this figure.

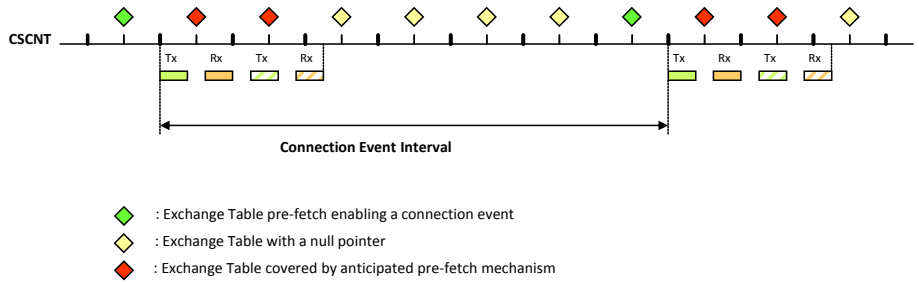

Figure 3-12 – Connection event programming and exchange table pre-fetch / Master device example

Figure 3-13 shows exchange table pre-fetch for a slave device, and how the Core Controller behaves during each connection events. Please note that pre-fetch time equals 312.5µs in this figure.

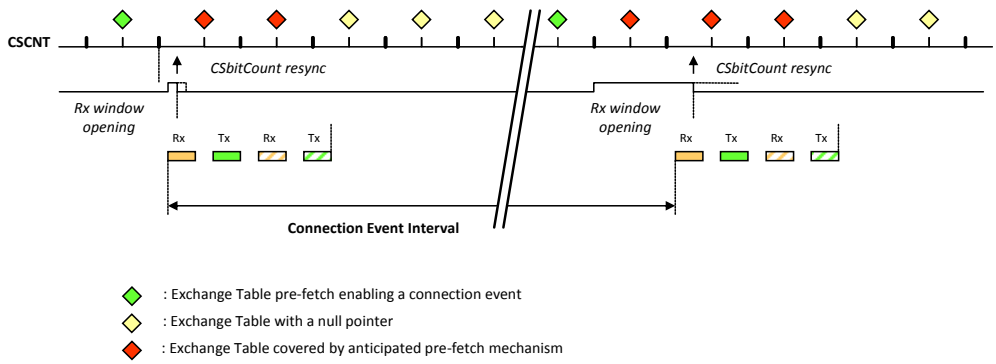

Figure 3-13 – Connection event programming and exchange table pre-fetch / Slave device example

The size of the synchronization window is indicated by the value of CS–RXWINSZ (in bits) when CS-RXWIDE indicates a normal window, and when the device is in Rx mode. In this case the synchronization word must be tracked for detection around 40µs (i.e. preamble and synchronization word duration) after the potential start of the incoming Rx packet. The size is given in 625µs base time reference (by CS-RXWINSZ) when CS-RXWIDE indicates a wide–open window. In this case, the connection event interval is greater than 3.125s.

In Figure 3-13, two examples of Rx window opening are given. On the left (first connection event), the Rx window is programmed for a connection event interval lower than 3.125s. On the right (second connection event), the Rx window is programmed for a connection event interval greater than 3.125s. Depending on the CS-FORMAT, advertising events can be also selected. Further details are provided onto advertising event scheduling in section 3.7.3.

3.6 Dual Mode Support

In RW-BLE Core, some provisions are taken for Dual Mode support. The basic rules of this priority mechanism are the following:

1. On first event programming (i.e. Link Creation, Scan event starting, etc…), the RW-BLE Software set CS-DMPRIOCNTL fields in the Control Structure. The values set depend on the parameters in use (i.e. Connection Interval, Scan Interval, etc...)
2. CS-PRIOINCSTEP[2:0] must never be set at 0x0. CS-MINPRIO[4:0] must never be set to 0x00 or 0x1F (minimum or maximum values are excluded).
3. On each valid Exchange Table entry, the CS-CURRENTPRIO[4:0] is read and provided to an external block.
4. On an empty Exchange Table entry (i.e. Pointer programmed @0x00), the priority provided for arbitration must be 0x00
5. In case of simultaneous BR/EDR frame and BLE event programming, current BR/EDR and current BLE priorities are checked: lowest priority frame/event is aborted, highest priority frame/event is granted.
6. If BR/EDR frame and BLE event priority levels are equivalent, other criteria are used for decision, but are out of the scope of this document
7. If an event is aborted, even after being started, then CS-CURRENTPRIO[4:0] is incremented by CS-PRIOINCSTEP[2:0], and CS-CONFLICT bit is set.
8. If an event is granted, CS-CURRENTPRIO[4:0] is set to CS-MINPRIO[4:0], and CS-CONFLICT is reset
9. For non-connected events (i.e. all Advertising Channels), in case the current event is aborted, the RW-BLE Software is required to re-schedule the event based on CS-CONFLICT information

Dynamic priority management is ensured depending on the connections in use in a Dual Mode device application.

This mechanism is described by the FSM of Figure 3-14.

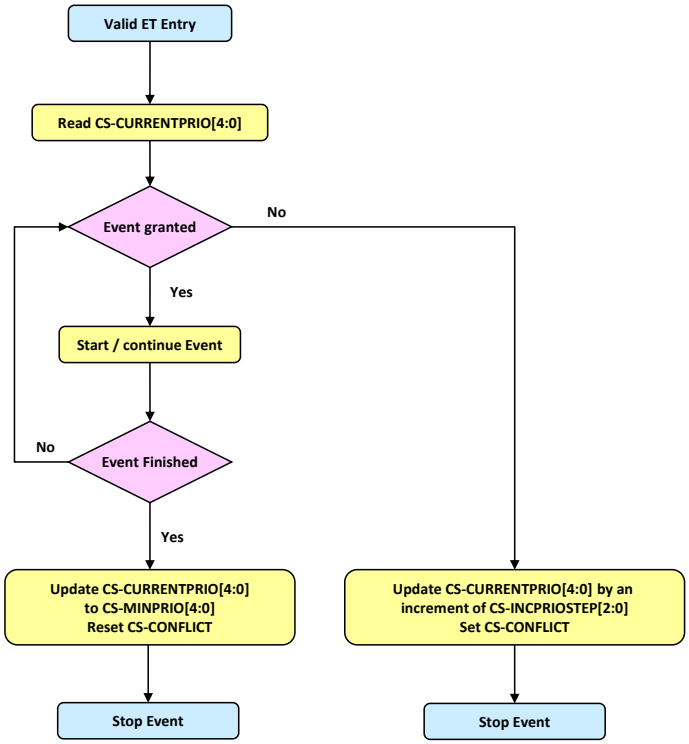

Figure 3-14 – Dual Mode Device Priority Management FSM

Additionally the decision instant is determined by BLEPRIOSCHARB register (See section 3.12.13 for details). Since pre-fetch instant is programmable, the system must wait for a given delay before the decision instant, as reflected in Figure 3-15. This delay is hence defined by the formula hereafter:

$$BLEDecisionInst = BLEPFInst + FCNTOFFSET - (RFPowerUp + BLEMARGIN)$$

with:

- BLEPFInst corresponding to TIMGENCNTL-PREFETCH_TIME[8:0]
- FNCTOFFSET corresponding to CS-FCNTOFFSET[9:0]
- RFPowerUp corresponding to either RADIOPWRUPDN-TXPOWERUP[7:0] or RADIOPWRUPDN-RXPOWERUP[7:0]
- BLEMARGIN corresponding a margin value allowing the RF power down properly before any other activity (see section 3.12.13)

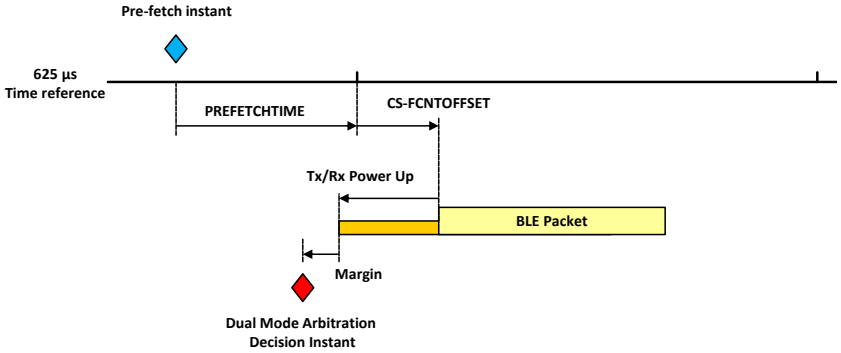

Figure 3-15 – Dual Mode Arbitration Decision instant

3.7 Advertising Channel

3.7.1 General considerations

Advertising channels are used typically for device discovery and device connection setup.

Advertising channels are used during Advertising events, Scanning events, and Initiating events.

Advertising channel packets have a particular PDU's header format as shown in Figure 3-16.

Figure 3-16 – Advertising PDU's Header Format

Advertising channel packet PDU's header is defined in CS-TXPHADV0/1 for transmits, and reported into CS-RXPHADV0/1 in receipt. Advertising channel packet PDU's Header Type field defines the advertising packet type, and its payload content.

Advertising channel packet PDU's Header TxADD and RxADD fields define specific information contained in the payload. RFU field is Reserved for Future Use and must be forced to 0x0 in transmit, and ignored on receipt.

Advertising channel mapping is defined according to ADVCHMAP register. Advertising Interval is defined according to ADVTIM-ADVINT register.

Type field values are described in Table 3-7.

Type	Packet Name
0x0	ADV_IND
0x1	ADV_DIRECT_IND
0x2	ADV_NONCONN_IND
0x3	SCAN_REQ
0x4	SCAN_RSP
0x5	CONNECT_REQ
0x6	ADV_SCAN_IND
0x7 – 0xF	Reserved

Table 3-7 – Advertising Packet Type field description

3.7.1.1 Advertising

There are four different advertising packet types that are:

- ADV_IND: connectable undirected advertising packet
- ADV_DIRECT_IND: connectable directed advertising packet
- ADV_NONCONN_IND: non-connectable undirected advertising packet
- ADV_SCAN_IND: scannable undirected advertising packet

These packets can be sent by the device in Advertising state, and can only be received either in Scanning state or in Initiating state. Each advertising packet has a payload for which the content differs and has a different signification, as shown in Figure 3-17.

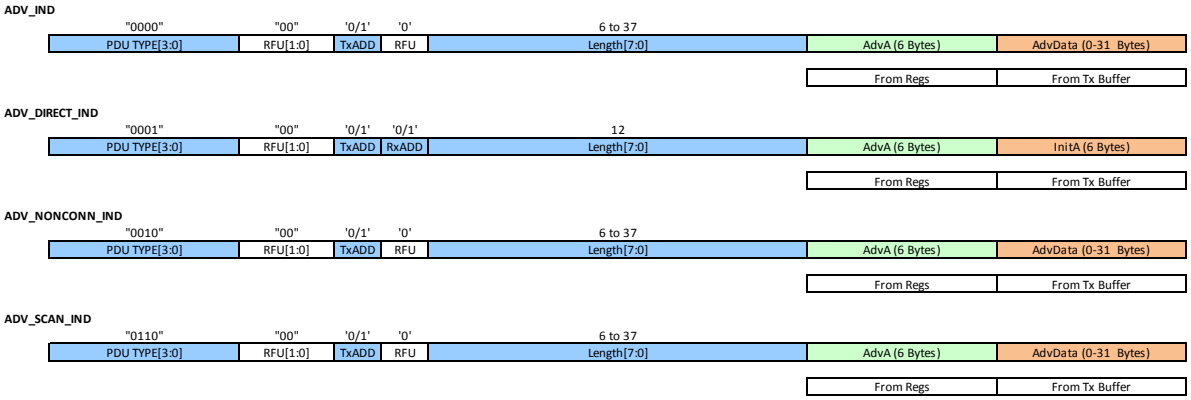

Figure 3-17 – Advertising Packets PDU content

For each packet, the *AdvA* 6-byte field of the payload defines the advertiser device address. When the device is in advertising mode (i.e. CS-FORMAT = 00100 or CS-FORMAT = 00101), the *AdvA* field is determined by BDADDRU and BDADDRL registers. *TxADD* is determined by BDADDRU-PRIV_NPUB field. Scanner devices and initiator devices compares *TxADD* and *AdvA* field to their own device white list , in order to decide whether it passes or fails their device filtering policy, and then may reply respectively either by a SCAN_REQ or a CONNECT_REQ packet.

In case of connectable directed advertising packet, the initiator device address (i.e. *InitA*) is appended in order to directly target the device to connect. *RxADD* field determines whether *InitA* address is private or public: both values are defined respectively by TxDESC-RXADD and in Tx Data Buffers.

Advertising packet can also content advertising data (i.e. *AdvDat*a field) that are specific to the application.

3.7.1.2 Scanning

There are two different scanning packets that are:

- SCAN_REQ: scan request packet
- SCAN_RSP: scan response packet

Scan request is sent by the device in Scanning State, and received in Advertising state only. Scan response is sent in Advertising state and received in Scanning state.

SCAN_REQ and SCAN_RSP packets have a specific payload as shown in Figure 3-18:

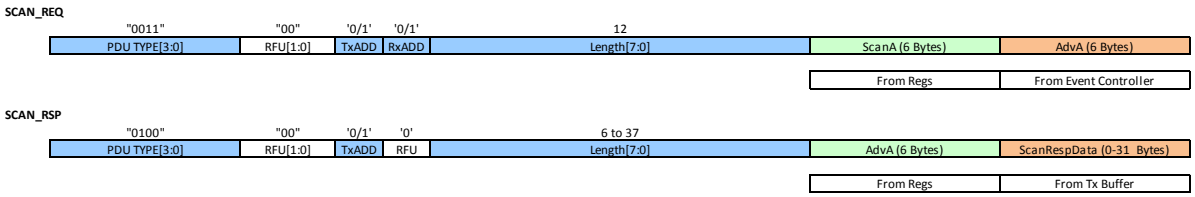

Figure 3-18 – Scanning Packets PDU content

When advertising packet passes the scanner device filtering policy, and when the device is in Active Scanning mode (i.e. CS-FORMAT = 01001), a SCAN_REQ packet is sent. The *ScanA* 6-byte field corresponds to the scanner device address. The *ScanA* field is picked from the BDADDRU and BDADDRL registers, instead of the Exchange Memory. *AdvA* field corresponds to the Advertiser device address that is returned, as well as *RxADD* field indicating whether the Advertiser device is private or public: both values come from the Advertising packet previously received.

On advertiser side, on SCAN_REQ receipt, the *ScanA* field is compared to its white list in order to determine whether it passes or fails its device filtering policy. The *AdvA* 6-byte field corresponds to the advertiser address to which the SCAN_REQ is sent. The *AdvA* field must be checked before allowing any SCAN_RSP packet response.

If SCAN_REQ packet passes the advertiser device filtering policy, a SCAN_RSP packet is sent. The *AdvA* 6-byte field corresponds to the advertiser device base address.

3.7.1.2.1 Passive Scan

In passive scan, the device only receives advertising packets, and does not send any scan request.

3.7.1.2.2 Active Scan

In active scan, the device receives advertising packets, and can send SCAN_REQ in certain conditions.

On ADV_DIRECT_IND and ADV_NONCONN_IND advertising packets, the devices must not send SCAN_REQ packet.

On ADV_IND and ADV_SCAN_IND advertising packets, depending on the filter policy, the device can send SCAN_REQ packet.

After sending a SCAN_REQ packet, the device must listen to SCAN_RSP packets. If a SCAN_RSP is received, it is considered as a success, else a failure.

However, in order to avoid collisions during scanning, a counting method is performed onto the successes and failures as described above, using *BackOffCnt* counter and *UpperLimit* value. On each scanning event, *BackOffCnt* counter and *UpperLimit* value are set to 0x1.

On two consecutive failures, *UpperLimit* value must be doubled until it reaches the value of 256. On every two consecutive successes, *UpperLimit* value must be halved until it reaches the value of 1.

On ADV_IND and ADV_SCAN_IND advertising packets, when the device filtering policy allows a SCAN_REQ packet to be sent, *BackOffCnt* must be decremented by 1. SCAN_REQ packets are sent only when *BackOffCnt* is zero.

On each SCAN_RSP successful or failed reception, *BackOffCnt* must be updated to a pseudo-random value in between [1:*UpperLimit*] range. This is done using a PRBS (based onto an 8-bit LFSR) and extracting the LSB value according to *UpperLimi*t max value.

BackOffCnt and *UpperLimit* are reported respectively in ACTSCANSTAT-BACKOFFCNT and ACTSCANSTAT-UPPERLIMIT register fields for testing purpose.

BackOffCnt and *UpperLimit* management algorithm is described in Figure 3-19.

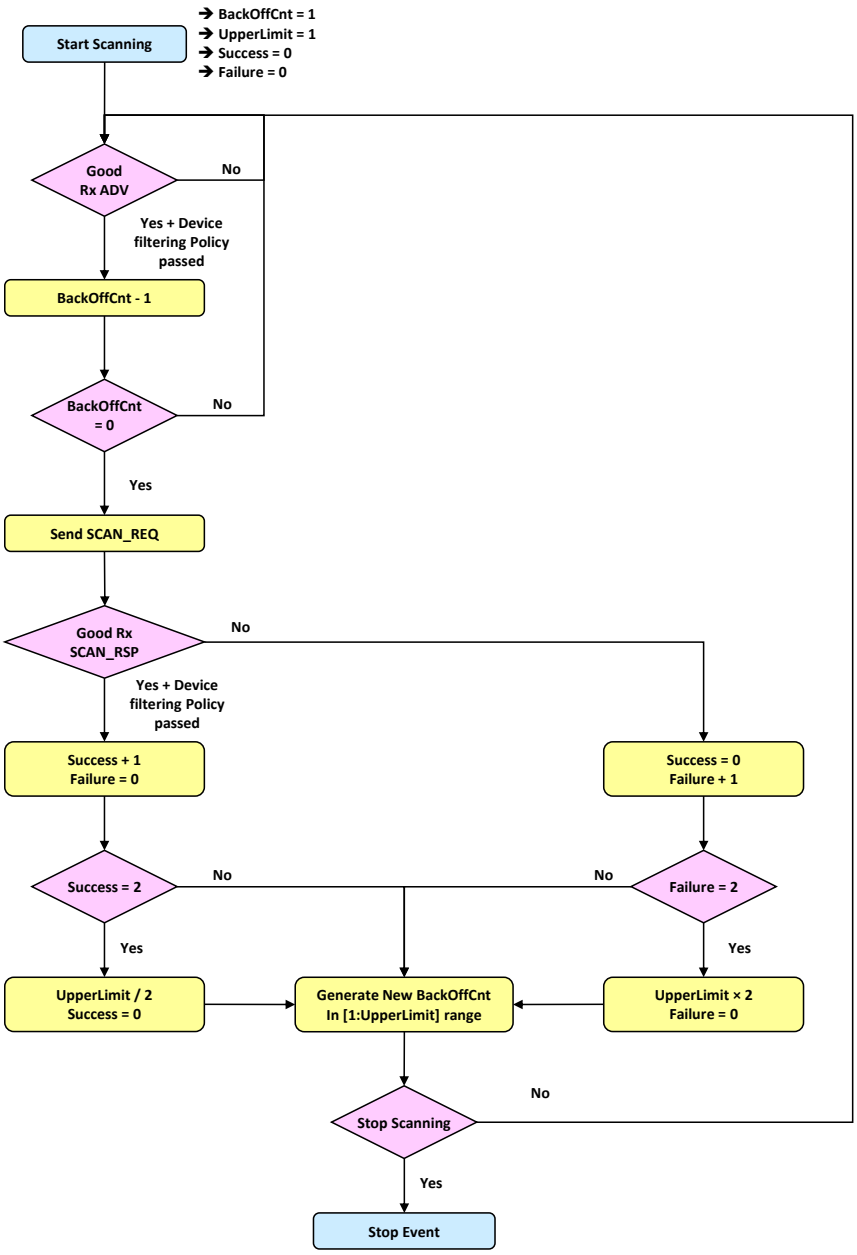

Figure 3-19 – BackOff and UpperLimit behavioral FSM

Figure 3-20 shows a short Back-Off mechanism sequence chart, that explains *BackOffCnt* and *UpperLimit* respective behavior for two scanners and one advertiser that starts at the same time. Note that in this case device filtering is not considered in order to ease understanding.

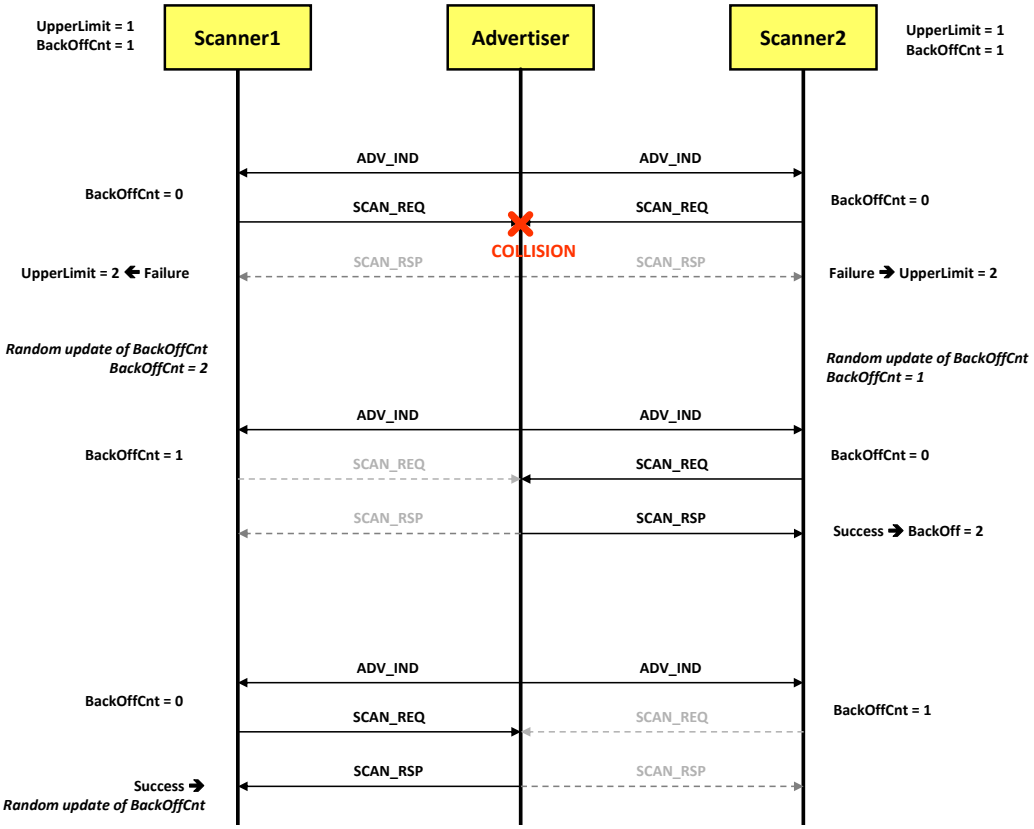

Figure 3-20 – BackOff Mechanism Sequence chart example

3.7.1.3 Initiating

There is only one packet used for connection initiating, that is

CONNECT_REQ

A connection request is sent by a device in Initiating state, and received by a device in Advertising state. According to the device filtering policy, once a CONNECT_REQ packet is received, the device can jump to connected state: this is done by programming a new Exchange Table and new Control Structure entry for a master device role. A device that was in Initiating state is then connected as master role. A device that was in Advertising state is then connected as slave role. CONNECT_REQ packet has a specific payload as shown in Figure 3-21.

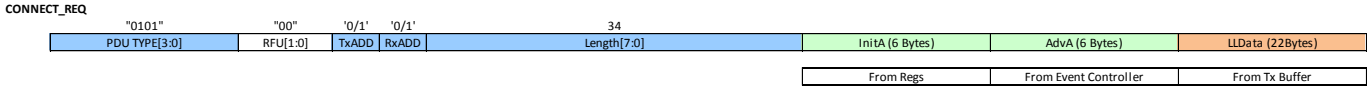

Figure 3-21 – Connection request Packet PDU content

When advertising packet passes the initiator device filtering policy, and when the device is in Initiating mode (i.e. CS-FORMAT = 01111), a CONNECT_REQ packet is sent. The *InitA* 6-byte field corresponds to the initiator device address. The *InitA* field is determined by BDADDRU and BDADDRL registers. *TxADD* field is defined by BDADDRU-PRIV_NPUB field. *AdvA* field corresponds to the Advertiser device address that is returned, as well as *RxADD* field indicating whether the Advertiser device is private or public: both values come from the Advertising packet previously received

On advertiser side, the *InitA* field is compared to its white list in order to determine whether it passes or fails its device filtering policy, and allows connection. The *AdvA* 6-byte field corresponds to the advertiser address to which the CONNECT_REQ is sent. The *AdvA* field must be checked by the advertiser device in order to determines whether it accept the connection creation.

3.7.2 Device Filtering

3.7.2.1 General rules

Device filtering is based onto peer device's address. It is used in order to minimize the number of devices to which to respond during device discovery / connection. Filter policies in between advertising state, scanning state and initiating state are independent. Device filtering policy is determined by CS-FILTER_POLICY field, and applies differently according to CS-FORMAT value. Additionally, RPA resolution may be enabled when the device uses Enhanced Privacy mode.

On top of the device filtering policy, the Advertising Event type allows, or forbid, the reply to advertising packets, as well as the scanning and connection requests, as described in Table 3-8 below.

Advertising Event Type	Advertising Packet Type	Allowable response packets	
		SCAN_REQ	CONNECT_REQ
Connectable Undirected event	ADV_IND	YES	YES
Connectable Directed event	ADV_DIRECT_IND	NO	YES (1)
Non-Connectable Undirected event	ADV_NONCONN_IND	NO	NO
Scannable Undirected event	ADV_SCAN_IND	YES	NO

Table 3-8 – Advertising Event Types and Device Filtering Policies

(1): Only the correctly addressed initiator may respond

3.7.2.2 White List Management

In order to perform address device filtering, the device must refer to a list of known devices: this list is named the White List. This list stands in the Exchange Memory, and depending on the device filtering policy, it is accessed after each received packets by the Event Controller. The Event controller performs this verification during the 150µs Inter Frame Space, when CS-FORMAT is set to advertising, scanning or initiating modes.

As the White List can contain several devices 48-bit address, it is accessed per block. Also, it is split in between public and private address: WLPUBADDPTR and WLPRIVADDPTR define respectively public and private white list address start pointers.

The number of addresses to check is defined respectively by WLNBDEV-NBPUBDEV and WLNBDEV-NBPRIVDEV.

Furthermore, device's address in the white list can be removed time to time, and then RW-BLE Software replaces it by 0xFFFFFFFFFFFF value that is ignored during parsing. This mechanism allows avoiding complete reordering of the addresses in the white list by the RW-BLE Software.

In order to speed-up the address parsing, addresses are tested LSB to MSB.

3.7.2.3 Resolving Address List Management

When the device uses enhanced Privacy mode, then instead of using the White List, it uses the Resolving Address List. This list stands in the Exchange Memory, and used when the Enhanced Privacy Mode is used. The RAL Lis is accessed after each received packets, when CS-FORMAT is set to advertising, scanning or initiating modes.

As the Resolving Address List can contain several structures, it is accessed per block with RALPTR defining the Resolving Address List start pointers. In case a RPA is identified as matching, the corresponding structure address pointer is reported in the current Rx Descriptor in use.

The number of addresses to check is defined by RALNBDEV.

RAL Structure is described in Figure 3-22. Table 3-9 gives description of the RAL Structure fields.

Address	Access		Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HW	SW																	
00'H	R/W	R/W	RAL_INFO	ENTRY_VALID	CONNECTED	IN_WHLIST						LOCAL_RPA_VALID	LOCAL_RPA_RENEW	LOCAL_IRK_VALID		PEER_RPA_VALID	PEER_RPA_RENEW	PEER_IRK_VALID	PEER_ID_TYPE
02'H-10'H	R	W	RAL_PEER_IRK	PEER_IRK[127:0]															
12'H-16'H	R/W	W	RAL_PEER_RPA	PEER_RPA[47:0]															
18'H-1C'H	R	W	RAL_PEER_ID	PEER_ID[47:0]															
1E'H-2C'H	R	W	RAL_LOCAL_IRK	LOCAL_IRK[127:0]															
2E'H-32'H	R/W	W	RAL_LOCAL_RPA	LOCAL_RPA[47:0]															

Figure 3-22 – RAL Structure

Field name	Description	Comments
ENTRY_VALID	Indicates whether the structure is ready to be used by the RAL engine 0: Invalid entry, can be skipped 1: Valid entry, can be used	
CONNECTED	Indicates the peer device is already connected to the RW-BLE Core 0: Peer is not connected 1: Peer is connected	If set, indicates that no response has to be sent to the peer.
IN_WHLIST	Indicates the peer device is in the White List 0: Peer device is not in the White List 1: Peer device is in the White List	Indicates as the device is already in the White List, no need to perform White List Search
LOCAL_RPA_VALID	Local RPA validity flag 0: Local RPA is not valid 1: Local RPA is valid	
LOCAL_RPA_RENEW	Indicates the local RPA has to be renewed	Set at 1 by the RW-BLE Software, reset by the RW-BLE Core once renewed Calculated RPA to be saved in RAL-LOCAL_RPA
LOCAL_IRK_VALID	Local IRK Validity flag 0: Local IRK is not valid / Local IRK is null 1: Local IRK is valid	
PEER_RPA_VALID	Peer RPA validity flag 0: Peer RPA is not valid 1: Peer RPA is valid	

PEER_RPA_RENEW	Indicates the peer device RPA has to be renewed	Set at 1 by the RW-BLE Software, reset by the RW-BLE Core once renewed Calculated RPA to be saved in RAL-PEER_RPA
PEER_IRK_VALID	Peer device IRK Validity flag 0: Peer IRK is not valid / Peer IRK is null 1: Peer IRK is valid	Set if valid, else indicate RAL-PEER_IRK is null
PEER_ID_TYPE	Peer Device Identity Type 0: Public Address 1: Private Static Address	
PEER_IRK[127:0]	Peer device IRK	
PEER_RPA[47:0]	Peer device RPA	
PEER_ID[47:0]	Peer device Identity Address	
LOCAL_IRK[127:0]	Local IRK	
LOCAL_RPA[47:0]	Local RPA	

Table 3-9 – RAL Structure content description

3.7.2.4 Device Filtering Sequence Charts

3.7.2.4.1 White List Device Filtering Operations

3.7.2.4.1.1 Connectable Undirected Event

In Connectable Undirected Event, the advertiser uses ADV_IND advertising packets, and it can receive either SCAN_REQ from a scanner, or a CONNECT_REQ from an initiator. Figure 3-23 shows the sequence chart for a connectable undirected event in advertiser / scanner use case.

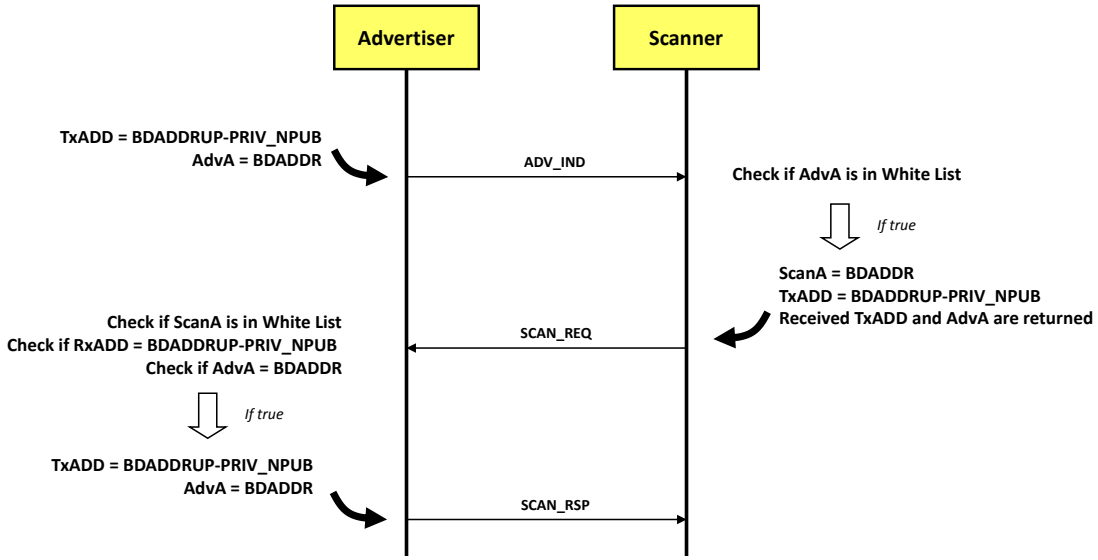

Figure 3-23 – MSC for Connectable Undirected Event / Advertiser – Scanner Case

Figure 3-24 shows the sequence chart for a connectable undirected event in advertiser / initiator use case.

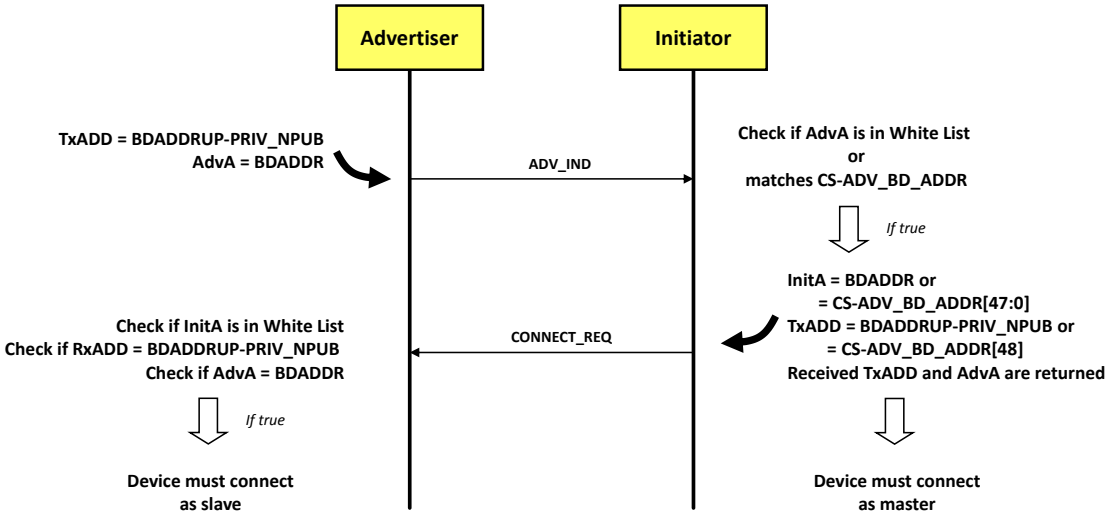

Figure 3-24 – MSC for Connectable Undirected Event / Advertiser – Initiator Case

3.7.2.4.1.2 Connectable Directed Event

In Connectable Directed Event, ADV_DIRECT_IND advertising packets are used. Only initiator devices are allowed to respond with CONNECT_REQ packets. Figure 3-25 shows the sequence chart for a connectable directed event in advertiser / initiator use case.

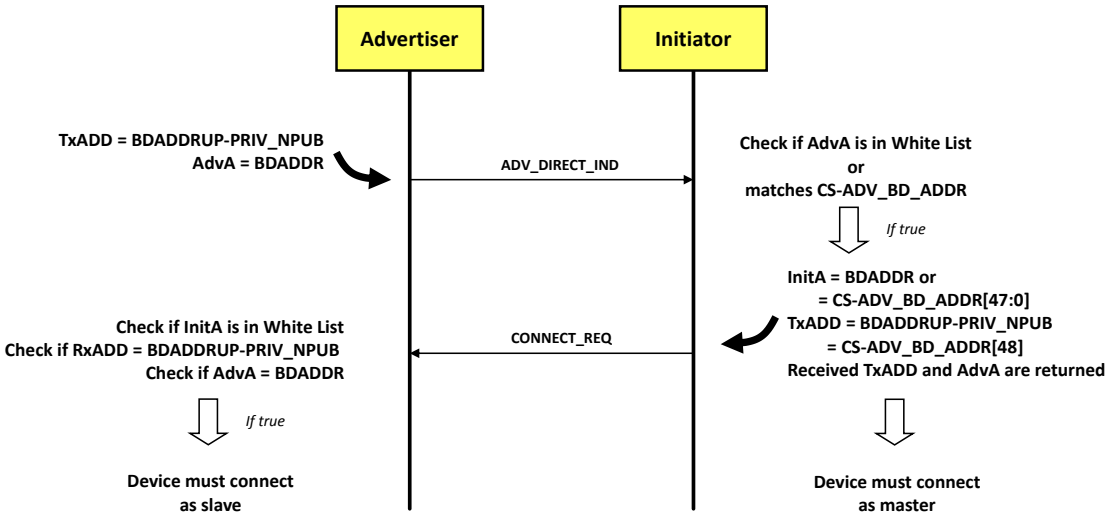

Figure 3-25 – MSC for Connectable Directed Event

3.7.2.4.1.3 Scannable Undirected Event

In Connectable Directed Event, ADV_SCAN_IND advertising packets are used. Only scanner devices are allowed to respond with SCAN_REQ packets.

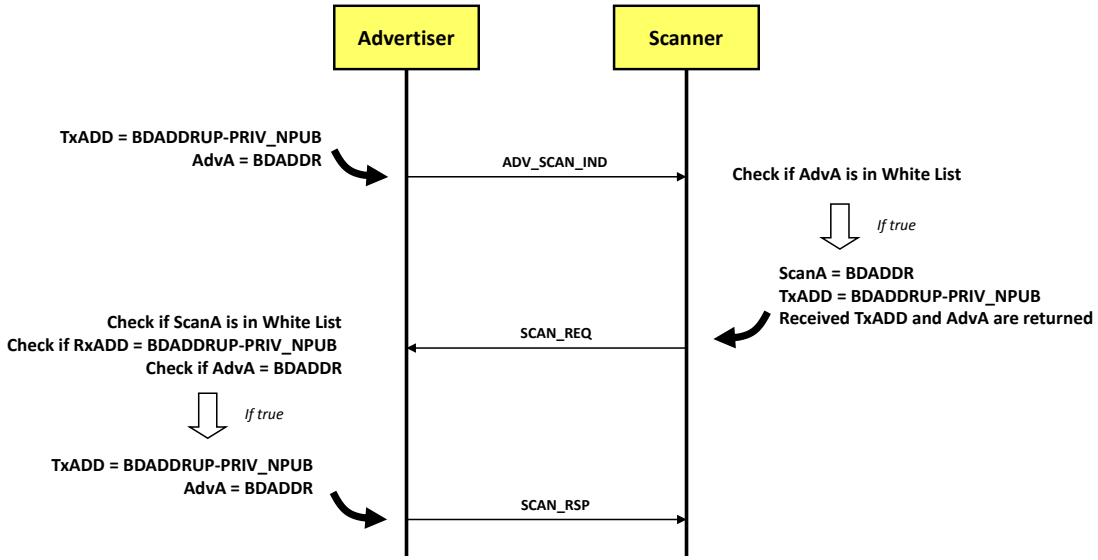

Figure 3-26 – MSC for Scannable Undirected Event

3.7.2.4.2 Resolvable Address List Device Filtering Operations / Enhanced Privacy

3.7.2.4.2.1 Connectable Undirected Event

In Enhanced Privacy Mode, and during Connectable Undirected Event, the advertiser uses ADV_IND advertising packets, and it can receive either SCAN_REQ from a scanner, or a CONNECT_REQ from an initiator. In this case, the Resolving Address List engine needs to be used if the Device Filtering Policy allows White List usage. Figure 3-27 shows the sequence chart for a connectable undirected event in advertiser / scanner use case.

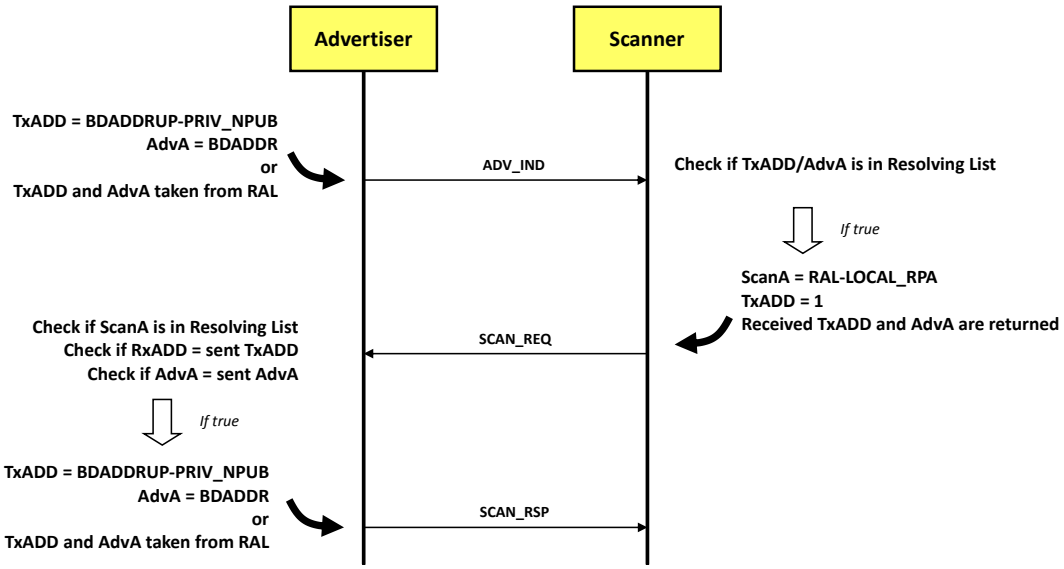

Figure 3-27 – Enhanced Privacy MSC for Connectable Undirected Event / Advertiser – Scanner Case

Figure 3-28 shows the sequence chart for a connectable undirected event in advertiser / initiator use case.

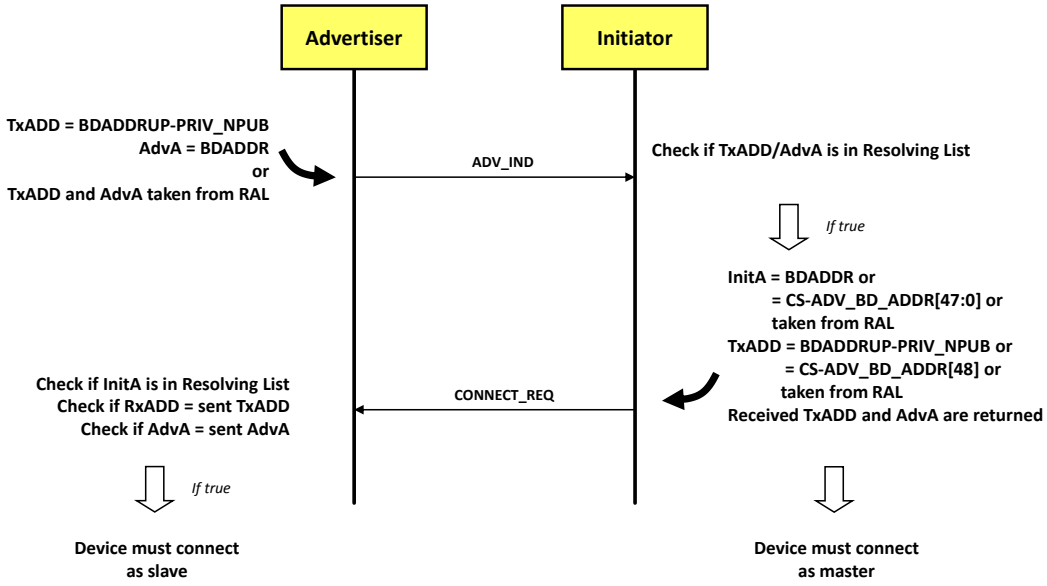

Figure 3-28 – Enhanced Privacy MSC for Connectable Undirected Event / Advertiser – Initiator Case

3.7.2.4.2.2 Connectable Directed Event

In Enhanced Privacy Mode, and during Connectable Directed Event, ADV_DIRECT_IND advertising packets are used. Only initiator devices are allowed to respond with CONNECT_REQ packets. Figure 3-29 shows the sequence chart for a connectable directed event in advertiser / initiator use case.

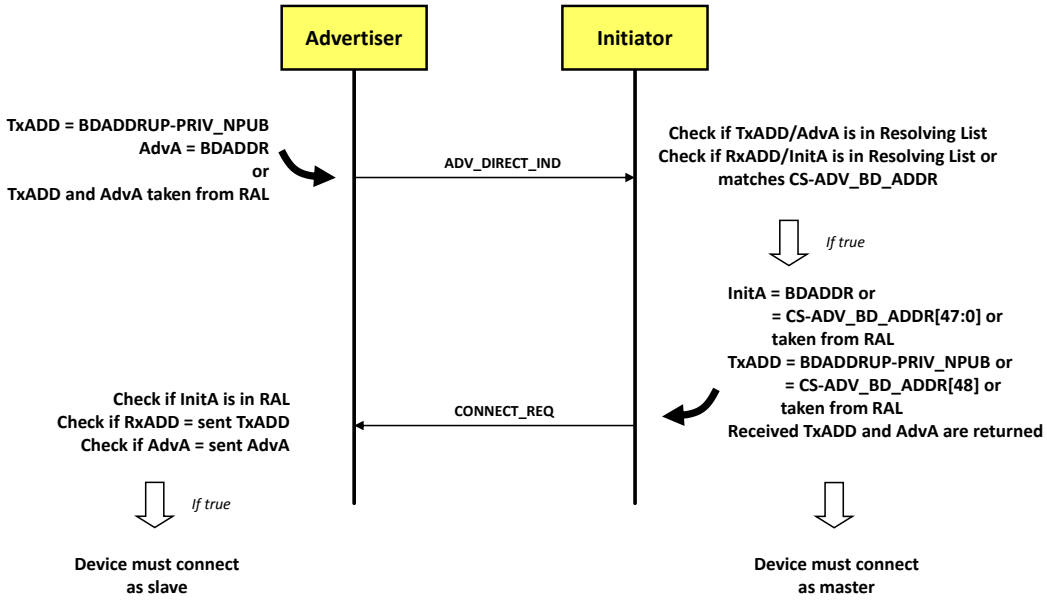

Figure 3-29 – Enhanced Privacy MSC for Connectable Directed Event

3.7.2.4.2.3 Scannable Undirected Event

In Enhanced Privacy Mode, and during Connectable Directed Event, ADV_SCAN_IND advertising packets are used. Only scanner devices are allowed to respond with SCAN_REQ packets.

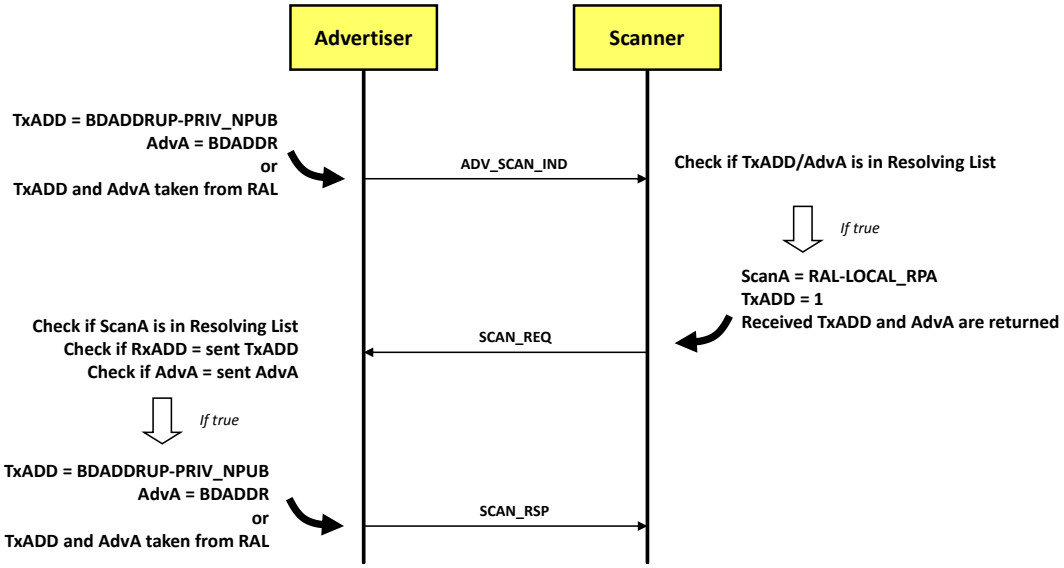

Figure 3-30 – Enhanced Privacy MSC for Scannable Undirected Event

3.7.3 Advertising Channel Event Timings

3.7.3.1 Connectable Event

3.7.3.1.1 Connectable Undirected Event

When connectable undirected event type is used, then ADV_IND advertising packets are sent by the advertiser device. Each advertising packet is sent with an interval equal or less than 10ms; this interval is determined by ADVTIM-ADVINT register.

Figure 3-31 shows an example of connectable undirected event with advertising packets only (i.e. no scan request, no connection request). Note in this case CS-FORMAT is set to Low Duty Cycle Advertising value, and that High Duty Cycle Advertising CS-FORMAT cannot be used.

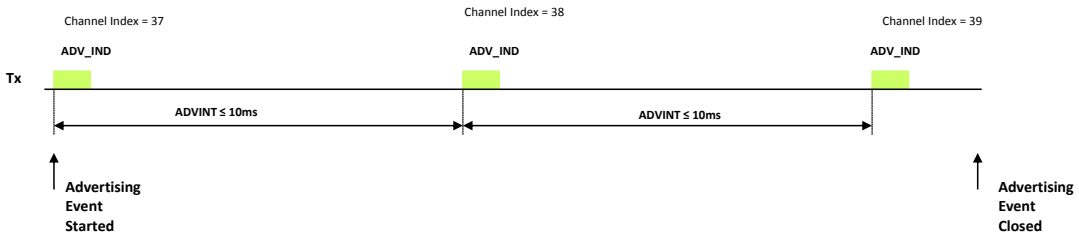

Figure 3-31 – Connectable Undirected Advertising Event with advertising packets only

A scanner can respond with a SCAN_REQ packet in respect with its device filtering policy, and then the advertiser can send a SCAN_RSP packet according to its device filtering policy.

Figure 3-32 shows an example of connectable undirected event with scan request packet and scan response packet on second advertising packet.

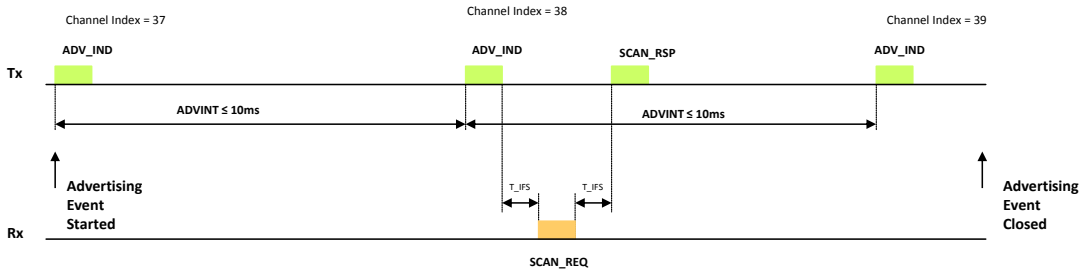

Figure 3-32 – Connectable Undirected Advertising Event with scan request on 2nd advertising packet

An initiator can respond with a CONNECT_REQ packet depending on its device filtering policy. If the advertiser receives properly the connection request, and if the device filtering policy is respected, then the event is closed and the advertiser must jump to slave connected state.

Figure 3-33 shows connectable undirected event with connection request packet on second advertising packet.

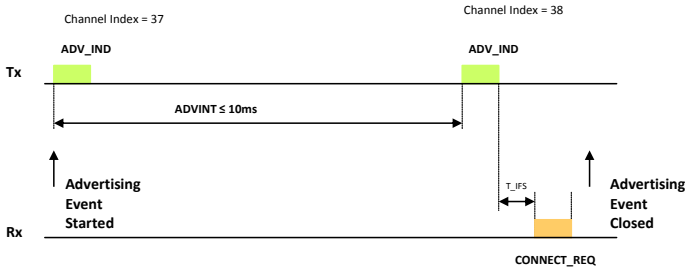

Figure 3-33 – Connectable Undirected Advertising Event with connection request on 2nd advertising packet

3.7.3.1.2 Connectable Directed Event

When connectable directed event type is used, then ADV_DIRECT_IND packets are sent by the advertiser device. Advertising packets onto the same channel are sent with an interval equal or less than 3.75ms; this interval is determined by ADVTIM-ADVINT register that is used to determine the interval in between advertising packet transmission.

SCAN_REQ packets are ignored by the advertiser, and only CONNECT_REQ are processed if the packet matches the advertiser device filtering policy. In this case the device must jump into Slave connected state.

Figure 3-34 shows connectable directed event with advertising packets only (i.e. no connection request received) Note in this case CS-FORMAT is set to High Duty Cycle Advertising value. Additionally, it is also possible to perform Connectable Directed Advertising event using Low Duty Cycle Advertising CS-FORMAT value: in this case timings as described in Figure 3-31 apply.

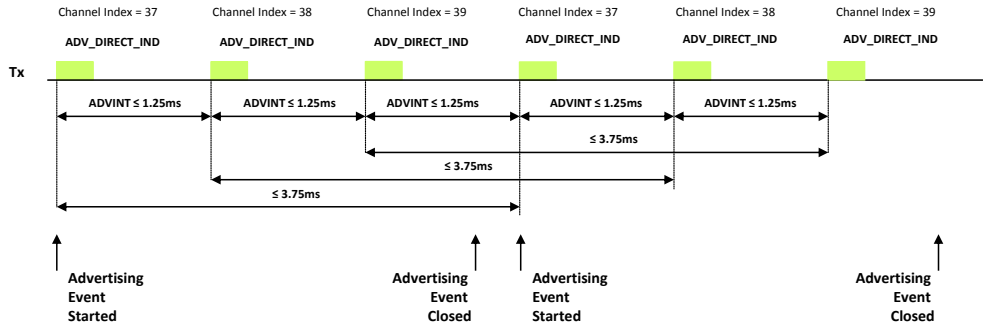

Figure 3-34 – Connectable Directed Advertising Event with advertising packets only

3.7.3.2 Scannable Undirected Event

When scannable undirected event type is used, then ADV_SCAN_IND packets are sent by the advertiser device. It allows a scanner device to respond with a SCAN_REQ packet in order to get more information about the advertiser. When the advertiser device receives SCAN_REQ packet that matches its advertising policy, then SCAN_RSP packets are sent to the scanner device.

Figure 3-35 shows an example of scannable undirected event with advertising packets only (i.e. no scan request). Note in this case CS-FORMAT is set to Low Duty Cycle Advertising value, and that High Duty Cycle Advertising CS-FORMAT cannot be used.

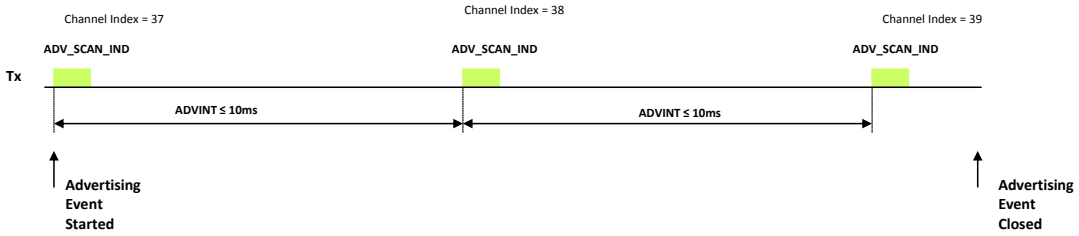

Figure 3-35 – Scannable Undirected Advertising Event with advertising packets only

Figure 3-36 shows an example of scannable undirected event with scan request packet and scan response packet on second advertising packet.

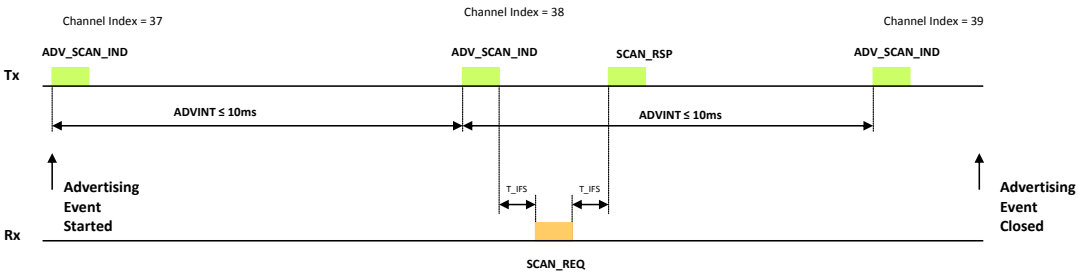

Figure 3-36 – Scannable Undirected Advertising Event with scan request on 2nd advertising packet

3.7.3.3 Non–connectable Event

When non-connectable undirected event type is used, then ADV_NONCONN_IND packets are sent by the advertiser device. It allows a scanner to receive information from the advertiser device. The advertiser device never listens after the ADV_NONCONN_IND packet transmission.

Figure 3-37 shows an example of non-connectable undirected event. Note in this case CS-FORMAT is set to Low Duty Cycle Advertising value, and that High Duty Cycle Advertising CS-FORMAT cannot be used.

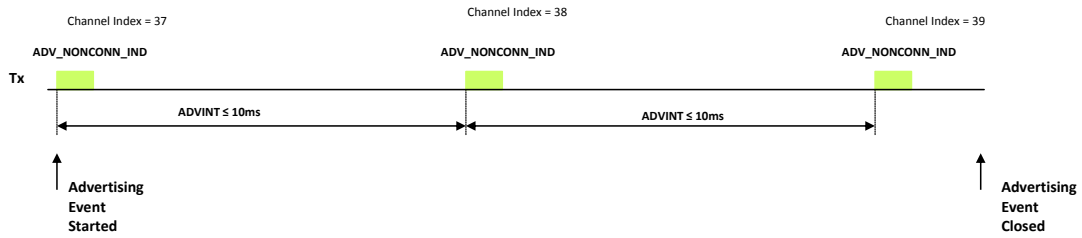

Figure 3-37 – Non-Connectable Undirected Advertising Event

3.7.3.4 Connection Setup

3.7.3.4.1 General Considerations

In order to ease master device scheduling, particularly when dealing with several slave devices, some timing flexibility has been added to the system in order to ease first data packet exchange. CONNECT_REQ packet sent by the master contains three parameters used to determine the first anchor point on which data packets are exchanged.

These parameters are:

- Transmit Window Offset (*transmitWindowOffset*), is a multiple of 1.25ms in the [0 : *connInterval*] range
- Transmit Window Size (*transmitWindowSize*), that is a multiple of 1.25ms in the [1.25ms : min(10ms, *connInterval* - 1.25ms)] range
- Connection Interval (*connInterval*), that is a multiple of 1.25ms in the [7.5ms : 4s] range

The first anchor point must not be earlier than 1.25ms + *transmitWindowOffset*, and no later than 1.25ms + *transmitWindowOffset* + *transmitWindowSize* after the end of the of the CONNECT_REQ packet.

In addition of *connInterval* value, *connSlaveLatency* value must be used for anchor point definition. The *connSlaveLatency* value defines the sub-rating for which the slave device is not required to listen for the master.

Theses parameters are used by the RW-BLE Software in order to determine the connection events scheduling.

Please refer to section 3.8.3 for scheduling details.

3.7.3.4.2 Successful connection setup

Figure 3-38 shows successful connection setup chronogram.

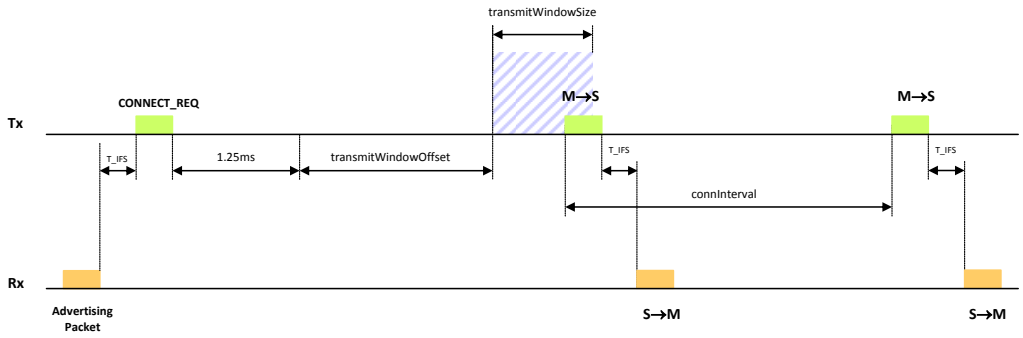

Figure 3-38 – Successful connection setup

3.7.3.4.3 Failed connection setup, and retry

In case first M→S packet is not received properly by the slave device during connection setup, then S→M is not sent by the slave. The master device then must retry to setup the connection as depicted in Figure 3-39, that shows a first attempt connection setup failure, and followed by a successful connection setup retry.

This particular scheduling is handled by the RW-BLE software.

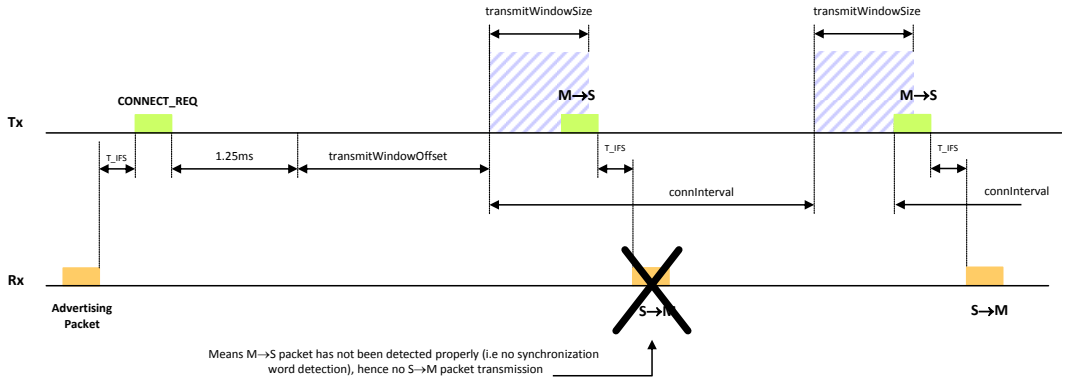

Figure 3-39 – Failed connection setup on first attempt, and retry

This mechanism is repeated on each connection interval till the slave device replies to the master.

3.7.3.4.4 Management of transmit Window Offset Value

In order to optimize connection delay, RW-BLE Software provides the instant at which the first Exchange Table entry (pointing onto a master Control Structure) value through CS-WINOFFSET.

Event Scheduler is in charge of calculating a valid *transmitWindowOffset* when a known Advertiser is whiling to connect. The Packet Controller replaces in real time the *transmitWindowOffset field* in the CONNECT_REQ packet in order to optimize and reduce connection time, evaluating the remaining time till next anchor point (i.e provided by *TxWinOffsetCounter* internal counter value).

Figure 3-40 describes this mechanism.

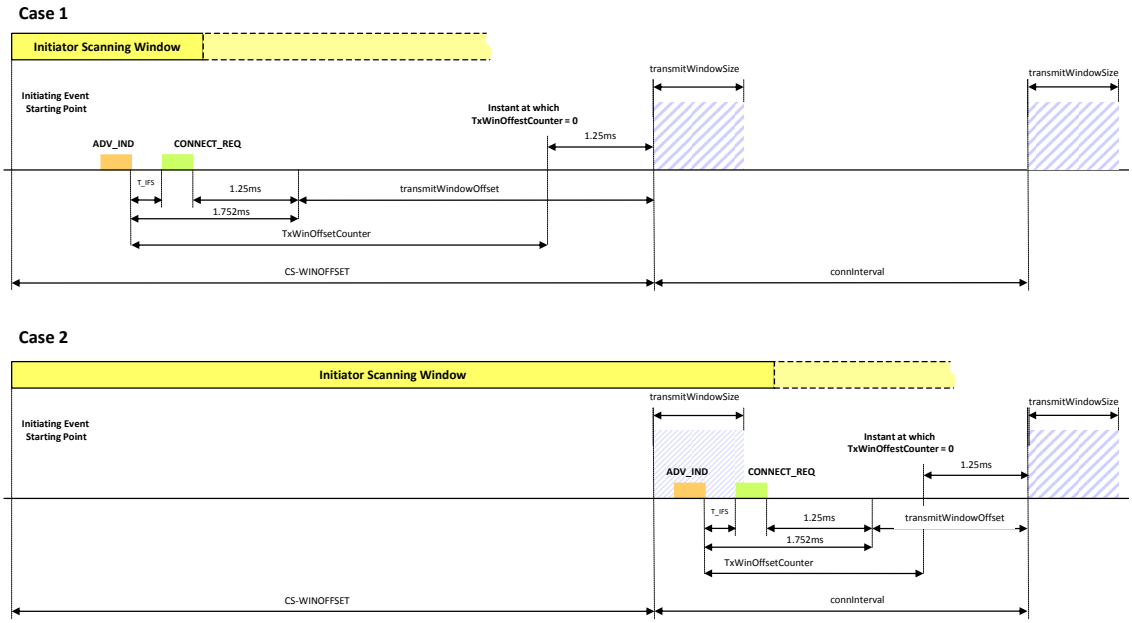

Figure 3-40 – Transmit Window Offset real Time processing

In case 1, the Advertising packet is received prior to *TxWinOffsetCounter* becomes null. Once it is null, and once a 625µs time reference instant happens, *TxWinOffsetCounter* is re-initialization to *connInterval*, and then case 2 applies.

This means the RW-BLE Software is able to control exactly when the first connection event happens, as *transmitWindowOffset* is calculated in real time, updated by Event Controller, and the resulting value inserted by the Packet Controller when CONNECT_REQ packet is sent.

The following formula applies for *transmitWindowOffset* initialization once Initiating is started:

$$transmitWindowOffset = floor\left(\frac{TxWinOffsetCounter - \frac{transmitWindowSize}{2} - 1752\mu s}{1250\mu s}\right)$$

Initial value of *transmitWindowOffset* is calculated using CS-WINOFFSET field. Connection interval is provided to the RW-BLE Core through CS-CONNINTERVAL field. Each time calculated *transmitWindowOffset* value becomes null during an Initiating window, then *transmitWindowOffset* is reset to CS-CONNINTERVAL and processing resumes till a valid advertising packet is received.

Real time *transmitWindowOffset* value sent over CONNECT_REQ packet is reported to the RW-BLE Software through CS-TXWINOFFSET field.

Note it is forbidden to set CS-WINOFFSET to a lower value than CS-CONNINTERVAL.

3.8 Data Channel

3.8.1 General Considerations

Data channels are used typically when two devices are connected. Data channel is used either to convey Link Layer user data, or Link Layer control data. Link Layer Data and Control packets have a particular PDU's header format as shown in Figure 3-41.

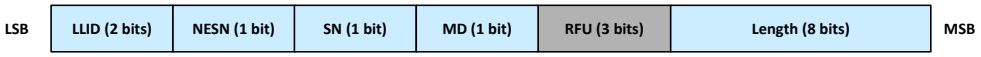

Figure 3-41 – Link Layer Data / Control Packet PDU's Header Format

Data channel packet PDU's header is defined in TxDESC-TXPHCE for transmits, and reported into RxDESC-RXPHCE in receipt. Details about LLID, NESN, SN, MD, and Length field details are provided in Table 3-2 and the sections hereafter. RFU field is Reserved for Future Use and must be forced to 0x0 in transmit, and ignored on receipt.

Link Layer channel mapping is defined according to CS-LLCHMAP.

3.8.2 Acknowledgement and Number of Exchanged Packets scheme

3.8.2.1 Packet Acknowledgement Scheme

Packet acknowledgements and retransmission requests are managed according to NESN (Next Sequence Number) bit that is used as an acknowledgment indicator of the good reception of previous packet that has been sent. For NESN, there are two cases to consider:

- If the device is master, then CS-NESN bit is used on the first packet to transmit, then NESN toggles each time a packet is received without errors.
- If the device is slave, then CS-NESN indicates the value of the last transmitted packet, and it is toggled each time a packet is received without errors. Received NESN bit in saved in RxDESC-RXPHCE.

On each packet receipt, the following checks are performed:

1. Synchronization Word detection
2. Received Packet length check, that must be under the maximum possible packet length
3. Received SN bit check (detects duplicate packets)
4. Received NESN bit check
5. Received Packet CRC check

Synchronization Word detection failure, as well as received packet length error lead to retransmission request and the end of the connection event (i.e. due to timing alignment loss). Any error in the SN sequence, as well as CRC error, leads to retransmission request, but the connection event is not closed in this case. Received NESN error (i.e. meaning no toggle in between two consecutive received packets) is considered as a retransmission request rather than a real error.

Figure 3-42 shows the acknowledgement / retransmission scheme of the RW-BLE Core.

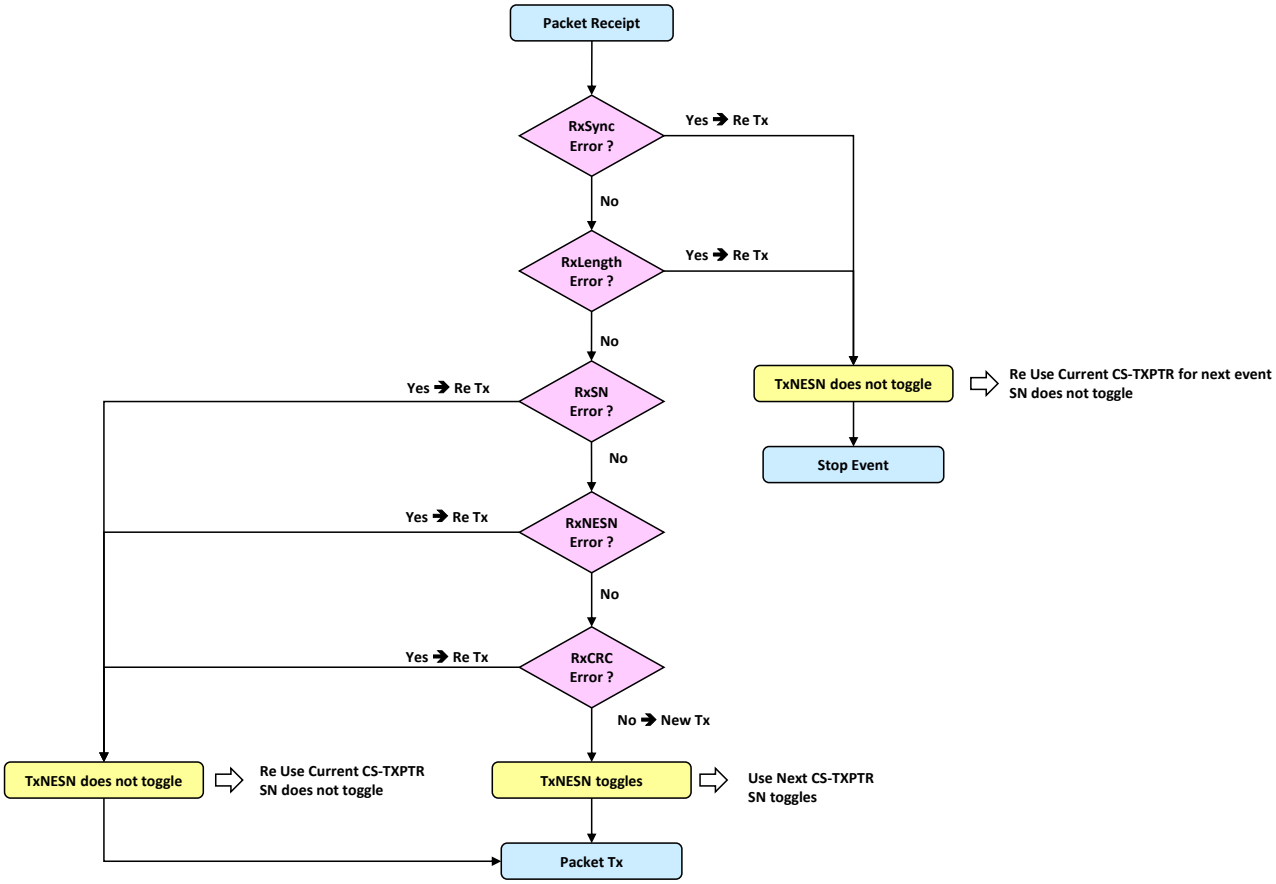

Figure 3-42 – Acknowledgment / Retransmission scheme

Notre that MIC error is not processed in the acknowledgment / retransmission request scheme as it leads to the termination of connection, and to disconnection.

3.8.2.2 Number of Exchanged Packets during Connection Event

The number of packets that are exchanged during a Link Layer connection event is determined by master and slave's MD bit of the PDU's header. The closure of a connection event is driven by the master, and it uses the MD bits provided by master and slave devices. MD bits are handled in different way depending whether the device is master connected or slave connected.

If master's MD bit = 1, and whatever slave's MD bit is, then the connection event is not closed: there are still data to exchange in between the connected devices. However the connection event is closed in anyway if the connection event duration is greater than CS-MAXEVTIME. Please note that if CS-MAXEVTIME equals 0 then the event has no time limitation.

If master's MD bit = 0 and slave's MD bit = 0, then the connection event is closed: there are no more data to exchange in between the devices.

If master's MD bit = 0 and slave's MD bit = 1, then the connection event can be closed by the master device, since the connection event duration is greater than CS-MINEVTIME.

In case of slave device, whatever master's MD bit value is, when event duration is greater than CS-MAXEVTIME, it forces the device to stop the event on a Tx packet. This is a protection mechanism that prevents programmed events to overlap, and that provides time to RW-BLE Software for Exchange table reprogramming and Rx Buffer read.

CS-MAXEVTIME and CS-MAXEVTIME are informative values that are used for bandwidth reservation.

3.8.3 Data Channel Connection Event Timings

Data Channel connection event timing is defined as per the *connInterval* parameter provided by the initiator device during connection setup, and then becomes the master of the connection. A master device imposes the scheduling to its Bluetooth Low Energy network, and as slaves have to follow the master scheduling, slave devices have to compensate for the clock drift, that lead to uncertainty in the anchor point time position.

During connection setup, a master provides its low power clock accuracy to a slave device. So the slave can calculate the expected anchor point time using the following formula:

$$RXWINSZ = 16 + 2 * \left(\frac{masterSCA + slaveSCA}{1000000} * connInterval + 1 \right)$$

with:

masterSCA : the master device low power clock accuracy in ppm

slaveSCA : the slave device low power clock accuracy in ppm

connInterval : the connection event interval in multiple of 1.25ms

RXWINSZ : result reported in the in CS-RXWINSZ field either in µs or in 625µs multiple values. Please refer to sections 2.5.3 and 3.3 for details.

Sixteen microseconds are added as an uncertainty of 16µs is allowed by the standard onto relative anchor point position. One microsecond is added in order to take into account the uncertainty of the Synchronization Found pulse edge position. In case an anchor point is lost by the slave (e.g. Synchronization Word detection failure due to interferers as example, or length bigger than the maximum possible value), the slave device must calculate the next anchor point and hence extends *RXWINSZ* by a factor of the number of missed synchronization (or missed events in case of slave latency). Once the anchor point position has been recovered, the normal *RXWINSZ* value can be reused in order to avoid useless power consumption. Figure 3-43 shows such adaptation mechanism.

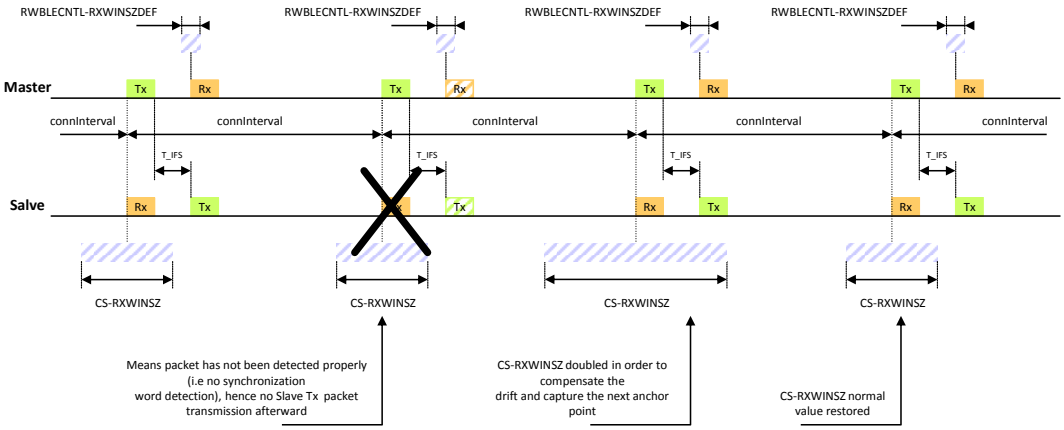

Figure 3-43 – Slave Rx Window during connection events

Note the RW-BLE Software is in charge of updating CS-RXWINSZ accordingly.

3.9 RF Test Modes

There are 3 main test modes supported by RW-BLE Core:

1. Tx test mode, that is Bluetooth RF qualification oriented for Tx modes
2. Rx test mode, that is Bluetooth RF qualification oriented for Rx modes
3. Tx/Rx test mode, that is Regulatory Body testing oriented

All these modes are set using CS-FORMAT. The packet format is based onto Advertising Channel packet format. In case of Tx test mode or Rx test mode, packet information is provided through the Control Structure. Please note that Access Address is defined as 0x71764129, Header packet type field defines data type contained in the payload (See Table 3-10) and Tx/RxADD are meaningless. Data are generated by the Packet Controller hence Tx Data Buffers are not used. Please refer [6] to for further details.

Packet Type	***Payload data***
0000	PRBS9 sequence
0001	Repeated 11110000 sequence
0010	Repeated 10101010 sequence
0011	PRBS15 sequence
0100	Repeated 11111111 sequence
0101	Repeated 00000000 sequence
0110	Repeated 00001111 sequence
0111	Repeated 01010101 sequence
1xxx	Reserved

Table 3-10 – Tx Test Mode Payload data Definition

In case of Rx test mode, data are processed without any specific check, only CRC error check ensures data integrity.
In case of Tx/Rx test modes, RW-BLE Software must define the packet (i.e. length, data type, data source, etc…) using RFTESTCNTL register fields. RWBLECNTL-RFTEST_ABORT is used to stop any of these test modes.
Note that for better PER estimate, Tx/Rx packets count can be obtained by setting respectively RFTESTCNTL-TXPKTCNTEN and/or RFTESTCNTL-RXPKTCNTEN. Tx/Rx Packet count results are provided in bot CS-<TX/RX>CCMPKTCNT fields over 40 bits, and RFTEST<TX/RX>STAT-<TX/RX>PKTCNT over 32 bits on abort command

3.9.1 Tx Test Mode timings

In case of Tx test mode, and when packet length is lower or equal than 37 bytes, Tx packets are sent each 625µs as shown in Figure 3-44.

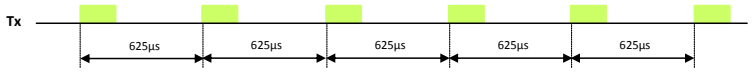

Figure 3-44 – Tx Test Mode packet scheduling

Packets sent are based onto Adverting Channel packet format, except that Access Address is specific, Header packet type field defines data type, and that Tx/RxADD are meaningless. Data are generated by the Packet Controller hence Tx Data Buffers are not used.

Note that the 625µs Tx time interval can be increased to higher values when packet length is bigger than 37 bytes (please refer to Table 2-22).

Please refer to [7] to for further details.

3.9.2 Rx Test Mode timings

In case of Rx test mode, and when packet length is lower or equal than 37 bytes, Rx packets are received each 625µs as shown in Figure 3-45

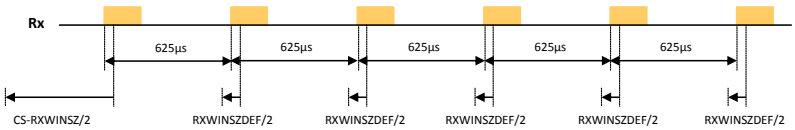

Figure 3-45 – Rx Test Mode packet scheduling

On first receipt, CS-RXWINSZ is used in order to capture the first incoming Rx packet. On next Rx packet, RWBLECNTL-RXWINSZDEF is used. Data are stored in Rx Data Buffers.

Note that the 625µs Rx time interval can be increased to higher values when decoded packet length is bigger than 37 bytes (please refer to Table 2-22).

Note also that in case a length error is reported (i.e when the decoded packet length is bigger than CS-RXMAXBUF), then the interval is considered as 625µs, hence It is recommended to the RW-BLE Software to program an Rx window greater than 2.5ms.

3.9.3 Tx/Rx Test Mode timings

In case of Tx/Rx test mode, the scheduling scheme follows master device scheme (i.e. sending first Tx packet, then receiving Rx packet, etc…) as shown in Figure 3-46.

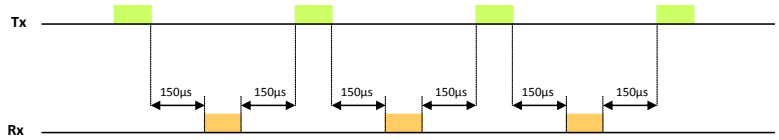

Figure 3-46 – Tx/Rx Test Mode packet scheduling

Tx Packets are defined as per RFTESTCNTL register. Rx packet payload is not stored in Rx Data buffers, only CRC is checked. Infinite Tx packet length and infinite Rx windows are also possible.

3.10 Interrupts Generation

The RW-BLE Core generates four interrupts that are used to synchronize with the RW-BLE Software. These four interrupts are:

- 625µs base time reference clock interrupt, available in active modes, conveyed through *ble_cscnt_irq* output pin
- Receipt interrupt at the end of either each CS-RXTHR number of received packets or each received packets, conveyed through *ble_rx_irq* output pin
- End of Sleep Mode events (see section 2.5.3) , conveyed through *ble_slp_irq* output pin
- End of Advertising / Scanning / Connection events, conveyed through *ble_event_irq* output pin.

Depending on the context, these interrupts are generated, or not. Note that grayed *ble_cscnt_irq* interrupt pulses in the following figures can be masked, or unmasked, according to INTCNTL-CSCNTDEVMSK value.

Figure 3-47 shows an example of interrupts generation for an advertiser device during advertising event. First advertising event shows advertising packet only. Second advertising event shows a scanner tried to exchange data with the advertiser device.

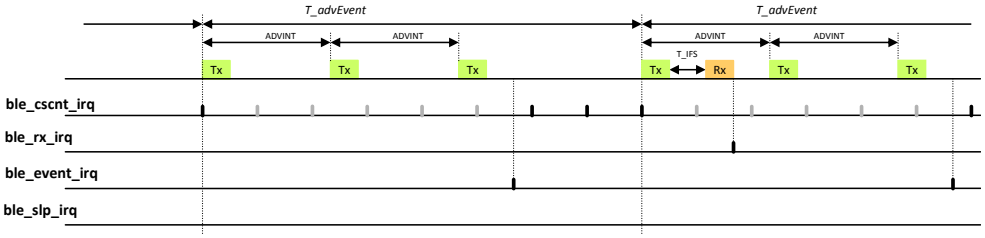

Figure 3-47 – Advertiser Device Interrupts Generation

Figure 3-48 shows an example of interrupts generation for a scanner device during scanning event. First scanning window shows a passive scan event onto channel 39. Second scanning window shows an active scan event with no scan response onto channel 37.

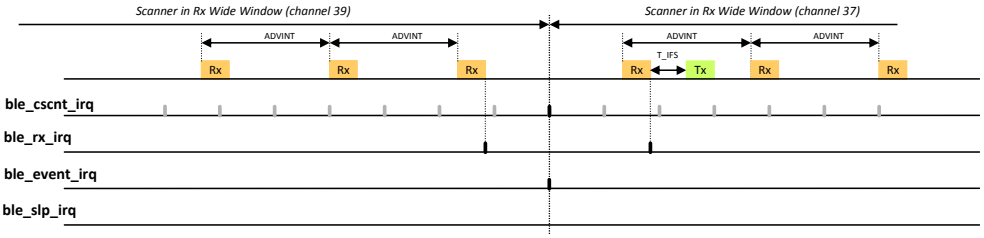

Figure 3-48 – Scanner Device Interrupts Generation

Figure 3-49 shows an example of interrupts generation for a master device during a Link Layer connection event without Deep Sleep in between anchor points. First and third connection event show two packets exchange, while second connection event shows 4 packets exchange.

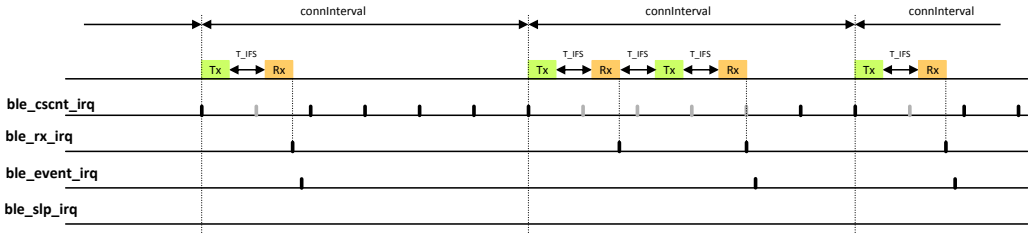

Figure 3-49 – Master Device Interrupts Generation / Link Layer Connection Event without Deep Sleep

Figure 3-50 shows an example of interrupts generation for a master device during a Link Layer connection event with Deep Sleep in between anchor points.

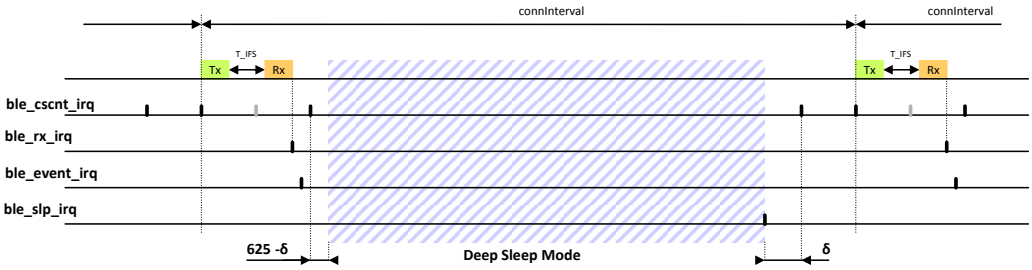

Figure 3-50 – Master Device Interrupts Generation / Link Layer Connection Event with Deep Sleep

Figure 3-51 shows an example of interrupts generation for a slave device during a Link Layer connection event without Deep Sleep in between anchor points.

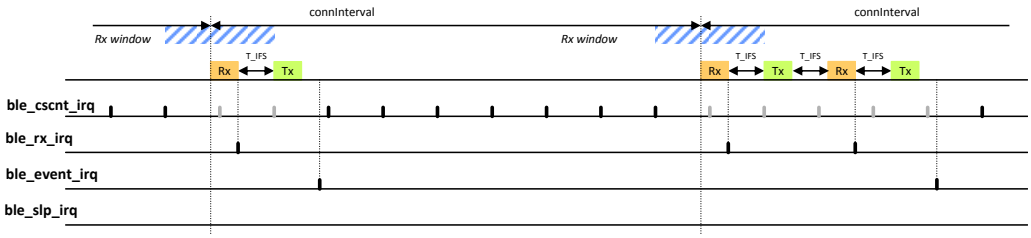

Figure 3-51 – Slave Device Interrupts Generation / Link Layer Connection Event without Deep Sleep

Figure 3-52 shows an example of interrupts generation for a slave device during a Link Layer connection event with Deep Sleep in between anchor points.

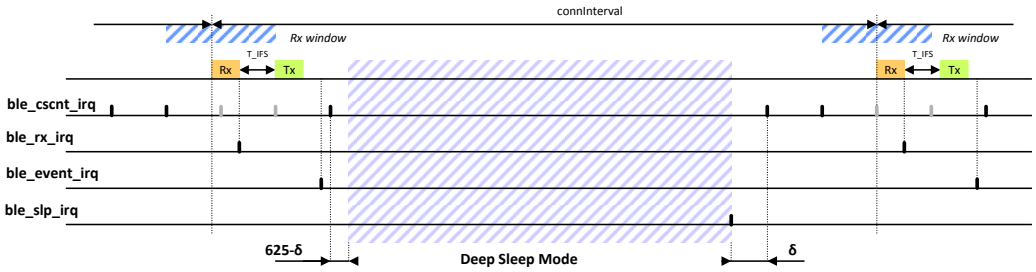

Figure 3-52 – Slave Device Interrupts Generation / Link Layer Connection Event with Deep Sleep

In addition the RW-BLE Core offers three specific interrupts that are: an error interrupt generated onto internal error (e.g. Exchange Memory access collision, etc...) and two general purpose timer interrupts that can be used by the RW-BLE Software (e.g. for Supervision timeout as example).

3.11 Hardware / Software handshaking

Hardware / Software handshaking is performed through the different interrupt lines. Depending on the context, different register and the exchange memory are read and/or updated.

To be able to know which Exchange Table pointer is read by the RW-BLE Core, RW-BLE Software must use *ble_cscnt_irq* interrupt, and then can read BASETIMECNT-BASETIMECNT field. It is possible then to synchronize RW-BLE Software and RW-BLE Core, and then to be able to maintain connection interval timing during connection event as example. It is used for Exchange Table programming.

To be able to know when a packet has been received, RW-BLE Software must wait for *ble_rx_irq* to occur. Then RW-BLE Software reads related Control Structure, checks the protocol fields, reads Rx data buffers, and decides whether Tx buffers need to be updated for the next event if necessary.

To be able to know when an event is finished, RW-BLE Software must wait for *ble_event_irq* to occur. Then RW-BLE Software read related Control Structure, check the protocol fields, read Rx data buffers, and decides whether Tx buffers need to be updated for the next event if necessary. It is also useful in order to save the context if a Deep Sleep is scheduled.

To be able to wake-up the system and to sort out from Deep Sleep mode, RW-BLE Software must wait for *ble_slp_irq* to occur. Then RW-BLE Software can restore the context and schedule the next connection event.

ble_sw_irq is generated on SW request, and it is used to force some SW tasks to run under interruption, in order to ensure it will not be stopped or delayed due to other activities.

3.12 Registers

This section groups the internal registers of the RW-BLE Core. They can be accessed at any time by the software.

All unused bits are read as 0. If the processor writes any value in these locations, it is ignored.

3.12.1 Standard registers

This set of registers is present in all versions of the RW-BLE Core, and are not configurable at synthesis time.

Address	Access		RWBLECNTL																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+00'H	R/W	R/W	MASTER_SOFT_RST	MASTER_TGSOFT_RST	REG_SOFT_RST	SWINT_REQ		RFTEST_ABORT	ADVERT_ABORT	SCAN_ABORT		MD_DSB	SN_DSB	NESN_DSB	CRYPT_DSB	WHIT_DSB	CRC_DSB	HOP_REMAP_DSB							ADVERTFILT_EN	RWBLE_EN	RXWINSZDEF[3]	RXWINSZDEF[2]	RXWINSZDEF[1]	RXWINSZDEF[0]		SYNCERR[2]	SYNCERR[1]	SYNCERR[0]
Reset value			0	0	0	0		0	0	0		0	0	0	0	0	0	0							0	0	0	0	0	0		0	0	0
Type			U	U	U	U		U	U	U		U	U	U	U	U	U	U							U	U	U	U	U	U		U	U	U
HW Access			R/W	R/W	R	R/W		R/W	R/W	R/W		R	R	R	R	R	R	R							R	R	R	R	R	R		R	R	R
SW Access			S	S	R/W	S		S	S	S		R/W	R/W	R/W	R/W	R/W	R/W	R/W							R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W
Verification			DT	DT	DT	DT		R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W							R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W

Register 3-1 – RWBLECNTL

MASTER_SOFT_RST	Reset the complete BLE Core except registers and timing generator, when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0. In case of Dual Mode implementation, reset also common blocks.
MASTER_TGSOFT_RST	Reset the timing generator, when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
REG_SOFT_RST	Reset the complete register block, when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
SWINT_REQ	Forces the generation of *ble_sw_irq* when written with a 1, and proper masking is set. Resets at 0 when action is performed. No action happens if it is written with 0.
RFTEST_ABORT	Abort the current RF Testing defined as per CS-FORMAT when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0. Note that when RFTEST_ABORT is requested 1/ In case of infinite Tx, the Packet Controller FSM stops at the end of the current byte in process, and processes accordingly the packet CRC. 2/ In case of Infinite Rx, the Packet Controller FSM either stops as the end of the current Packet reception (if Access address has been detected), or simply stop the processing switching off the RF.
SCAN_ABORT	Abort the current scan window when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
ADVERT_ABORT	Abort the current Advertising event when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
MD_DSB	0: Normal operation of MD bits management 1: Allow a single Tx/Rx exchange whatever the MD bits are. - value forced by SW from Tx Descriptor - value just saved in Rx Descriptor during reception
SN_DSB	0: Normal operation of Sequence number 1: Sequence Number Management disabled:

	- value forced by SW from Tx Descriptor - value ignored in Rx ➔ No SN error reported.
NESN_DSB	0: Normal operation of Acknowledge 1: Acknowledge scheme disabled: - value forced by SW from Tx Descriptor - value ignored in Rx ➔ No NESN error reported.
CRYPT_DSB	0: Normal operation. Encryption / Decryption enabled. 1: Encryption / Decryption disabled. Note that if CS-CRYPT_EN is set, then MIC is generated, and only data encryption is disabled, meaning data sent are plain data.
WHIT_DSB	0: Normal operation. Whitening enabled. 1: Whitening disabled.
CRC_DSB	0: Normal operation. CRC removed from data stream. 1: CRC stripping disabled on Rx packets, CRC replaced by 0x000 in Tx.
HOP_REMAP_DSB	0: Normal operation. Frequency Hopping Remapping algorithm enabled. 1: Frequency Hopping Remapping algorithm disabled
ADVERTFILT_EN	Advertising Channels Error Filtering Enable control 0: RW-BLE Core reports all errors to RW-BLE Software 1: RW-BLE Core reports only correctly received packet, without error to RW-BLE Software
RWBLE_EN	0: Disable RW-BLE Core Exchange Table pre-fetch mechanism. 1: Enable RW-BLE Core Exchange table pre-fetch mechanism..
RXWINSZDEF[3:0]	Default Rx Window size in µs. Used when device - is master connected - performs its second receipt. '0 'is not a valid value. Recommended value is 10 (in decimal).
SYNCERR[2:0]	Indicates the maximum number of errors allowed to recognize the synchronization word.

Address	Access		VERSION																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+04'H	F	R	TYP[7]	TYP[6]	TYP[5]	TYP[4]	TYP[3]	TYP[2]	TYP[1]	TYP[0]	REL[7]	REL[6]	REL[5]	REL[4]	REL[3]	REL[2]	REL[1]	REL[0]	UPG[7]	UPG[6]	UPG[5]	UPG[4]	UPG[3]	UPG[2]	UPG[1]	UPG[0]	BUILD[7]	BUILD[6]	BUILD[5]	BUILD[4]	BUILD[3]	BUILD[2]	BUILD[1]	BUILD[0]
Reset value			0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-2 – VERSION

TYP[7:0]	RW-BLE Core Type – 0x8 means BLE v4.2 (i.e. correspond LL version assigned number). Correspond to FS v**8**.0.05)
REL[7:0]	RW-BLE Core version – Major release number. Correspond to FS v8.**0**.05
UPG[7:0]	RW-BLE Core upgrade – Upgrade number. Correspond to FS v8.0.**05**
BUILD[7:0]	RW-BLE Core Build – Build number

The version information is set as a constant in the RTL code. It must be modified for each specific implementation. The code to be placed in this register is given by RivieraWaves for each implementation in a product.

Address	Access		RWBLECONF																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+08'H	F	R	DMMODE								DECIPHER		WLANCOEX	RFIF[4]	RFIF[3]	RFIF[2]	RFIF[1]	RFIF[0]	USEDBG	USECRYPT	CLK_SEL[5]	CLK_SEL[4]	CLK_SEL[3]	CLK_SEL[2]	CLK_SEL[1]	CLK_SEL[0]	INTMODE	BUS_TYPE	DATA_WIDTH	ADDR_WIDTH[4]	ADDR_WIDTH[3]	ADDR_WIDTH[2]	ADDR_WIDTH[1]	ADDR_WIDTH[0]
Reset value			0								1		1	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0	0	1	1	0	1
Type			U								U		U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			F								F		F	F	F	F	F	F	F	F	W	W	W	W	W	W	F	F	F	F	F	F	F	F
SW Access			R								R		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R								R		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-3 – RWBLECONF

DMMODE	0: RW-BLE Core is used as a standalone BLE device 1: RW-BLE Core is used in a Dual Mode device
DECIPHER	0: AES deciphering not present 1: AES deciphering present
WLANCOEX	0: WLAN Coexistence mechanism not present 1: WLAN Coexistence mechanism present (Default Value)
RFIF[4:0]	RFIF[k]= 0: Control logic supporting radio k not present RFIF[k]= 1: Control logic supporting radio k present Index k values are: 00001: Ripple RF. 00010: External Radio Controller Support 00100: IcyTRx Radio xxx000: Reserved Default value is 0000001
USEDBG	0: Diagnostic port not instantiated 1: Diagnostic port instantiated (Default Value)
USECRYPT	0: AES-CCM Encryption block not present 1: AES-CCM Encryption block present (Default Value)
CLK_SEL[5:0]	Operating Frequency (in MHz) Default value is 13MHz
INTMODE	0: Interrupts are edge level generated, i.e. pulse. 1: Interrupts are trigger level generated, i.e. stays active at 1 till acknowledgement (Default Value)
BUSTYPE	Processor Bus Type 0: AHB Bus 1: X-Bar Bus
DATA_WIDTH	Processor bus width: 0: 16 bits (Default Value) 1: 32 bits
ADDR_WIDTH[4:0]	Value of the *RW_BLE_ADDRESS_WIDTH* parameter concerted into binary. Default value is 13 (in decimal)

Please note this register supplies the actual status of the configurable features.

Address	Access		INTCNTL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+0C'H	R	R/W																	CSCNTDEVMSK						SWINTMSK	EVENTAPFAINTMSK	FINETGTIMINTMSK	GROSSTGTIMINTMSK	ERRORINTMSK	CRYPTINTMSK	EVENTINTMSK	SLPINTMSK	RXINTMSK	CSCNTINTMSK
Reset value																			1						0	1	0	0	0	1	1	1	1	1
Type																			U						U	U	U	U	U	U	U	U	U	U
HW Access																			R						R	R	R	R	R	R	R	R	R	R
SW Access																			R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-4 – INTCNTL

CSCNTDEVMSK	CSCNT interrupt mask during event. This bit allows to enable CSCNT interrupt generation during events (i.e. advertising, scanning, initiating, and connection) 0: CSCNT Interrupt not generated during events. 1: CSCNT Interrupt generated during events.
SWINTMSK	SW triggered interrupt Mask 0: Interrupt not generated 1: Interrupt generated
EVENTAPFAINTMSK	End of event / anticipated pre-fetch abort interrupt Mask 0: Interrupt not generated 1: Interrupt generated
FINETGTIMINTMSK	Fine Target Timer Mask 0: Interrupt not generated 1: Interrupt generated
GROSSTGTIMINTMSK	Gross Target Timer Mask 0: Interrupt not generated 1: Interrupt generated
ERRORINTMSK	Error Interrupt Mask 0: Interrupt not generated 1: Interrupt generated
CRYPTINTMSK	Encryption engine Interrupt Mask 0: Interrupt not generated 1: Interrupt generated
EVENTINTMSK	End of event Interrupt Mask 0: Interrupt not generated 1: Interrupt generated
SLPINTMSK	Sleep Mode Interrupt Mask 0: Interrupt not generated 1: Interrupt generated
RXINTMSK	Rx Interrupt Mask 0: Interrupt not generated 1: Interrupt generated
CSCNTINTMSK	625µs Base Time Interrupt Mask 0: Interrupt not generated 1: Interrupt generated

Address	Access		INTSTAT																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+10'H	W	R																							SWINTSTAT	EVENTAPFAINTSTAT	FINETGTIMINTSTAT	GROSSTGTIMINTSTAT	ERRORINTSTAT	CRYPTINTSTAT	EVENTINTSTAT	SLPINTSTAT	RXINTSTAT	CSCNTINTSTAT
Reset value																									0	0	0	0	0	0	0	0	0	0
Type																									U	U	U	U	U	U	U	U	U	U
HW Access																									W	W	W	W	W	W	W	W	W	W
SW Access																									R	R	R	R	R	R	R	R	R	R
Verification																									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-5 – INTSTAT

SWINTSTAT	SW triggered interrupt status 0: No SW triggered interrupt. 1: A SW triggered interrupt is pending.
EVENTAPFAINTSTAT	End of event / Anticipated Pre-Fetch Abort interrupt status 0: No End of Event interrupt. 1: An End of Event interrupt is pending.
FINETGTIMINTSTAT	Masked Fine Target Timer Error interrupt status 0: No Fine Target Timer interrupt. 1: A Fine Target Timer interrupt is pending.
GROSSTGTIMINTSTAT	Masked Gross Target Timer interrupt status 0: No Gross Target Timer interrupt. 1: A Gross Target Timer interrupt is pending.
ERRORINTSTAT	Masked Error interrupt status 0: No Error interrupt. 1: An Error interrupt is pending.
CRYPTINTSTAT	Masked Encryption engine interrupt status 0: No Encryption / Decryption interrupt. 1: An Encryption / Decryption interrupt is pending.
EVENTINTSTAT	Masked End of Event interrupt status 0: No End of Advertising / Scanning / Connection interrupt. 1: An End of Advertising / Scanning / Connection interrupt is pending.
SLPINTSTAT	Masked Sleep interrupt status 0: No End of Sleep Mode interrupt. 1: An End of Sleep Mode interrupt is pending.
RXINTSTAT	Masked Packet Reception interrupt status 0: No Rx interrupt. 1: An Rx interrupt is pending.
CSCNTINTSTAT	Masked 625µs base time reference interrupt status 0: No 625µs Base Time interrupt. 1: A 625µs Base Time interrupt is pending.

Address	Access		INTRAWSTAT																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+14'H	W	R																							SWINTRAWSTAT	EVENTAPFAINTRAWSTAT	FINETGTIMINTRAWSTAT	GROSSTGTIMINTRAWSTAT	ERRORINTRAWSTAT	CRYPTINTRAWSTAT	EVENTINTRAWSTAT	SLPINTRAWSTAT	RXINTRAWSTAT	CSCNTINTRAWSTAT
Reset value																									0	0	0	0	0	0	0	0	0	0
Type																									U	U	U	U	U	U	U	U	U	U
HW Access																									W	W	W	W	W	W	W	W	W	W
SW Access																									R	R	R	R	R	R	R	R	R	R
Verification																									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-6 – INTRAWSTAT

SWINTRAWSTAT	SW triggered interrupt raw status 0: No SW triggered interrupt. 1: A SW triggered interrupt is pending.
EVENTAPFAINTRAWSTAT	End of event / Anticipated Pre-Fetch Abort interrupt raw status 0: No End of Event interrupt. 1: An End of Event interrupt is pending.
FINETGTIMINTRAWSTAT	Fine Target Timer Error interrupt raw status 0: No Fine Target Timer interrupt. 1: A Fine Target Timer interrupt is pending.
GROSSTGTIMINTRAWSTAT	Gross Target Timer interrupt raw status 0: No Gross Target Timer interrupt. 1: A Gross Target Timer interrupt is pending.
ERRORINTRAWSTAT	Error interrupt raw status 0: No Error interrupt. 1: An Error interrupt is pending.
CRYPTINTRAWSTAT	Encryption engine interrupt raw status 0: No Encryption / Decryption interrupt. 1: An Encryption / Decryption interrupt is pending.
EVENTINTRAWSTAT	End of Event interrupt raw status 0: No End of Advertising / Scanning / Connection interrupt. 1: An End of Advertising / Scanning / Connection interrupt is pending.
SLPINTRAWSTAT	Sleep interrupt raw status 0: No End of Sleep Mode interrupt. 1: An End of Sleep Mode interrupt is pending.
RXINTRAWSTAT	Packet Reception interrupt raw status 0: No Rx interrupt. 1: An Rx interrupt is pending.
CSCNTINTRAWSTAT	625µs base time reference interrupt raw status 0: No 625µs Base Time interrupt. 1: A 625µs Base Time interrupt is pending.

Address	Access		INTACK																									
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+18'H	R	C																							SWINTACK	EVENTAPFAINTACK	FINETGTIMINTACK	GROSSTGTIMINTACK	ERRORINTACK	CRYPTINTACK	EVENTINTACK	SLPINTACK	RXINTACK	CSCNTINTACK
Reset value																									0	0	0	0	0	0	0	0	0	0
Type																									U	U	U	U	U	U	U	U	U	U
HW Access																									R	R	R	R	R	R	R	R	R	R
SW Access																									C	C	C	C	C	C	C	C	C	C
Verification																									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-7 – INTACK

SWINTACK	SW triggered interrupt acknowledgement bit Software writing 1 acknowledges the SW triggered interrupt. This bit resets SWINTSTAT and SWINTRAWSTAT flags. Resets at 0 when action is performed
EVENTAPFAINTACK	End of event / Anticipated Pre-Fetch Abort interrupt acknowledgement bit Software writing 1 acknowledges the End of event / Anticipated Pre-Fetch Abort interrupt. This bit resets EVENTAPFAINTSTAT and EVENTAPFAINTRAWSTAT flags. Resets at 0 when action is performed
FINETGTIMINTACK	Fine Target Timer interrupt acknowledgement bit Software writing 1 acknowledges the Fine Timer interrupt. This bit resets FINETGTIMINTSTAT and FINETGTIMINTRAWSTAT flags. Resets at 0 when action is performed
GROSSTGTIMINTACK	Gross Target Timer interrupt acknowledgement bit Software writing 1 acknowledges the Gross Timer interrupt. This bit resets GROSSTGTIMINTSTAT and GROSSTGTIMINTRAWSTAT flags. Resets at 0 when action is performed
ERRORINTACK	Error interrupt acknowledgement bit Software writing 1 acknowledges the Error interrupt. This bit resets ERRORINTSTAT and ERRORINTRAWSTAT flags. Resets at 0 when action is performed
CRYPTINTACK	Encryption engine interrupt acknowledgement bit Software writing 1 acknowledges the Encryption engine interrupt. This bit resets CRYPTINTSTAT and CRYPTINTRAWSTAT flags. Resets at 0 when action is performed
EVENTINTACK	End of Event interrupt acknowledgment bit Software writing 1 acknowledges the End of Advertising / Scanning / Connection interrupt. This bit resets SLPINTSTAT and SLPINTRAWSTAT flags. Resets at 0 when action is performed
SLPINTACK	End of Deep Sleep interrupt acknowledgment bit Software writing 1 acknowledges the End of Sleep Mode interrupt. This bit resets SLPINTSTAT and SLPINTRAWSTAT flags. Resets at 0 when action is performed
RXINTACK	Packet Reception interrupt acknowledgment bit Software writing 1 acknowledges the Rx interrupt. This bit resets RXINTSTAT and RXINTRAWSTAT flags. Resets at 0 when action is performed
CSCNTINTACK	625µs base time reference interrupt acknowledgment bit Software writing 1 acknowledges the CLKN interrupt. This bit resets CLKINTSTAT and CLKINTRAWSTAT flags. Resets at 0 when action is performed

Address	Access		BASETIMECNT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+1C'H	R/W	R/W	SAMP					BASETIMECNT[26]	BASETIMECNT[25]	BASETIMECNT[24]	BASETIMECNT[23]	BASETIMECNT[22]	BASETIMECNT[21]	BASETIMECNT[20]	BASETIMECNT[19]	BASETIMECNT[18]	BASETIMECNT[17]	BASETIMECNT[16]	BASETIMECNT[15]	BASETIMECNT[14]	BASETIMECNT[13]	BASETIMECNT[12]	BASETIMECNT[11]	BASETIMECNT[10]	BASETIMECNT[9]	BASETIMECNT[8]	BASETIMECNT[7]	BASETIMECNT[6]	BASETIMECNT[5]	BASETIMECNT[4]	BASETIMECNT[3]	BASETIMECNT[2]	BASETIMECNT[1]	BASETIMECNT[0]
Reset value			0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U					U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R/W					W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			S					R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R/W					R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-8 – BASETIMECNT

SAMP	Writing a 1 samples the Base Time Counter value in BASETIMECNT register field. Resets at 0 when action is performed
BASETIMECNT[26:0]	Value of the 625µs base time reference counter. Updated each time SAMP field is written. Used by the SW in order to synchronize with the HW

Address	Access		FINETIMECNT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+20'H	W	R																							FINECNT[9]	FINECNT[8]	FINECNT[7]	FINECNT[6]	FINECNT[5]	FINECNT[4]	FINECNT[3]	FINECNT[2]	FINECNT[1]	FINECNT[0]
Reset value																									0	0	0	0	0	0	0	0	0	0
Type																									U	U	U	U	U	U	U	U	U	U
HW Access																									W	W	W	W	W	W	W	W	W	W
SW Access																									R	R	R	R	R	R	R	R	R	R
Verification																									R	R	R	R	R	R	R	R	R	R

Register 3-9 – FINETIMECNT

FINECNT[9:0]	Value of the current µs fine time reference counter. Updated each time SAMP field is written. Used by the SW in order to synchronize with the HW, and obtain a more precise sleep duration

Address	Access		BDADDRL																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+24'H	R	R/W	BDADDRL[31]	BDADDRL[30]	BDADDRL[29]	BDADDRL[28]	BDADDRL[27]	BDADDRL[26]	BDADDRL[25]	BDADDRL[24]	BDADDRL[23]	BDADDRL[22]	BDADDRL[21]	BDADDRL[20]	BDADDRL[19]	BDADDRL[18]	BDADDRL[17]	BDADDRL[16]	BDADDRL[15]	BDADDRL[14]	BDADDRL[13]	BDADDRL[12]	BDADDRL[11]	BDADDRL[10]	BDADDRL[9]	BDADDRL[8]	BDADDRL[7]	BDADDRL[6]	BDADDRL[5]	BDADDRL[4]	BDADDRL[3]	BDADDRL[2]	BDADDRL[1]	BDADDRL[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-10 – BDADDRL

BDADDRL[31:0]	Bluetooth Low Energy Device Address. LSB part.

Address	Access		BDADDRU																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+28'H	R	R/W																PRIV_NPUB	BDADDRU[15]	BDADDRU[14]	BDADDRU[13]	BDADDRU[12]	BDADDRU[11]	BDADDRU[10]	BDADDRU[9]	BDADDRU[8]	BDADDRU[7]	BDADDRU[6]	BDADDRU[5]	BDADDRU[4]	BDADDRU[3]	BDADDRU[2]	BDADDRU[1]	BDADDRU[0]
Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																		U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-11 – BDADDRU

PRIV_NPUB	Bluetooth Low Energy Device Address privacy indicator 0: Public Bluetooth Device Address 1: Private Bluetooth Device Address
BDADDRU[15:0]	Bluetooth Low Energy Device Address. MSB part.

Address	Access		ET_CURRENTRXDESCPTR																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+2C'H	R/W	R/W	ETPTR[15]	ETPTR[14]	ETPTR[13]	ETPTR[12]	ETPTR[11]	ETPTR[10]	ETPTR[9]	ETPTR[8]	ETPTR[7]	ETPTR[6]	ETPTR[5]	ETPTR[4]	ETPTR[3]	ETPTR[2]	ETPTR[1]	ETPTR[0]		CURRENTRXDESCPTR[14]	CURRENTRXDESCPTR[13]	CURRENTRXDESCPTR[12]	CURRENTRXDESCPTR[11]	CURRENTRXDESCPTR[10]	CURRENTRXDESCPTR[9]	CURRENTRXDESCPTR[8]	CURRENTRXDESCPTR[7]	CURRENTRXDESCPTR[6]	CURRENTRXDESCPTR[5]	CURRENTRXDESCPTR[4]	CURRENTRXDESCPTR[3]	CURRENTRXDESCPTR[2]	CURRENTRXDESCPTR[1]	CURRENTRXDESCPTR[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U		U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-12 – ET_CURRENTRXDESCPTR

ETPTR[15:0]	Exchange Table Pointer that determines the starting point of the Exchange Table
CURRENTRXDESCPTR[14:0]	Rx Descriptor Pointer that determines the starting point of the Receive Buffer Chained List

3.12.2 Deep sleep registers

The following registers are only implemented if the low power controller is present in the RW-BLE Core.

Address	Access		DEEPSLCNTL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+30'H	R/W	R/W	EXTWKUPDSB																DEEP_SLEEP_STAT											SOFT_WAKEUP_REQ	DEEP_SLEEP_CORR_EN	DEEP_SLEEP_ON	RADIO_SLEEP_EN	OSC_SLEEP_EN
Reset value			0																0											0	0	0	0	0
Type			U																U											U	U	U	U	U
HW Access			R																W											R/W	R/W	R/W	R	R
SW Access			R/W																R											S	S	S	R/W	R/W
Verification			R/W																R/W											R/W	R/W	R/W	R/W	R/W

Register 3-13 – DEEPSLCNTL

EXTWKUPDSB	External Wake-Up disable 0: RW-BLE Core can be woken by external wake-up 1: RW-BLE Core cannot be woken up by external wake-up
DEEP_SLEEP_STAT	Indicator of current Deep Sleep clock mux status: 0: RW-BLE Core is not yet in Deep Sleep Mode 1: RW-BLE Core is in Deep Sleep Mode (only *low_power_clk* is running)
SOFT_WAKEUP_REQ	Wake Up Request from RW-BLE Software. Applies when system is in Deep Sleep Mode. It wakes up the RW-BLE Core when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
DEEP_SLEEP_CORR_EN	625µs base time reference integer and fractional part correction. Applies when system has been woken-up from Deep Sleep Mode. It enables Fine Counter and Base Time counter when written with a 1. Resets at 0 when action is performed. No action happens if it is written with 0.
DEEP_SLEEP_ON	0: RW-BLE Core in normal active mode 1: Request RW-BLE Core to switch in deep sleep mode. This bit is reset on DEEP_SLEEP_STAT falling edge.
RADIO_SLEEP_EN	Controls the Radio module 0: Radio stands in normal active mode 1: Allow to disable Radio
OSC_SLEEP_EN	Controls the RF High frequency crystal oscillator 0: High frequency crystal oscillator stands in normal active mode 1: Allow to disable High frequency crystal oscillator

Note the register is reset at the end of the sleep phase. Not to be written until completion of the sleep phase.

Address	Access		DEEPSLWKUP																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+34'H	R	R/W	DEEPSLTIME[31]	DEEPSLTIME[30]	DEEPSLTIME[29]	DEEPSLTIME[28]	DEEPSLTIME[27]	DEEPSLTIME[26]	DEEPSLTIME[25]	DEEPSLTIME[24]	DEEPSLTIME[23]	DEEPSLTIME[22]	DEEPSLTIME[21]	DEEPSLTIME[20]	DEEPSLTIME[19]	DEEPSLTIME[18]	DEEPSLTIME[17]	DEEPSLTIME[16]	DEEPSLTIME[15]	DEEPSLTIME[14]	DEEPSLTIME[13]	DEEPSLTIME[12]	DEEPSLTIME[11]	DEEPSLTIME[10]	DEEPSLTIME[9]	DEEPSLTIME[8]	DEEPSLTIME[7]	DEEPSLTIME[6]	DEEPSLTIME[5]	DEEPSLTIME[4]	DEEPSLTIME[3]	DEEPSLTIME[2]	DEEPSLTIME[1]	DEEPSLTIME[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-14 – DEEPSLWKUP

DEEPSLTIME[31:0]	Determines the time in *low_power_clk* clock cycles to spend in Deep Sleep Mode before waking-up the device. This ensures a maximum of 37 hours and 16mn sleep mode capabilities at 32kHz. This ensures a maximum of 36 hours and 16mn sleep mode capabilities at 32.768kHz

Please note:

1. If DEEPSLTIME is set to zero, the Deep Sleep Time duration is considered as infinite, and only wake up requests can restore active behavior.
2. RW-BLE Software must ensure DEEPSLTIME value to be greater than 2 in order to cope with control resynchronization requirements in between *master1_gclk* and *low_power_clk* clock domains.

Address	Access		DEEPSLSTAT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+38'H	W	R	DEEPSLDUR[31]	DEEPSLDUR[30]	DEEPSLDUR[29]	DEEPSLDUR[28]	DEEPSLDUR[27]	DEEPSLDUR[26]	DEEPSLDUR[25]	DEEPSLDUR[24]	DEEPSLDUR[23]	DEEPSLDUR[22]	DEEPSLDUR[21]	DEEPSLDUR[20]	DEEPSLDUR[19]	DEEPSLDUR[18]	DEEPSLDUR[17]	DEEPSLDUR[16]	DEEPSLDUR[15]	DEEPSLDUR[14]	DEEPSLDUR[13]	DEEPSLDUR[12]	DEEPSLDUR[11]	DEEPSLDUR[10]	DEEPSLDUR[9]	DEEPSLDUR[8]	DEEPSLDUR[7]	DEEPSLDUR[6]	DEEPSLDUR[5]	DEEPSLDUR[4]	DEEPSLDUR[3]	DEEPSLDUR[2]	DEEPSLDUR[1]	DEEPSLDUR[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-15 – DEEPSLSTAT

DEEPSLDUR[31:0]	Actual duration of the last deep sleep phase measured in *low_power_clk* clock cycle. DEEPSLDUR is set to zero at the beginning of the deep sleep phase, and is incremented at each *low_power_clk* clock cycle until the end of the deep sleep phase.

Address	Access		ENBPRESET																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+3C'H	R	R/W	TWEXT[10]	TWEXT[9]	TWEXT[8]	TWEXT[7]	TWEXT[6]	TWEXT[5]	TWEXT[4]	TWEXT[3]	TWEXT[2]	TWEXT[1]	TWEXT[0]	TWOSC[10]	TWOSC[9]	TWOSC[8]	TWOSC[7]	TWOSC[6]	TWOSC[5]	TWOSC[4]	TWOSC[3]	TWOSC[2]	TWOSC[1]	TWOSC[0]	TWRM[9]	TWRM[8]	TWRM[7]	TWRM[6]	TWRM[5]	TWRM[4]	TWRM[3]	TWRM[2]	TWRM[1]	TWRM[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-16 – ENBPRESET

TWEXT[10:0]	Time in low power oscillator cycles allowed for stabilization of the high frequency oscillator following an external wake–up request (signal wakeup_req) [0…64ms] for 32kHz; [0…62.5ms] for 32.768kHz
TWOSC[10:0]	Time in low power oscillator cycles allowed for stabilization of the high frequency oscillator when the deep–sleep mode has been left due to sleep–timer expiry (DEEPSLWKUP-DEEPSLTIME]) [0…64ms] for 32kHz; [0…62.5ms] for 32.768kHz
TWRM[9:0]	Time in low power oscillator cycles allowed for the radio module to leave low–power mode [0…32ms] for 32kHz; [0…31.25ms] for 32.768kHz

Note RW-BLE Software must ensure T_{WRM} is lower than, or equal to, T_{WEXT} / T_{WOSC} for normal operation. Note that having all values set to 0 is considered as a valid setting.

Address	Access		FINECNTCORR																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+40'H	R	R/W																							FINECNTCORR[9]	FINECNTCORR[8]	FINECNTCORR[7]	FINECNTCORR[6]	FINECNTCORR[5]	FINECNTCORR[4]	FINECNTCORR[3]	FINECNTCORR[2]	FINECNTCORR[1]	FINECNTCORR[0]
Reset value																									0	0	0	0	0	0	0	0	0	0
Type																									U	U	U	U	U	U	U	U	U	U
HW Access																									R	R	R	R	R	R	R	R	R	R
SW Access																									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																									R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-17 – FINECNTCORR

FINECNTCORR[9:0]	Phase correction value for the 625µs reference counter (i.e. Fine Counter) in µs.

Address	Access		BASETIMECNTCORR																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+44'H	R	R/W						BASETIMECNTCORR[26]	BASETIMECNTCORR[25]	BASETIMECNTCORR[24]	BASETIMECNTCORR[23]	BASETIMECNTCORR[22]	BASETIMECNTCORR[21]	BASETIMECNTCORR[20]	BASETIMECNTCORR[19]	BASETIMECNTCORR[18]	BASETIMECNTCORR[17]	BASETIMECNTCORR[16]	BASETIMECNTCORR[15]	BASETIMECNTCORR[14]	BASETIMECNTCORR[13]	BASETIMECNTCORR[12]	BASETIMECNTCORR[11]	BASETIMECNTCORR[10]	BASETIMECNTCORR[9]	BASETIMECNTCORR[8]	BASETIMECNTCORR[7]	BASETIMECNTCORR[6]	BASETIMECNTCORR[5]	BASETIMECNTCORR[4]	BASETIMECNTCORR[3]	BASETIMECNTCORR[2]	BASETIMECNTCORR[1]	BASETIMECNTCORR[0]
Reset value								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type								U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access								R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-18 – BASETIMECNTCORR

BASETIMECNTCORR[26:0]	Base Time Counter correction value.

3.12.3 Validation registers

These registers are implemented only if *RW_BLE_DIAG_INST* is defined. This is chosen at synthesis time.

Address	Access		DIAGCNTL																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+50'H	R	R/W	DIAG3_EN		DIAG3[5]	DIAG3[4]	DIAG3[3]	DIAG3[2]	DIAG3[1]	DIAG3[0]	DIAG2_EN		DIAG2[5]	DIAG2[4]	DIAG2[3]	DIAG2[2]	DIAG2[1]	DIAG2[0]	DIAG1_EN		DIAG1[5]	DIAG1[4]	DIAG1[3]	DIAG1[2]	DIAG1[1]	DIAG1[0]	DIAG0_EN		DIAG0[5]	DIAG0[4]	DIAG0[3]	DIAG0[2]	DIAG0[1]	DIAG0[0]
Reset value			0		0	0	0	0	0	0	0		0	0	0	0	0	0	0		0	0	0	0	0	0	0		0	0	0	0	0	0
Type			U		U	U	U	U	U	U	U		U	U	U	U	U	U	U		U	U	U	U	U	U	U		U	U	U	U	U	U
HW Access			R		R	R	R	R	R	R	R		R	R	R	R	R	R	R		R	R	R	R	R	R	R		R	R	R	R	R	R
SW Access			R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W

Register 3-19 – DIAGCNTL

DIAG0_EN	'0:' Disable diagnostic port 0 output. All outputs are set to 0x0. 1: Enable diagnostic port 0 output.
DIAG0[5:0]	***Only relevant when DIAGEN0 = 1.*** Selection of the outputs that must be driven to the diagnostic port 0. See section 2.16 for a detailed description.
DIAG1_EN	0: Disable diagnostic port 1 output. All outputs are set to 0x0. 1: Enable diagnostic port 1 output.
DIAG1[5:0]	***Only relevant when DIAGEN1 = 1.*** Selection of the outputs that must be driven to the diagnostic port 1. See section 2.16 for a detailed description.
DIAG2_EN	0: Disable diagnostic port 2 output. All outputs are set to 0x0. 1: Enable diagnostic port 2 output.
DIAG2[5:0]	***Only relevant when DIAGEN2 = 1.*** Selection of the outputs that must be driven to the diagnostic port 2. See section 2.16 for a detailed description.
DIAG3_EN	0: Disable diagnostic port 3 output. All outputs are set to 0x0. 1: Enable diagnostic port 3 output.
DIAG3[5:0]	***Only relevant when DIAGEN3 = 1.*** Selection of the outputs that must be driven to the diagnostic port 3. See section 2.16 for a detailed description.

Address	Access		DIAGSTAT																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+54'H	W	R	DIAG3STAT[7]	DIAG3STAT[6]	DIAG3STAT[5]	DIAG3STAT[4]	DIAG3STAT[3]	DIAG3STAT[2]	DIAG3STAT[1]	DIAG3STAT[0]	DIAG2STAT[7]	DIAG2STAT[6]	DIAG2STAT[5]	DIAG2STAT[4]	DIAG2STAT[3]	DIAG2STAT[2]	DIAG2STAT[1]	DIAG2STAT[0]	DIAG1STAT[7]	DIAG1STAT[6]	DIAG1STAT[5]	DIAG1STAT[4]	DIAG1STAT[3]	DIAG1STAT[2]	DIAG1STAT[1]	DIAG1STAT[0]	DIAG0STAT[7]	DIAG0STAT[6]	DIAG0STAT[5]	DIAG0STAT[4]	DIAG0STAT[3]	DIAG0STAT[2]	DIAG0STAT[1]	DIAG0STAT[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-20 – DIAGSTAT

DIAG3STAT[7:0]	Directly connected to *ble_dbg3*[7:0] output. Debug use only.
DIAG2STAT[7:0]	Directly connected to *ble_dbg2*[7:0] output. Debug use only.
DIAG1STAT[7:0]	Directly connected to *ble_dbg1*[7:0] output. Debug use only.
DIAG0STAT[7:0]	Directly connected to *ble_dbg0*[7:0] output. Debug use only.

Address	Access		DEBUGADDMAX																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+58'H	R	R/W	REG_ADDMAX[15]	REG_ADDMAX[14]	REG_ADDMAX[13]	REG_ADDMAX[12]	REG_ADDMAX[11]	REG_ADDMAX[10]	REG_ADDMAX[9]	REG_ADDMAX[8]	REG_ADDMAX[7]	REG_ADDMAX[6]	REG_ADDMAX[5]	REG_ADDMAX[4]	REG_ADDMAX[3]	REG_ADDMAX[2]	REG_ADDMAX[1]	REG_ADDMAX[0]	EM_ADDMAX[15]	EM_ADDMAX[14]	EM_ADDMAX[13]	EM_ADDMAX[12]	EM_ADDMAX[11]	EM_ADDMAX[10]	EM_ADDMAX[9]	EM_ADDMAX[8]	EM_ADDMAX[7]	EM_ADDMAX[6]	EM_ADDMAX[5]	EM_ADDMAX[4]	EM_ADDMAX[3]	EM_ADDMAX[2]	EM_ADDMAX[1]	EM_ADDMAX[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-21 – DEBUGADDMAX

REG_ADDMAX[15:0]	Upper limit for the Register zone indicated by the *reg_inzone* flag (see section 2.16)
EM_ADDMAX[15:0]	Upper limit for the Exchange Memory zone indicated by the *em_inzone* flag (see section 2.16)

Address	Access		DEBUGADDMIN																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+5C'H	R	R/W	REG_ADDMIN[15]	REG_ADDMIN[14]	REG_ADDMIN[13]	REG_ADDMIN[12]	REG_ADDMIN[11]	REG_ADDMIN[10]	REG_ADDMIN[9]	REG_ADDMIN[8]	REG_ADDMIN[7]	REG_ADDMIN[6]	REG_ADDMIN[5]	REG_ADDMIN[4]	REG_ADDMIN[3]	REG_ADDMIN[2]	REG_ADDMIN[1]	REG_ADDMIN[0]	EM_ADDMIN[15]	EM_ADDMIN[14]	EM_ADDMIN[13]	EM_ADDMIN[12]	EM_ADDMIN[11]	EM_ADDMIN[10]	EM_ADDMIN[9]	EM_ADDMIN[8]	EM_ADDMIN[7]	EM_ADDMIN[6]	EM_ADDMIN[5]	EM_ADDMIN[4]	EM_ADDMIN[3]	EM_ADDMIN[2]	EM_ADDMIN[1]	EM_ADDMIN[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-22 – DEBUGADDMIN

REG_ADDMIN[15:0]	Lower limit for the Register zone indicated by the *reg_inzone* flag (see section 2.16)
EM_ADDMIN [15:0]	Lower limit for the Exchange Memory zone indicated by the *em_inzone* flag (see section 2.16)

Address	Access		ERRORTYPESTAT																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+60'H	W	R													RAL_UNDERRUN	RAL_ERROR	CONCEVTIRQ_ERROR	RXDATA_PTR_ERROR	TXDATA_PTR_ERROR	RXDESC_EMPTY_ERROR	TXDESC_EMPTY_ERROR	CSFORMAT_ERROR	LLCHMAP_ERROR	ADV_UNDERRUN	IFS_UNDERRUN	WHITELIST_ERROR	EVT_CNTL_APFM_ERROR	EVT_SCHDL_APFM_ERROR	EVT_SCHDL_ENTRY_ERROR	EVT_SCHDL_EMACC_ERROR	RADIO_EMACC_ERROR	PKTCNTL_EMACC_ERROR	RXCRYPT_ERROR	TXCRYPT_ERROR
Reset value															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type															U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access															W	W	W	W	W	W	W	R	W	W	W	W	W	W	W	W	W	W	W	W
SW Access															R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification															R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-23 – ERRORTYPESTAT

RAL_UNDERRUN	Indicates Resolving Address List engine Under run issue, happens when RAL List parsing not finished on time 0: No error 1: Error occurred
RAL_ERROR	Indicates Resolving Address List engine faced a bad setting (e.g CS-RAL_EN = 1 and null RALPTR, or RALPTR > CS-PEER_RALPTR). 0: No error 1: Error occurred
CONCEVTIRQ_ERROR	Indicates whether two consecutive and concurrent *ble_event_irq* have been generated, and not acknowledged in time by the RW-BLE Software. 0: No error 1: Error occurred
RXDATA_PTR_ERROR	Indicates whether Rx data buffer pointer value programmed is null: this is a major programming failure. 0: No error 1: Error occurred
TXDATA_PTR_ERROR	Indicates whether Tx data buffer pointer value programmed is null during Advertising / Scanning / Initiating events, or during Master / Slave connections with non-null packet length: this is a major programming failure. 0: No error 1: Error occurred
RXDESC_EMPTY_ERROR	Indicates whether Rx Descriptor pointer value programmed in register is null: this is a major programming failure. 0: No error 1: Error occurred
TXDESC_EMPTY_ERROR	Indicates whether Tx Descriptor pointer value programmed in Control Structure is null during Advertising / Scanning / Initiating events: this is a major programming failure. 0: No error 1: Error occurred
CSFORMAT_ERROR	Indicates whether CS-FORMAT has been programmed with an invalid value: this is a major software programming failure. 0: No error 1: Error occurred
LLCHMAP_ERROR	Indicates Link Layer Channel Map error, happens when actual number of CS-LLCHMAP bit set to one is different from CS-NBCHGOOD at the beginning of Frequency Hopping process 0: No error 1: Error occurred
ADV_UNDERRUN	Indicates Advertising Interval Under run, occurs if time between two consecutive Advertising packet (in Advertising mode) is lower than described in Table 3-11.

	0: No error 1: Error occurred
IFS_UNDERRUN	Indicates Inter Frame Space Under run, occurs if IFS time is not enough to update and read Control Structure/Descriptors, and/or White List parsing is not finished and/or Decryption time is too long to be finished on time 0: No error 1: Error occurred
WHITELIST_ERROR	Indicates White List Timeout error, occurs if White List parsing is not finished on time 0: No error 1: Error occurred
EVT_CNTL_APFM_ERROR	Indicates Anticipated Pre-Fetch Mechanism error: happens when 2 consecutive events are programmed, and when the first event is not completely finished while second pre-fetch instant is reached. 0: No error 1: Error occured
EVT_SCHDL_APFM_ERROR	Indicates Anticipated Pre-Fetch Mechanism error: happens when 2 consecutive events are programmed, and when the first event is not completely finished while second pre-fetch instant is reached. 0: No error 1: Error occured
EVT_SCHDL_ENTRY_ERROR	Indicates Event Scheduler faced Invalid timing programing on two consecutive ET entries (e.g first one with 624µs offset and second one with no offset) 0: No error 1: Error occurred
EVT_SCHDL_EMACC_ERROR	Indicates Event Scheduler Exchange Memory access error, happens when Exchange Memory accesses are not served in time, and blocks the Exchange Table entry read 0: No error 1: Error occurred
RADIO_EMACC_ERROR	Indicates Radio Controller Exchange Memory access error, happens when Exchange Memory accesses are not served in time and data are corrupted. 0: No error 1: Error occurred
PKTCNTL_EMACC_ERROR	Indicates Packet Controller Exchange Memory access error, happens when Exchange Memory accesses are not served in time and Tx/Rx data are corrupted 0: No error 1: Error occurred
RXCRYPT_ERROR	Indicates real time decryption error, happens when AES-CCM decryption is too slow compared to Packet Controller requests. A 16-bytes block has to be decrypted prior the next block is received by the Packet Controller 0: No error 1: Error occurred
TXCRYPT_ERROR	Indicates Real Time encryption error, happens when AES-CCM encryption is too slow compared to Packet Controller requests. A 16-bytes block has to be encrypted and prepared on Packet Controller request, and needs to be ready before the Packet Controller has to send ti 0: No error 1: Error occurred

Address	Access		SWPROFILING																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+64'H	R	R/W	SWPROF31	SWPROF30	SWPROF29	SWPROF28	SWPROF27	SWPROF26	SWPROF25	SWPROF24	SWPROF23	SWPROF22	SWPROF21	SWPROF20	SWPROF19	SWPROF18	SWPROF17	SWPROF16	SWPROF15	SWPROF14	SWPROF13	SWPROF12	SWPROF11	SWPROF10	SWPROF9	SWPROF8	SWPROF7	SWPROF6	SWPROF5	SWPROF4	SWPROF3	SWPROF2	SWPROF1	SWPROF0
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-24 – SWPROFILING

SWPROFVAL[31:0]	Software Profiling register: used by RW-BLE Software for profiling purpose: this value is copied on Diagnostic port (Please refer to section 2.16 for details)

3.12.4 Radio registers

In order to configure the RF chip, registers are reserved for control of various functions of the Radio Interface or behavior of the RW-BLE Core radio signals. Some of these registers may not be present depending on RW-BLE Core configuration The exact meaning and content of these registers depends on the actual selected radio, and are detailed in their respective User Manual document..

Address	Access		RADIOCNTL0																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+70'H	R	R/W	RADIO0[31]	RADIO0[30]	RADIO0[29]	RADIO0[28]	RADIO0[27]	RADIO0[26]	RADIO0[25]	RADIO0[24]	RADIO0[23]	RADIO0[22]	RADIO0[21]	RADIO0[20]	RADIO0[19]	RADIO0[18]	RADIO0[17]	RADIO0[16]	RADIO0[15]	RADIO0[14]	RADIO0[13]	RADIO0[12]	RADIO0[11]	RADIO0[10]	RADIO0[9]	RADIO0[8]	RADIO0[7]	RADIO0[6]	RADIO0[5]	RADIO0[4]	RADIO0[3]	RADIO0[2]	RADIO0[1]	RADIO0[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-25 – RADIOCNTL0

RADIO0[31:0]	Principal control register for the radio interface.

Address	Access		RADIOCNTL1																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+74'H	R	R/W	RADIO1[31]	RADIO1[30]	RADIO1[29]	RADIO1[28]	RADIO1[27]	RADIO1[26]	RADIO1[25]	RADIO1[24]	RADIO1[23]	RADIO1[22]	RADIO1[21]	RADIO1[20]	RADIO1[19]	RADIO1[18]	RADIO1[17]	RADIO1[16]	RADIO1[15]	RADIO1[14]	RADIO1[13]	RADIO1[12]	RADIO1[11]	RADIO1[10]	RADIO1[9]	RADIO1[8]	RADIO1[7]	RADIO1[6]	RADIO1[5]	RADIO1[4]	RADIO1[3]	RADIO1[2]	RADIO1[1]	RADIO1[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-26 – RADIOCNTL1

RADIO1[31:0]	Second control register for the radio interface.

Address	Access		RADIOCNTL2																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+78'H	R	R/W	RADIO2[31]	RADIO2[30]	RADIO2[29]	RADIO2[28]	RADIO2[27]	RADIO2[26]	RADIO2[25]	RADIO2[24]	RADIO2[23]	RADIO2[22]	RADIO2[21]	RADIO2[20]	RADIO2[19]	RADIO2[18]	RADIO2[17]	RADIO2[16]	RADIO2[15]	RADIO2[14]	RADIO2[13]	RADIO2[12]	RADIO2[11]	RADIO2[10]	RADIO2[9]	RADIO2[8]	RADIO2[7]	RADIO2[6]	RADIO2[5]	RADIO2[4]	RADIO2[3]	RADIO2[2]	RADIO2[1]	RADIO2[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-27 – RADIOCNTL2

RADIO2[31:0]	Third control register for the radio interface.

Address	Access		RADIOCNTL3																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+7C'H	R	R/W	RADIO3[31]	RADIO3[30]	RADIO3[29]	RADIO3[28]	RADIO3[27]	RADIO3[26]	RADIO3[25]	RADIO3[24]	RADIO3[23]	RADIO3[22]	RADIO3[21]	RADIO3[20]	RADIO3[19]	RADIO3[18]	RADIO3[17]	RADIO3[16]	RADIO3[15]	RADIO3[14]	RADIO3[13]	RADIO3[12]	RADIO3[11]	RADIO3[10]	RADIO3[9]	RADIO3[8]	RADIO3[7]	RADIO3[6]	RADIO3[5]	RADIO3[4]	RADIO3[3]	RADIO3[2]	RADIO3[1]	RADIO3[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-28 – RADIOCNTL3

RADIO3[31:0]	Fourth control register for the radio interface.

Please refer to Table 2-13 and Radio User Manuals for radio memory space definition.

Address	Access		RADIOPWRUPDN																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+80'H	R	R/W		RTRIP_DELAY[6]	RTRIP_DELAY[5]	RTRIP_DELAY[4]	RTRIP_DELAY[3]	RTRIP_DELAY[2]	RTRIP_DELAY[1]	RTRIP_DELAY[0]	RXPWRUP[7]	RXPWRUP[6]	RXPWRUP[5]	RXPWRUP[4]	RXPWRUP[3]	RXPWRUP[2]	RXPWRUP[1]	RXPWRUP[0]				TXPWRDN[4]	TXPWRDN[3]	TXPWRDN[2]	TXPWRDN[1]	TXPWRDN[0]	TXPWRUP[7]	TXPWRUP[6]	TXPWRUP[5]	TXPWRUP[4]	TXPWRUP[3]	TXPWRUP[2]	TXPWRUP[1]	TXPWRUP[0]
Reset value				0	0	0	0	0	0	0	1	1	0	1	0	0	1	0				0	0	0	1	1	1	1	0	1	0	0	1	0
Type				U	U	U	U	U	U	U	U	U	U	U	U	U	U	U				U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access				R	R	R	R	R	R	R	R	R	R	R	R	R	R	R				R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-29 – RADIOPWRUPDN

RTRIP_DELAY[6:0]	Defines round trip delay value. This value correspond to the addition of data latency in Tx and data latency in Rx. Value is in µs
RXPWRUP[7:0]	This register holds the length in µs of the RX power up phase for the current radio device. Default value is 210µs (reset value). Operating range depends on the selected radio.
TXPWRDN[4:0]	This register extends the length in µs of the TX power down phase for the current radio device. Default value is 3 µs (reset value). Operating range depends on the selected radio.
TXPWRUP[7:0]	This register holds the length in µs of the TX power up phase for the current radio device. Default value is 210µs (reset value). Operating range depends on the selected radio.

Note that RXPWRUP delay must take Rx path latency into account in order to ensure correct enable of the RF in receipt, hence a correct synchronization window opening time.

Address	Access		SPIPTRCNTL0																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+84'H	R	R/W	SPIPTR0[31]	SPIPTR0[30]	SPIPTR0[29]	SPIPTR0[28]	SPIPTR0[27]	SPIPTR0[26]	SPIPTR0[25]	SPIPTR0[24]	SPIPTR0[23]	SPIPTR0[22]	SPIPTR0[21]	SPIPTR0[20]	SPIPTR0[19]	SPIPTR0[18]	SPIPTR0[17]	SPIPTR0[16]	SPIPTR0[15]	SPIPTR0[14]	SPIPTR0[13]	SPIPTR0[12]	SPIPTR0[11]	SPIPTR0[10]	SPIPTR0[9]	SPIPTR0[8]	SPIPTR0[7]	SPIPTR0[6]	SPIPTR0[5]	SPIPTR0[4]	SPIPTR0[3]	SPIPTR0[2]	SPIPTR0[1]	SPIPTR0[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-30 – SPIPTRCNTL0

SPIPTR0[31:0]	First control register for the radio interface SPI pointers

Address	Access		SPIPTRCNTL1																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+88'H	R	R/W	SPIPTR1[31]	SPIPTR1[30]	SPIPTR1[29]	SPIPTR1[28]	SPIPTR1[27]	SPIPTR1[26]	SPIPTR1[25]	SPIPTR1[24]	SPIPTR1[23]	SPIPTR1[22]	SPIPTR1[21]	SPIPTR1[20]	SPIPTR1[19]	SPIPTR1[18]	SPIPTR1[17]	SPIPTR1[16]	SPIPTR1[15]	SPIPTR1[14]	SPIPTR1[13]	SPIPTR1[12]	SPIPTR1[11]	SPIPTR1[10]	SPIPTR1[9]	SPIPTR1[8]	SPIPTR1[7]	SPIPTR1[6]	SPIPTR1[5]	SPIPTR1[4]	SPIPTR1[3]	SPIPTR1[2]	SPIPTR1[1]	SPIPTR1[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-31 – SPIPTRCNTL1

SPIPTR1[31:0]	Second control register for the radio interface SPI pointers

Address	Access		SPIPTRCNTL2																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+8C'H	R	R/W	SPIPTR2[31]	SPIPTR2[30]	SPIPTR2[29]	SPIPTR2[28]	SPIPTR2[27]	SPIPTR2[26]	SPIPTR2[25]	SPIPTR2[24]	SPIPTR2[23]	SPIPTR2[22]	SPIPTR2[21]	SPIPTR2[20]	SPIPTR2[19]	SPIPTR2[18]	SPIPTR2[17]	SPIPTR2[16]	SPIPTR2[15]	SPIPTR2[14]	SPIPTR2[13]	SPIPTR2[12]	SPIPTR2[11]	SPIPTR2[10]	SPIPTR2[9]	SPIPTR2[8]	SPIPTR2[7]	SPIPTR2[6]	SPIPTR2[5]	SPIPTR2[4]	SPIPTR2[3]	SPIPTR2[2]	SPIPTR2[1]	SPIPTR2[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-32 – SPIPTRCNTL2

SPIPTR2[31:0]	Third control register for the radio interface SPI pointers

3.12.5 Advertising Channel Map register

Advertising Channel Map defines the channel indexes available for Advertising event.

Address	Access		ADVCHMAP																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+90'H	R	R/W																														ADVCHMAP[2]	ADVCHMAP[1]	ADVCHMAP[0]
Reset value																																1	1	1
Type																																U	U	U
HW Access																																R	R	R
SW Access																																R/W	R/W	R/W
Verification																																R/W	R/W	R/W

Register 3-33 – ADVCHMAP

ADVCHMAP[2:0]	Advertising Channel Map, defined as per the advertising connection settings. Contains advertising channels index 37 to 39. If ADVCHMAP[i] equals: 0: Do not use data channel i+37. 1: Use data channel i+37.

3.12.6 Advertising timer and Scanning timer registers

This section provides necessary delay information that handles advertising event timers.

Address	Access		ADVTIM																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+A0'H	R	R/W																			ADVINT[13]	ADVINT[12]	ADVINT[11]	ADVINT[10]	ADVINT[9]	ADVINT[8]	ADVINT[7]	ADVINT[6]	ADVINT[5]	ADVINT[4]	ADVINT[3]	ADVINT[2]	ADVINT[1]	ADVINT[0]
Reset value																					0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																					U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																					R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																					R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-34 – ADVTIM

ADVINT[13:0]	Advertising Packet Interval defines the time interval in between two ADV_xxx packet sent. Value is in µs. Value to program depends on the used Advertising Packet type and the device filtering policy. Please refer to Table 3-11 for details about ADVINT programming range.

Advertising Packet Type	ADV_INT range			
	Min		Max	
ADV_IND	≥ 1.378ms[(1)]	≥ 1.028ms[(2)]	≤ 10ms	
ADV_DIRECT_IND	≥ 1.028ms[(3)]	≥ 1.028ms[(4)]	≤ 1.25ms[(3)]	≤ 10ms[(4)]
ADV_NONCONN_IND	≥ 0.526ms		≤ 10ms	
ADV_SCAN_IND	≥ 1.378ms		≤ 10ms	

Table 3-11 – Advertising Packet Interval Range

[(1)]: Case of Scanner response. Used if Device Filtering Policy allows response to a scanner or an Initiator

[(2)] Case of Initiator response. Used if Device filtering Policy allows response to Initiator, and scanner are discarded.

[(3)] Case of High Duty cycle Directed Advertising CS-FORMAT

[(4)] Case of Low Duty cycle Directed Advertising CS-FORMAT

Address	Access		ACTSCANSTAT																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+A4'H	W	R								BACKOFF[8]	BACKOFF[7]	BACKOFF[6]	BACKOFF[5]	BACKOFF[4]	BACKOFF[3]	BACKOFF[2]	BACKOFF[1]	BACKOFF[0]								UPPERLIMIT[8]	UPPERLIMIT[7]	UPPERLIMIT[6]	UPPERLIMIT[5]	UPPERLIMIT[4]	UPPERLIMIT[3]	UPPERLIMIT[2]	UPPERLIMIT[1]	UPPERLIMIT[0]
Reset value										0	0	0	0	0	0	0	0	1								0	0	0	0	0	0	0	0	1
Type										U	U	U	U	U	U	U	U	U								U	U	U	U	U	U	U	U	U
HW Access										W	W	W	W	W	W	W	W	W								W	W	W	W	W	W	W	W	W
SW Access										R	R	R	R	R	R	R	R	R								R	R	R	R	R	R	R	R	R
Verification										R	R	R	R	R	R	R	R	R								R	R	R	R	R	R	R	R	R

Register 3-35 – ACTSCANSTAT

BACKOFF[8:0]	Active scan mode back-off counter initialization value.
UPPERLIMIT[8:0]	Active scan mode upper limit counter value.

3.12.7 Device filtering registers

This section provides necessary information to the Event Controller for device filtering.

Address	Access		WLPUBADDPTR																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+B0'H	R	R/W																	WLPUBADDPTR[15]	WLPUBADDPTR[14]	WLPUBADDPTR[13]	WLPUBADDPTR[12]	WLPUBADDPTR[11]	WLPUBADDPTR[10]	WLPUBADDPTR[9]	WLPUBADDPTR[8]	WLPUBADDPTR[7]	WLPUBADDPTR[6]	WLPUBADDPTR[5]	WLPUBADDPTR[4]	WLPUBADDPTR[3]	WLPUBADDPTR[2]	WLPUBADDPTR[1]	WLPUBADDPTR[0]
Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-36 – WLPUBADDPTR

WLPUBADDPTR[15:0]	Start address pointer of the public devices white list.

Address	Access		WLPRIVADDPTR																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+B4'H	R	R/W																	WLPRIVADDPTR[15]	WLPRIVADDPTR[14]	WLPRIVADDPTR[13]	WLPRIVADDPTR[12]	WLPRIVADDPTR[11]	WLPRIVADDPTR[10]	WLPRIVADDPTR[9]	WLPRIVADDPTR[8]	WLPRIVADDPTR[7]	WLPRIVADDPTR[6]	WLPRIVADDPTR[5]	WLPRIVADDPTR[4]	WLPRIVADDPTR[3]	WLPRIVADDPTR[2]	WLPRIVADDPTR[1]	WLPRIVADDPTR[0]
Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-37 – WLPRIVADDPTR

WLPRIVADDPTR[15:0]	Start address pointer of the private devices white list.

Address	Access		WLNBDEV																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+B8'H	R	R/W																	NBPRIVDEV[7]	NBPRIVDEV[6]	NBPRIVDEV[5]	NBPRIVDEV[4]	NBPRIVDEV[3]	NBPRIVDEV[2]	NBPRIVDEV[1]	NBPRIVDEV[0]	NBPUBDEV[7]	NBPUBDEV[6]	NBPUBDEV[5]	NBPUBDEV[4]	NBPUBDEV[3]	NBPUBDEV[2]	NBPUBDEV[1]	NBPUBDEV[0]
Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-38 – WLNBDEV

NBPRIVDEV[7:0]	Number of private devices in the white list.
NBPUBDEV[7:0]	Number of public devices in the white list.

3.12.8 Encryption registers

This section provides necessary information regarding AES-128 ciphering process in case of RW-BLE Software request. It also details AES-CCM status registers (plain MIC value in Tx and Rx). These registers are present only if *RW_BLE_CRYPT_INST* is set

Address	Access		AESCNTL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+C0'H	R/W	R/W																															AES_MODE	AES_START
Reset value																																	0	0
Type																																	U	U
HW Access																																	R	R/W
SW Access																																	R/W	S
Verification																																	R/W	R/W

Register 3-39 – AESCNTL

AES_MODE	0: Cipher mode 1: Decipher mode
AES_START	Writing a 1 starts AES-128 ciphering/deciphering process. This bit is reset once the process is finished (i.e. *ble_crypt_irq* interrupt occurs, even masked)

Address	Access		AESKEY31_0																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+C4'H	R	R/W	AESKEY31_0[31]	AESKEY31_0[30]	AESKEY31_0[29]	AESKEY31_0[28]	AESKEY31_0[27]	AESKEY31_0[26]	AESKEY31_0[25]	AESKEY31_0[24]	AESKEY31_0[23]	AESKEY31_0[22]	AESKEY31_0[21]	AESKEY31_0[20]	AESKEY31_0[19]	AESKEY31_0[18]	AESKEY31_0[17]	AESKEY31_0[16]	AESKEY31_0[15]	AESKEY31_0[14]	AESKEY31_0[13]	AESKEY31_0[12]	AESKEY31_0[11]	AESKEY31_0[10]	AESKEY31_0[9]	AESKEY31_0[8]	AESKEY31_0[7]	AESKEY31_0[6]	AESKEY31_0[5]	AESKEY31_0[4]	AESKEY31_0[3]	AESKEY31_0[2]	AESKEY31_0[1]	AESKEY31_0[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-40 – AESKEY31_0

AESKEY31_0[31:0]	AES encryption 128-bit key. Bit 31 down to 0

Address	Access		AESKEY63_32																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+C8'H	R	R/W	AESKEY63_32[31]	AESKEY63_32[30]	AESKEY63_32[29]	AESKEY63_32[28]	AESKEY63_32[27]	AESKEY63_32[26]	AESKEY63_32[25]	AESKEY63_32[24]	AESKEY63_32[23]	AESKEY63_32[22]	AESKEY63_32[21]	AESKEY63_32[20]	AESKEY63_32[19]	AESKEY63_32[18]	AESKEY63_32[17]	AESKEY63_32[16]	AESKEY63_32[15]	AESKEY63_32[14]	AESKEY63_32[13]	AESKEY63_32[12]	AESKEY63_32[11]	AESKEY63_32[10]	AESKEY63_32[9]	AESKEY63_32[8]	AESKEY63_32[7]	AESKEY63_32[6]	AESKEY63_32[5]	AESKEY63_32[4]	AESKEY63_32[3]	AESKEY63_32[2]	AESKEY63_32[1]	AESKEY63_32[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-41 – AESKEY63_32

AESKEY63_32[31:0]	AES encryption 128-bit key. Bit 63 down to 32

Address	Access		AESKEY95_64																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+CC'H	R	R/W	AESKEY95_64[31]	AESKEY95_64[30]	AESKEY95_64[29]	AESKEY95_64[28]	AESKEY95_64[27]	AESKEY95_64[26]	AESKEY95_64[25]	AESKEY95_64[24]	AESKEY95_64[23]	AESKEY95_64[22]	AESKEY95_64[21]	AESKEY95_64[20]	AESKEY95_64[19]	AESKEY95_64[18]	AESKEY95_64[17]	AESKEY95_64[16]	AESKEY95_64[15]	AESKEY95_64[14]	AESKEY95_64[13]	AESKEY95_64[12]	AESKEY95_64[11]	AESKEY95_64[10]	AESKEY95_64[9]	AESKEY95_64[8]	AESKEY95_64[7]	AESKEY95_64[6]	AESKEY95_64[5]	AESKEY95_64[4]	AESKEY95_64[3]	AESKEY95_64[2]	AESKEY95_64[1]	AESKEY95_64[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-42 – AESKEY95_64

AESKEY95_64[31:0]	AES encryption 128-bit key. Bit 95 down to 64

Address	Access		AESKEY127_96																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+D0'H	R	R/W	AESKEY127_96[31]	AESKEY127_96[30]	AESKEY127_96[29]	AESKEY127_96[28]	AESKEY127_96[27]	AESKEY127_96[26]	AESKEY127_96[25]	AESKEY127_96[24]	AESKEY127_96[23]	AESKEY127_96[22]	AESKEY127_96[21]	AESKEY127_96[20]	AESKEY127_96[19]	AESKEY127_96[18]	AESKEY127_96[17]	AESKEY127_96[16]	AESKEY127_96[15]	AESKEY127_96[14]	AESKEY127_96[13]	AESKEY127_96[12]	AESKEY127_96[11]	AESKEY127_96[10]	AESKEY127_96[9]	AESKEY127_96[8]	AESKEY127_96[7]	AESKEY127_96[6]	AESKEY127_96[5]	AESKEY127_96[4]	AESKEY127_96[3]	AESKEY127_96[2]	AESKEY127_96[1]	AESKEY127_96[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-43 – AESKEY127_96

AESKEY128_96[31:0]	AES encryption 128-bit key. Bit 127 down to 96

Address	Access		AESPTR																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+D4'H	R	R/W																	AESPTR[15]	AESPTR[14]	AESPTR[13]	AESPTR[12]	AESPTR[11]	AESPTR[10]	AESPTR[9]	AESPTR[8]	AESPTR[7]	AESPTR[6]	AESPTR[5]	AESPTR[4]	AESPTR[3]	AESPTR[2]	AESPTR[1]	AESPTR[0]
Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-44 – AESPTR

AESPTR[15:0]	Pointer to the memory zone where the block to cipher/decipher using AES-128 is stored.

Address	Access		TXMICVAL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+D8'H	W	R	TXMICVAL[31]	TXMICVAL[30]	TXMICVAL[29]	TXMICVAL[28]	TXMICVAL[27]	TXMICVAL[26]	TXMICVAL[25]	TXMICVAL[24]	TXMICVAL[23]	TXMICVAL[22]	TXMICVAL[21]	TXMICVAL[20]	TXMICVAL[19]	TXMICVAL[18]	TXMICVAL[17]	TXMICVAL[16]	TXMICVAL[15]	TXMICVAL[14]	TXMICVAL[13]	TXMICVAL[12]	TXMICVAL[11]	TXMICVAL[10]	TXMICVAL[9]	TXMICVAL[8]	TXMICVAL[7]	TXMICVAL[6]	TXMICVAL[5]	TXMICVAL[4]	TXMICVAL[3]	TXMICVAL[2]	TXMICVAL[1]	TXMICVAL[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-45 – TXMICVAL

TXMICVAL[31:0]	AES-CCM plain MIC value. Valid on when MIC has been calculated (in Tx)

Address	Access		RXMICVAL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+DC'H	W	R	RXMICVAL[31]	RXMICVAL[30]	RXMICVAL[29]	RXMICVAL[28]	RXMICVAL[27]	RXMICVAL[26]	RXMICVAL[25]	RXMICVAL[24]	RXMICVAL[23]	RXMICVAL[22]	RXMICVAL[21]	RXMICVAL[20]	RXMICVAL[19]	RXMICVAL[18]	RXMICVAL[17]	RXMICVAL[16]	RXMICVAL[15]	RXMICVAL[14]	RXMICVAL[13]	RXMICVAL[12]	RXMICVAL[11]	RXMICVAL[10]	RXMICVAL[9]	RXMICVAL[8]	RXMICVAL[7]	RXMICVAL[6]	RXMICVAL[5]	RXMICVAL[4]	RXMICVAL[3]	RXMICVAL[2]	RXMICVAL[1]	RXMICVAL[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Register 3-46 – RXMICVAL

RXMICVAL[31:0]	AES-CCM plain MIC value. Valid on once MIC has been extracted from Rx packet.

3.12.9 Regulatory Body and RF Testing Register

This section provides the different settings related to RF testing and regulatory body support.

Address	Access		RFTESTCNTL																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+E0'H	R	R/W	INFINITERX				RXPKTCNTEN												INFINITETX	TXLENGTHSRC	PRBSTYPE	TXPLDSRC	TXPKTCNTEN			TXLENGTH[8]	TXLENGTH[7]	TXLENGTH[6]	TXLENGTH[5]	TXLENGTH[4]	TXLENGTH[3]	TXLENGTH[2]	TXLENGTH[1]	TXLENGTH[0]
Reset value			0				0												0	0	0	0	0			0	0	0	0	0	0	0	0	0
Type			U				U												U	U	U	U	U			U	U	U	U	U	U	U	U	U
HW Access			R				R												R	R	R	R	R			R	R	R	R	R	R	R	R	R
SW Access			R/W				R/W												R/W	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W				R/W												R/W	R/W	R/W	R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-47 – RFTESTCNTL

INFINITERX	Applicable to all event type 0: Normal mode of operation 1: Infinite Rx window
RXPKTCNTEN	Applicable in RF Direct Rx Test mode only 0: Rx packet count disabled 1: Rx packet count enabled, and reported in CS-RXCCMPKTCNT and RFTESTRXSTAT-RXPKTCNT on RF abort command
INFINITETX	Applicable to all event type 0: Normal mode of operation. 1: Infinite Tx packet / Normal start of a packet but endless payload
TXLENGTHSRC	Applicable to all event type 0: Normal mode of operation: TxDESC-<TXADVLEN/TXLEN> controls the Tx packet payload size 1: Uses RFTESTCNTL-TXLENGTH packet length (can support up to 512 bytes transmit)
PRBSTYPE	Defines the PRBS in use 0: Tx Packet Payload are PRBS9 type 1: Tx Packet Payload are PRBS15 type
TXPLDSRC	Applicable to all event type 0: Tx Packet Payload source is the Control Structure 1: Tx Packet Payload are PRBS generator
TXPKTCNTEN	Applicable in RF Direct Tx Test mode only 0: Tx packet count disabled 1: Tx packet count enabled, and reported in CS-TXCCMPKTCNT and RFTESTTXSTAT-TXPKTCNT on RF abort command
TXLENGTH[8:0]	Applicable to all event type, valid when RFTESTCNTL-TXLENGTHSRC = 1 Tx packet length in number of byte

Note this register is useful only for RF Testing purpose.

In case of infinite Tx mode, all packet types can be supported (depending on Event type). Transmitted data can come either from PRBS generators, or Tx Data Buffers. When Tx Data Buffers are selected, then the length defined in the Control Structure is used to determine the pattern length to loop on. RW-BLE Core continues to send the selected data till RFTESTCNTL-RFTEST_ABORT is set.

PRBS9 is defined as $p(x)=1+x^5+x^9$. The LFSR used for the PRBS9 generator is initialized with 0x1FF value.

PRBS15 is defined as $p(x)=1+x^{14}+x^{15}$. The LFSR used for the PRBS15 generator is initialized with 0x7FFF value.

In case of infinite Tx payload, and when PRBS source is not selected, then RFTESTCNTL-TXLENGTH field provides the length of the pattern to repeat in the payload.

Address	Access		RFTESTTXSTAT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+E4'H	W	R	TXPKTCNT[31]	TXPKTCNT[30]	TXPKTCNT[29]	TXPKTCNT[28]	TXPKTCNT[27]	TXPKTCNT[26]	TXPKTCNT[25]	TXPKTCNT[24]	TXPKTCNT[23]	TXPKTCNT[22]	TXPKTCNT[21]	TXPKTCNT[20]	TXPKTCNT[19]	TXPKTCNT[18]	TXPKTCNT[17]	TXPKTCNT[16]	TXPKTCNT[15]	TXPKTCNT[14]	TXPKTCNT[13]	TXPKTCNT[12]	TXPKTCNT[11]	TXPKTCNT[10]	TXPKTCNT[9]	TXPKTCNT[8]	TXPKTCNT[7]	TXPKTCNT[6]	TXPKTCNT[5]	TXPKTCNT[4]	TXPKTCNT[3]	TXPKTCNT[2]	TXPKTCNT[1]	TXPKTCNT[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-48 – RFTESTTXSTAT

TXPKTCNT[31:0]	Reports number of transmitted packet during Test Modes. Value is valid if RFTESTCNTL-TXPKTCNTEN is set

Address	Access		RFTESTRXSTAT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+E8'H	W	R	RXPKTCNT[31]	RXPKTCNT[30]	RXPKTCNT[29]	RXPKTCNT[28]	RXPKTCNT[27]	RXPKTCNT[26]	RXPKTCNT[25]	RXPKTCNT[24]	RXPKTCNT[23]	RXPKTCNT[22]	RXPKTCNT[21]	RXPKTCNT[20]	RXPKTCNT[19]	RXPKTCNT[18]	RXPKTCNT[17]	RXPKTCNT[16]	RXPKTCNT[15]	RXPKTCNT[14]	RXPKTCNT[13]	RXPKTCNT[12]	RXPKTCNT[11]	RXPKTCNT[10]	RXPKTCNT[9]	RXPKTCNT[8]	RXPKTCNT[7]	RXPKTCNT[6]	RXPKTCNT[5]	RXPKTCNT[4]	RXPKTCNT[3]	RXPKTCNT[2]	RXPKTCNT[1]	RXPKTCNT[0]
Reset value			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
SW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-49 – RFTESTRXSTAT

RXPKTCNT[31:0]	Reports number of correctly received packet during Test Modes (no sync error, no CRC error). Value is valid if RFTESTCNTL-RXPKTCNTEN is set

3.12.10 Timing Generator Register

This section provides Timing Generator settings.

Address	Access		TIMGENCNTL																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+F0'H	R	R/W	APFM_EN						PREFETCHABORT_TIME[9]	PREFETCHABORT_TIME[8]	PREFETCHABORT_TIME[7]	PREFETCHABORT_TIME[6]	PREFETCHABORT_TIME[5]	PREFETCHABORT_TIME[4]	PREFETCHABORT_TIME[3]	PREFETCHABORT_TIME[2]	PREFETCHABORT_TIME[1]	PREFETCHABORT_TIME[0]								PREFETCH_TIME[8]	PREFETCH_TIME[7]	PREFETCH_TIME[6]	PREFETCH_TIME[5]	PREFETCH_TIME[4]	PREFETCH_TIME[3]	PREFETCH_TIME[2]	PREFETCH_TIME[1]	PREFETCH_TIME[0]
Reset value			1						0	1	1	1	1	1	1	1	1	0								0	1	0	0	1	0	1	1	0
Type			U						U	U	U	U	U	U	U	U	U	U								U	U	U	U	U	U	U	U	U
HW Access			R						R	R	R	R	R	R	R	R	R	R								R	R	R	R	R	R	R	R	R
SW Access			R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W						R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-50 – TIMGENCNTL

APFM_EN	Controls the Anticipated pre-Fetch Abort mechanism 0: Disabled 1: Enabled
PREFETCHABORT_TIME[9:0]	Defines the instant in µs at which immediate abort is required after anticipated pre-fetch abort

PREFETCH_TIME[8:0]	Defines Exchange Table pre-fetch instant in µs

Address	Access		GROSSTIMTGT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+F4'H	R	R/W										GROSSTARGET[22]	GROSSTARGET[21]	GROSSTARGET[20]	GROSSTARGET[19]	GROSSTARGET[18]	GROSSTARGET[17]	GROSSTARGET[16]	GROSSTARGET[15]	GROSSTARGET[14]	GROSSTARGET[13]	GROSSTARGET[12]	GROSSTARGET[11]	GROSSTARGET[10]	GROSSTARGET[9]	GROSSTARGET[8]	GROSSTARGET[7]	GROSSTARGET[6]	GROSSTARGET[5]	GROSSTARGET[4]	GROSSTARGET[3]	GROSSTARGET[2]	GROSSTARGET[1]	GROSSTARGET[0]
Reset value												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type												U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access												R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access												R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification												R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-51 – GROSSTIMTGT

GROSSTARGET[22:0]	Gross Timer Target value on which a *ble_grosstgtim_irq* must be generated. This timer has a precision of 10ms: interrupt is generated only when GROSSTARGET[22:0] = BASETIMECNT[26:4] and BASETIMECNT[3:0] = 0.

Address	Access		FINETIMTGT																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+F8'H	R	R/W						FINETARGET[26]	FINETARGET[25]	FINETARGET[24]	FINETARGET[23]	FINETARGET[22]	FINETARGET[21]	FINETARGET[20]	FINETARGET[19]	FINETARGET[18]	FINETARGET[17]	FINETARGET[16]	FINETARGET[15]	FINETARGET[14]	FINETARGET[13]	FINETARGET[12]	FINETARGET[11]	FINETARGET[10]	FINETARGET[9]	FINETARGET[8]	FINETARGET[7]	FINETARGET[6]	FINETARGET[5]	FINETARGET[4]	FINETARGET[3]	FINETARGET[2]	FINETARGET[1]	FINETARGET[0]
Reset value								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type								U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access								R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-52 – FINETIMTGT

FINETARGET[26:0]	Fine Timer Target value on which a *ble_finetgtim_irq* must be generated. This timer has a precision of 625µs: interrupt is generated only when FINETARGET = BASETIMECNT

3.12.11 WLAN Coexistence Register

This section provides WLAN Coexistence interface behavior programming details. These registers are present when *RW_BLE_WLAN_COEX_INST* is set

Address	Access		COEXIFCNTL0																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+100'H	R	R/W							MWSSCANFREQMSK[1]	MWSSCANFREQMSK[0]			WLCRXPRIOMODE[1]	WLCRXPRIOMODE[0]			WLCTXPRIOMODE[1]	WLCTXPRIOMODE[0]	MWSTXFRQMSK[1]	MWSTXFRQMSK[0]	MWSRXFRQMSK[1]	MWSRXFRQMSK[0]	MWSTXMSK[1]	MWSTXMSK[0]	MWSRXMSK[1]	MWSRXMSK[0]	WLANTXMSK[1]	WLANTXMSK[0]	WLANRXMSK[1]	WLANRXMSK[0]	MWSWCI_EN	MWSCOEX_EN	SYNCGEN_EN	WLANCOEX_EN
Reset value									0	0			0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Type									U	U			U	U			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access									R	R			R	R			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access									R/W	R/W			R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification									R/W	R/W			R/W	R/W			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-53 – COEXIFCNTL0

MWSSCANFREQMSK[1:0]	Determines how mws_scan_frequency impacts BLE Tx and Rx / Valid in Dual Mode only 00: mws_scan_frequency has no impact ***(default mode)*** 01: mws_scan_frequency can stop BLE Tx, no impact on BLE Rx 10: mws_scan_frequency can stop BLE Rx, no impact on BLE Tx 11: mws_scan_frequency can stop both BLE Tx and BLE Rx
WLCRXPRIOMODE[1:0]	Defines Bluetooth Low Energy packet *ble_rx* mode behavior. 00: Rx indication excluding Rx Power up delay (starts when correlator is enabled) 01: Rx indication including Rx Power up delay 10: Rx High priority indicator 11: n/a
WLCTXPRIOMODE[1:0]	Defines Bluetooth Low Energy packet *ble_tx* mode behavior 00: Tx indication excluding Tx Power up delay 01: Tx indication including Tx Power up delay 10: Tx High priority indicator 11: n/a
MWSTXFREQMSK[1:0]	Determines how MWS Tx Frequency impacts BLE Tx and Rx / Valid in Dual Mode only 00: MWS Tx Frequency has no impact ***(default mode)*** 01: MWS Tx Frequency can stop BLE Tx, no impact on BLE Rx 10: MWS Tx Frequency can stop BLE Rx, no impact on BLE Tx 11: MWS Tx Frequency can stop both BLE Tx and BLE Rx
MWSRXFREQMSK[1:0]	Determines how MWS Rx Frequency impacts BLE Tx and Rx / Valid in Dual Mode only 00: MWS Tx Frequency has no impact ***(default mode)*** 01: MWS Tx Frequency can stop BLE Tx, no impact on BLE Rx 10: MWS Tx Frequency can stop BLE Rx, no impact on BLE Tx 11: MWS Tx Frequency can stop both BLE Tx and BLE Rx
MWSTXMSK[1:0]	Determines how *mws_tx* impacts BLE Tx and Rx / Valid in Dual Mode only 00: mws_tx has no impact ***(default mode)*** 01: mws_tx can stop BLE Tx, no impact on BLE Rx 10: mws_tx can stop BLE Rx, no impact on BLE Tx 11: mws_tx can stop both BLE Tx and BLE Rx
MWSRXMSK[1:0]	Determines how *mws_rx* impacts BLE Tx and Rx / Valid in Dual Mode only 00: mws_tx has no impact ***(default mode)*** 01: mws_tx can stop BLE Tx, no impact on BLE Rx 10: mws_tx can stop BLE Rx, no impact on BLE Tx 11: mws_tx can stop both BLE Tx and BLE Rx
WLANTXMSK[1:0]	Determines how *wlan_tx* impact BLE Tx and Rx 00: *wlan_tx* has no impact ***(default mode)*** 01: *wlan_tx* can stop BLE Tx, no impact on BLE Rx 10: *wlan_tx* can stop BLE Rx, no impact on BLE Tx 11: *wlan_tx* can stop both BLE Tx and BLE Rx
WLANRXMSK[1:0]	Determines how *wlan_rx* impact BLE Tx and Rx 00: *wlan_rx* has no impact 01: *wlan_rx* can stop BLE Tx, no impact on BLE Rx ***(default mode)*** 10: *wlan_rx* can stop BLE Rx, no impact on BLE Tx

	11: *wlan_rx* can stop both BLE Tx and BLE Rx
MWSWCI_EN	Enable / Disable control of the WCI MWS Coexistence interface / Valid in Dual Mode only 0: MWS WCI Interface disabled 1: MWS WCI Interface enabled
MWSCOEX_EN	Enable / Disable control of the MWS Coexistence control / Valid in Dual Mode only 0: MWS Coexistence interface disabled 1: MWS Coexistence interface enabled
SYNCGEN_EN	Determines whether *ble_sync* is generated or not. 0: *ble_sync* pulse not generated 1: *ble_sync* pulse generated
WLANCOEX_EN	Enable / Disable control of the WLAN Coexistence control 0: WLAN Coexistence interface disabled 1: WLAN Coexistence interface enabled

Address	Access		COEXIFCNTL1																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+104'H	R	R/W				WLCPRXTHR[4]	WLCPRXTHR[3]	WLCPRXTHR[2]	WLCPRXTHR[1]	WLCPRXTHR[0]				WLCPTXTHR[4]	WLCPTXTHR[3]	WLCPTXTHR[2]	WLCPTXTHR[1]	WLCPTXTHR[0]		WLCPDURATION[6]	WLCPDURATION[5]	WLCPDURATION[4]	WLCPDURATION[3]	WLCPDURATION[2]	WLCPDURATION[1]	WLCPDURATION[0]		WLCPDELAY[6]	WLCPDELAY[5]	WLCPDELAY[4]	WLCPDELAY[3]	WLCPDELAY[2]	WLCPDELAY[1]	WLCPDELAY[0]
Reset value						0	0	0	0	0				0	0	0	0	0		0	0	0	0	0	0	0		0	0	0	0	0	0	0
Type						U	U	U	U	U				U	U	U	U	U		U	U	U	U	U	U	U		U	U	U	U	U	U	U
HW Access						R	R	R	R	R				R	R	R	R	R		R	R	R	R	R	R	R		R	R	R	R	R	R	R
SW Access						R/W	R/W	R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification						R/W	R/W	R/W	R/W	R/W				R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-54 – COEXIFCNTL1

WLCPRXTHR[4:0]	Applies on *ble_rx* if WLCRXPRIOMODE equals “10” Determines the threshold for Rx priority setting. If *ble_pti[3:0]* output value is greater than WLCPRXTHR, then Rx Bluetooth Low Energy priority is considered as high, and must be provided to the WLAN coexistence interface
WLCPTXTHR[4:0]	Applies on *ble_tx* if WLCTXPRIOMODE equals “10” Determines the threshold for priority setting. If *ble_pti[3:0]* output value is greater than WLCPTXTHR, then Tx Bluetooth Low Energy priority is considered as high, and must be provided to the WLAN coexistence interface
WLCPDURATION[6:0]	Applies on *ble_tx* if WLCTXPRIOMODE equals “10” Applies on *ble_rx* if WLCRXPRIOMODE equals “10” Determines how many µs the priority information must be maintained Note that if WLCPDURATION = 0x00, then Tx/Rx priority levels are maintained till Tx/Rx EN are de-asserted.
WLCPDELAY[6:0]	Applies on *ble_tx* if WLCTXPRIOMODE equals “10” Applies on *ble_rx* if WLCRXPRIOMODE equals “10” Determines the delay (in µs) in Tx/Rx enables rises the time Bluetooth Low energy Tx/Rx priority has to be provided .

Address	Access		COEXIFCNTL2																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+108'H	R	R/W																					RX_ANT_DELAY[3]	RX_ANT_DELAY[2]	RX_ANT_DELAY[1]	RX_ANT_DELAY[0]					TX_ANT_DELAY[3]	TX_ANT_DELAY[2]	TX_ANT_DELAY[1]	TX_ANT_DELAY[0]
Reset value																							0	0	0	0					0	0	0	0
Type																							U	U	U	U					U	U	U	U
HW Access																							R	R	R	R					R	R	R	R
SW Access																							R/W	R/W	R/W	R/W					R/W	R/W	R/W	R/W
Verification																							R/W	R/W	R/W	R/W					R/W	R/W	R/W	R/W

Register 3-55 – COEXIFCNTL2

RX_ANT_DELAY[3:0]	Applies if WLCRXPRIOMODE equals to "01" Time (in µs) by which is anticipated bt_rx to be provided before effective Radio receipt operation
TX_ANT_DELAY[3:0]	Applies if WLCTXPRIOMODE equals to "01" Time (in µs) by which is anticipated bt_tx to be provided before effective Radio transmit operation

Address	Access		BLEMPRIO0																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+10C'H	R	R/W	BLEM7[3]	BLEM7[2]	BLEM7[1]	BLEM7[0]	BLEM6[3]	BLEM6[2]	BLEM6[1]	BLEM6[0]	BLEM5[3]	BLEM5[2]	BLEM5[1]	BLEM5[0]	BLEM4[3]	BLEM4[2]	BLEM4[1]	BLEM4[0]	BLEM3[3]	BLEM3[2]	BLEM3[1]	BLEM3[0]	BLEM2[3]	BLEM2[2]	BLEM2[1]	BLEM2[0]	BLEM1[3]	BLEM1[2]	BLEM1[1]	BLEM1[0]	BLEM0[3]	BLEM0[2]	BLEM0[1]	BLEM0[0]
Reset value			0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	0	1	1	0	1	1	1	1	0	1	1	1	1
Type			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-56 – BLEMPRIO0

BLEM0[3:0]	Set Priority value for Initiating (Connection Request Response) BLE message
BLEM1[3:0]	Set Priority value for LLCP BLE message
BLEM2[3:0]	Set Priority value for Data Channel transmission BLE message
BLEM3[3:0]	Set Priority value for Initiating (Scanning) BLE message
BLEM4[3:0]	Set Priority value for Active Scanning BLE message
BLEM5[3:0]	Set Priority value for Connectable Advertising BLE message
BLEM6[3:0]	Set Priority value for Non-Connectable Advertising
BLEM7[3:0]	Set Priority value for Passive Scanning

Address	Access		BLEMPRIO1																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+110'H	R	R/W	BLEMDEFAULT[3]	BLEMDEFAULT[2]	BLEMDEFAULT[1]	BLEMDEFAULT[0]																												
Reset value			0	0	1	1																												
Type			U	U	U	U																												
HW Access			R	R	R	R																												
SW Access			R/W	R/W	R/W	R/W																												
Verification			R/W	R/W	R/W	R/W																												

Register 3-57 – BLEMPRIO1

BLEMDEFAULT[3:0]	Set default priority value for other BLE message than those defined above

3.12.12 Resolving Address List Registers

This registers are used by the Resolving Address List engine to locate the RAL Structure start address pointer, determine the number of devices stored in the RAL structure, and the RW-BLE Core device random number to be used for Local RPA generation.

Address	Access		RALPTR																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+120'H	R	R/W																	RALPTR[15]	RALPTR[14]	RALPTR[13]	RALPTR[12]	RALPTR[11]	RALPTR[10]	RALPTR[9]	RALPTR[8]	RALPTR[7]	RALPTR[6]	RALPTR[5]	RALPTR[4]	RALPTR[3]	RALPTR[2]	RALPTR[1]	RALPTR[0]
Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Type																			U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access																			R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
SW Access																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-58 – RALPTR

RALPTR[15:0]	Start address pointer of the RAL structure

Address	Access		RALNBDEV																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+124'H	R	R/W																									NBRALDEV[7]	NBRALDEV[6]	NBRALDEV[5]	NBRALDEV[4]	NBRALDEV[3]	NBRALDEV[2]	NBRALDEV[1]	NBRALDEV[0]
Reset value																											0	0	0	0	0	0	0	0
Type																											U	U	U	U	U	U	U	U
HW Access																											R	R	R	R	R	R	R	R
SW Access																											R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																											R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-59 – RALNBDEV

RALNBDEV[7:0]	Number of devices in RAL Structure

Address	Access		RAL_LOCAL_RND																											
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+128'H	R/W	R/W	LRND_INIT										LRND_VAL[21]	LRND_VAL[20]	LRND_VAL[19]	LRND_VAL[18]	LRND_VAL[17]	LRND_VAL[16]	LRND_VAL[15]	LRND_VAL[14]	LRND_VAL[13]	LRND_VAL[12]	LRND_VAL[11]	LRND_VAL[10]	LRND_VAL[9]	LRND_VAL[8]	LRND_VAL[7]	LRND_VAL[6]	LRND_VAL[5]	LRND_VAL[4]	LRND_VAL[3]	LRND_VAL[2]	LRND_VAL[1]	LRND_VAL[0]
Reset value			0										1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Type			U										U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R/W										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SW Access			S										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			DT										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-60 – RAL_LOCAL_RND

LRND_INIT	Writing a 1 initializes of Local RPA random number generation LFSR

	This bit is reset once the LFSR is loaded
LRND_VAL[21:0]	Initialization value for Local RPA random generation when LRDN_INIT is set to 1, else reports the current Local RPA random number LFSR value

Address	Access		RAL_PEER_RND																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+12C'H	R/W	R/W	PRND_INIT										PRND_VAL[21]	PRND_VAL[20]	PRND_VAL[19]	PRND_VAL[18]	PRND_VAL[17]	PRND_VAL[16]	PRND_VAL[15]	PRND_VAL[14]	PRND_VAL[13]	PRND_VAL[12]	PRND_VAL[11]	PRND_VAL[10]	PRND_VAL[9]	PRND_VAL[8]	PRND_VAL[7]	PRND_VAL[6]	PRND_VAL[5]	PRND_VAL[4]	PRND_VAL[3]	PRND_VAL[2]	PRND_VAL[1]	PRND_VAL[0]
Reset value			0										1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
Type			U										U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
HW Access			R/W										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SW Access			S										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification			DT										R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-61 – RAL_PEER_RND

PRND_INIT	Writing a 1 initializes of Peer RPA random number generation LFSR This bit is reset once the LFSR is loaded
PRND_VAL[21:0]	Initialization value for Peer RPA random generation when PRDN_INIT is set to 1, else reports the current Peer RPA random number LFSR value

3.12.13 Priority Scheduling Arbiter Control Register

This register controls the decision instant for Priority Scheduling Arbitration.

This registers is present when *RW_DM_SUPPORT* is set, and are dedicated to Bluetooth Dual Mode device implementations.

Address	Access		BLEPRIOSCHARB																													
	HW	SW	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+130'H	R	R/W																	BLEPRIOMODE								BLEMARGIN[7]	BLEMARGIN[6]	BLEMARGIN[5]	BLEMARGIN[4]	BLEMARGIN[3]	BLEMARGIN[2]	BLEMARGIN[1]	BLEMARGIN[0]
Reset value																			0								0	0	0	0	0	0	0	0
Type																			U								U	U	U	U	U	U	U	U
HW Access																			R								R	R	R	R	R	R	R	R
SW Access																			R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Verification																			R/W								R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Register 3-62 – BLEPRIOSCHARB

BLEPRIOMODE	Determine BLE Priority Scheduling Arbitration Mode 0: BLE Decision instant not used 1: BLE Decision instant used
BLEMARGIN[7:0]	Determine the decision instant margin for Priority Scheduling Arbitration. Decision instant is defined as per formula of section 3.6

4 Bluetooth Low Energy Packets

This section presents a summary of Bluetooth Low Energy packets and their characteristics.

Packet	Preamble	Access Code	Payload		CRC	Total bits
			header	Data		
ADV_IND	8	32	16	296	24	376
ADV_DIRECT_IND	8	32	16	96	24	176
ADV_NONCONN_IND	8	32	16	296	24	376
ADV_SCAN_IND	8	32	16	296	24	376
SCAN_REQ	8	32	16	96	24	176
SCAN_RESP	8	32	16	296	24	376
CONNECT_REQ	8	32	16	272	24	352
LL DATA (not encrypted)	8	32	16	2008	24	2088
LL CONTROL (not encrypted)	8	32	16	2008	24	2088
LL DATA (encrypted)	8	32	16	2040	24	2120
LL CONTROL (encrypted)	8	32	16	2040	24	2120

Table 4-1 – Bluetooth Low Energy packets

References

[1]	**Title**	Specification of the Bluetooth System		
	Reference			
	Version	v4.2	**Date**	December 3, 2013
	Source	Bluetooth SIG		

[2]	**Title**	AMBA Specifications		
	Reference	IHI 0011A		
	Version	2.0	**Date**	May 13, 1999
	Source	ARM Limited		

[3]	**Title**	Cortus APS3 CPU Core Integration Manual		
	Reference	1007011553		
	Version	3	**Date**	July 7, 2010
	Source	Cortus		

[4]	**Title**	RW-BLE Core Integration Guide		
	Reference	RW-BLE–CORE–IG		
	Version		**Date**	
	Source	RivieraWaves		

[5]	**Title**	Recommendation for Block Cipher Modes of operation		
	Reference	NIST Special Publication 800-38C		
	Version		**Date**	
	Source	National Institute of Standards and Technology		

[6]	**Title**	Advanced Encryption Standard (AES)		
	Reference	FIPS-197		
	Version		**Date**	
	Source	Federal Information / Processing Standards Publication		

[7]	**Title**	Bluetooth Low Energy PHY Test Specification		
	Reference	RF-PHY.TS		
	Version	4.1.0	**Date**	December 3, 2013
	Source	Bluetooth SIG		