RW-BLE Link Layer Software

Functional Specification

RW-BLE-LL-SW-FS

Version 8.0.0

2015-02-19

RivieraWaves
http://www.rivierawaves.com/

Revision History

Version	**Date**	**Revision Description**	**Author**
8.00	2014-02-16	Initial version	SB
8.00	2014-03-09	Add Enhanced Privacy Feature	FBE

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Acronym or abbreviation	Writing out in full / Meaning
BD_ADDR	Bluetooth Device Address
BLE	Bluetooth Low energy
BR/EDR	Basic Rate / Enhanced Data Rate
Core	Base Band
CPU	Core Processing Unit
CS	Control Structure
DUT	Device Under Test
EA	Event Arbiter
EM	Exchange Memory
ET	Exchange Table
HW	Hardware/BLE Core
IP	Intellectual Property
IRK	Identity Resolving Key
LLD	Link Layer Driver (RW naming)
LLM	Link Layer Manager (RW naming)
LLC	Link Layer Controller (RW naming)
LLCP	Link Layer Control Protocol
LLID	Logical Link IDentifier
LSTO	Link Supervision Time Out
LT_ADDR	Logical Transport Address
LTK	Long Term Key
IFS	Inter Frame Space
MSC	Message Sequence Chart
MTU	Maximum Transport Unit (= Max Size – L2CAP header length – Mic length)
OS	Operating System
PDU	Packet Data Unit
PRBS	Pseudo Random Binary Sequence
RAL	Resolving Address List
RF	Radio Frequency
RFU	Reserved for Future Use
RPA	Resolving Private Address
RSSI	Received Signal Strength Indication
RSVD	Reserved bandwidth at LL level
RT	Real Time
RTD	Reference Test Device
Rx	Receive
SCA	Sleep Clock Accuracy
SPI	Serial Port Interface
SW	Software/Firmware
TS	Time Stamp
Tx	Transmit

Table 1-1 – Acronyms and Abbreviations table

1 Overview

1.1 Document Overview

This document describes the functionalities of the different blocks present in the RW-BLE software lower stack.

1.2 Product Overview

The RW-BLE-SW stack can have three different implementation models:

- ✓ The split: where the lower stack runs on one CPU and the upper stack runs on another CPU.
- ✓ The full-embedded: where the lower, upper, profile(s)(optional) and protocol stacks run on the same CPU.
- ✓ The full-hosted: where everything run on the same CPU.

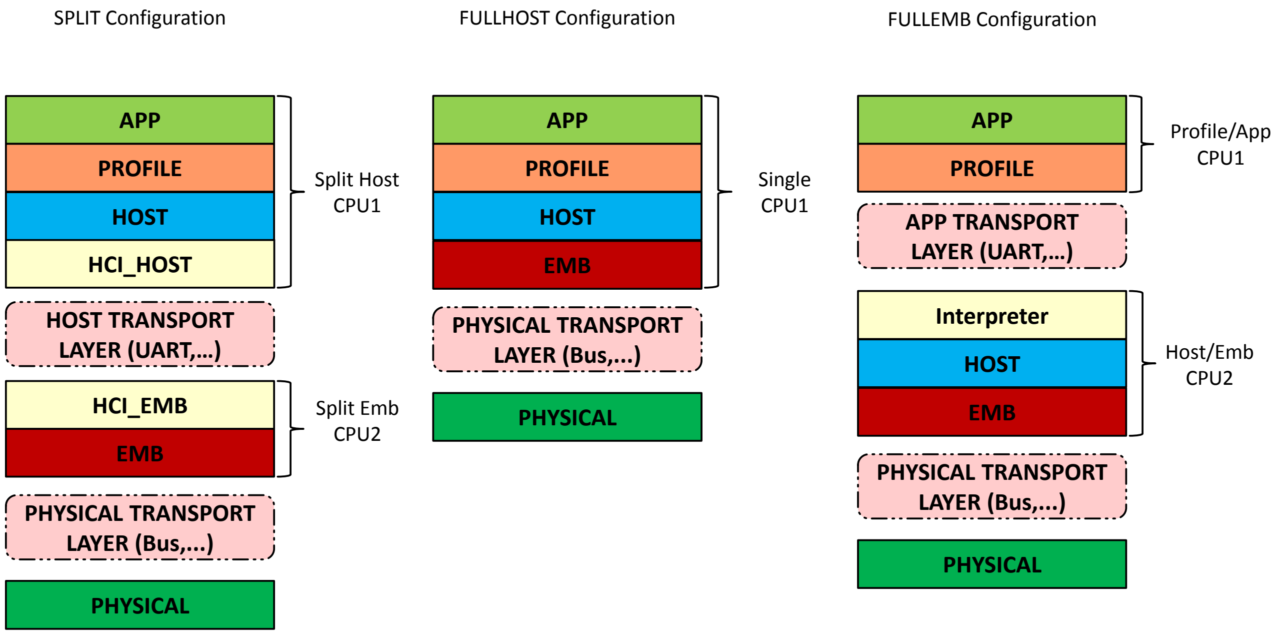


Figure 1-1- BLE stack models overview

Only the BLE lower stack, link layer are explained in this document. The BR/EDR is not covered.

2 Topology

The BLE system is based on a piconet topology. A device can be slave in one piconet and master in another one.

2.1 General Topology

- The RW device is the master of the piconet, linked to multiple slave devices. There are several topologies supported in the RW BLE system. The maximum number of possible links (N) is dictated by a configuration value set at build time in the FW.

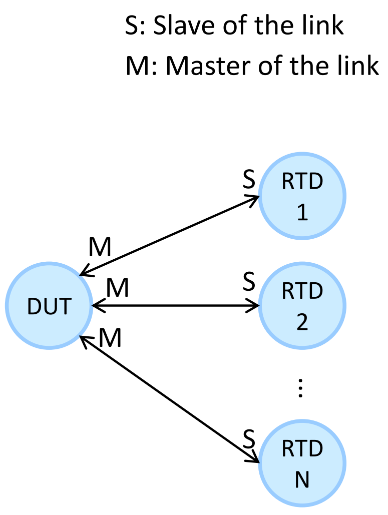


Figure 2-1 – Star Topology

- The RW device is at least one time the master of a piconet and at the same time the slave of another piconet. Several links can be created, bounded by the constraints on the number of possible links set in the FW.

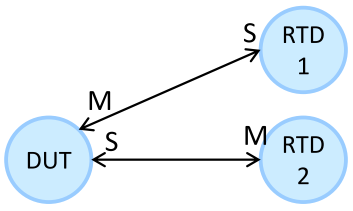


Figure 2-2 – Scatternet Topology

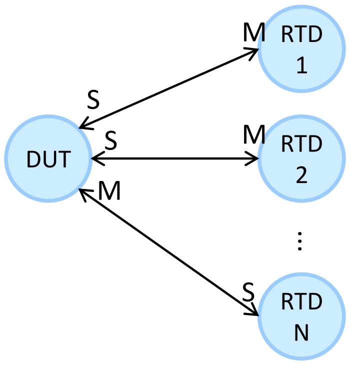


Figure 2-3 – Example: 2 times slave and 1 time master

2.2 Design Limitation

With the current implementation the stack supports only two slave links, adding more slave links is possible but impacts the memory foot print and the bandwidth.

- A Device in piconet or scatternet topology can advertise.

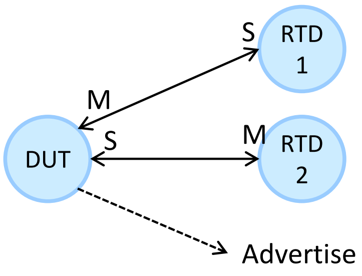


Figure 2-4 – Scatternet topology with advertising

- A Device in piconet or scatternet topology can scan or initiate a connection.

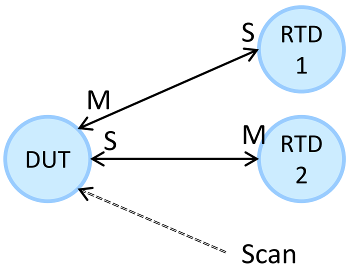


Figure 2-5 – Scatternet topology with scanning.

See Role Combination and states chapter to have more details.

3 Software / Hardware Interfaces

The link layer software needs to have an interface to communicate with the hardware interfaces (Base Band or Host) (see [2]).

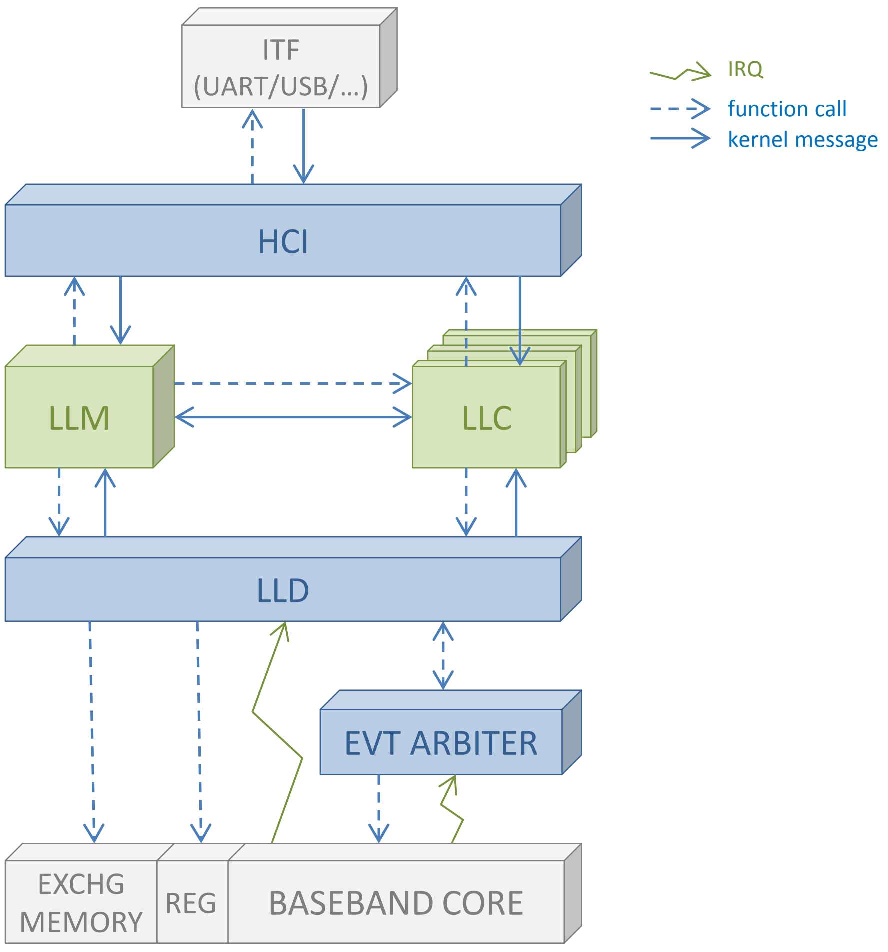


Figure 3-1 – HW/SW interfaces overview

3.1 Slot definition

The system is synchronized as per a base time unit called a "slot" tht is equal to 625 us.

An HW counter named "base time counter" manages the system slot counting.

An HW counter named "fine time counter" performs a 1us precision counting down within a slot. Details are given in [2]

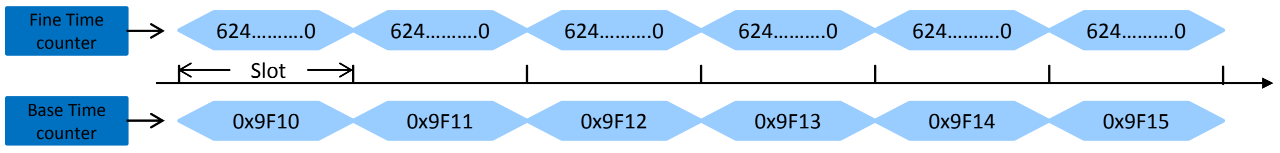


Figure 3-2 – Slot definition

The base time counter start at 0 on power up. It is 27 bits wide and it is increased every 625µs.

3.2 Interrupt definition

The real time scheduling and the system wake up are synchronized over several interruptions.

Those interruptions are generated by the HW (BaseBand / Core).

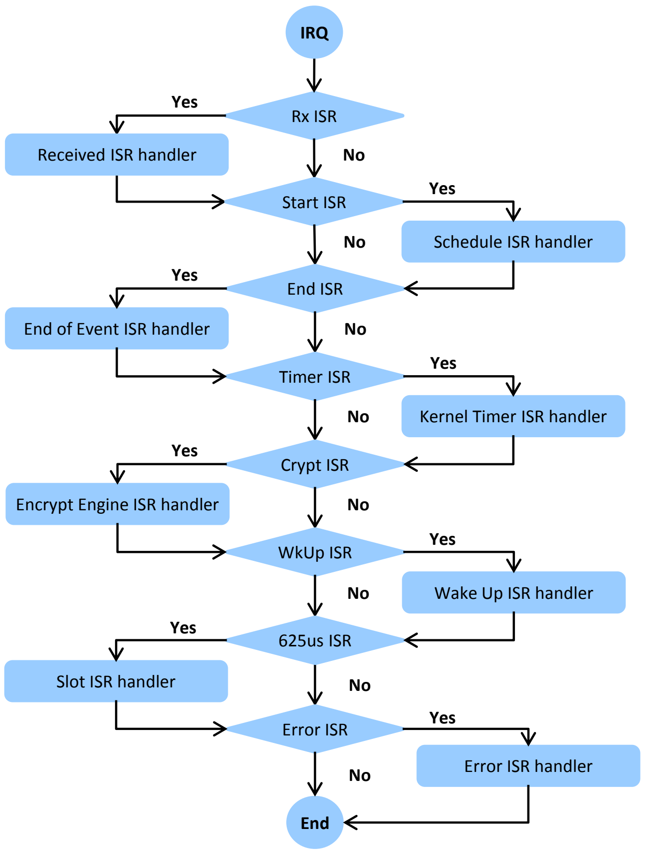


Figure 3-3 – Interrupts overview

3.2.1 Schedule interrupt

The schedule interruption allows the EA to program the ET. This interruption happens in advance compare to the execution time. This delay is called PROG LATENCY, is counted in slot number and is set to 2 by define in the FW (i.e. 2*625us = 1.25 ms).

This latency allows the FW to safely program the CS fields and TX buffers, before the HW fetching.

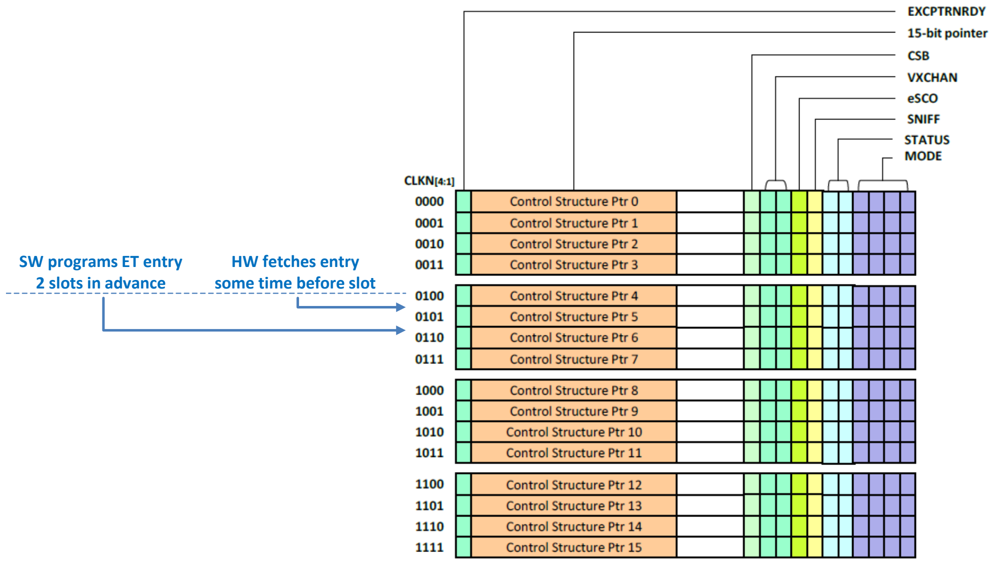


Figure 3-4 – Programming process overview

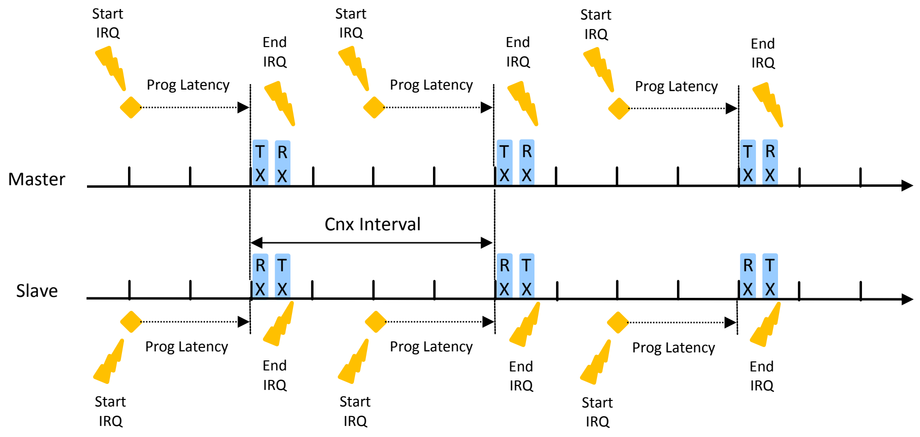


Figure 3-5 – Schedule/End of Event interrupts overview

3.2.2 End of event interrupt

The End of event interruption allows the FW to check what has been received and sent. This interruption happens at the end of the event, even if nothing has been received. See Figure 3-5.

The FW checks the CS-CONFLICT bit to increase the priority in case where the event has been aborted by the HW.

3.2.3 Reception interrupt

The reception interruption allows the FW to check what has been received and sent. This interrupt occurs when the threshold set in the CS (CS-RXTHR see [2]) has been reached or when something is received during scanning activity.

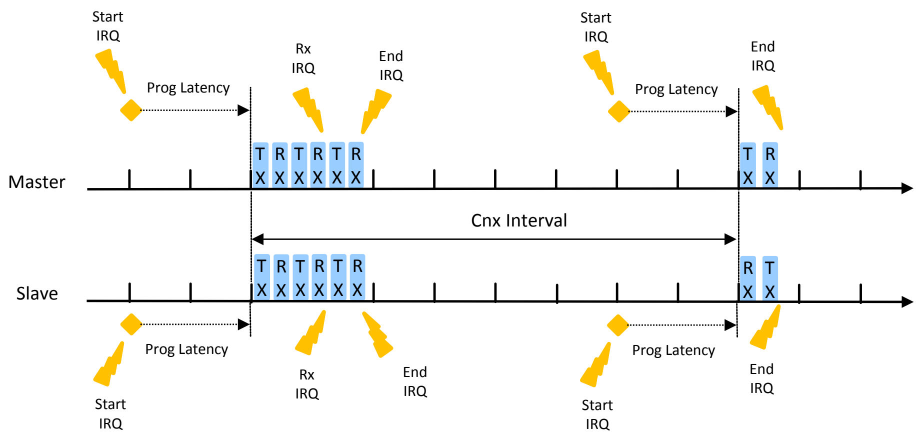


Figure 3-6 – RX interrupts, Rx threshold set to 2

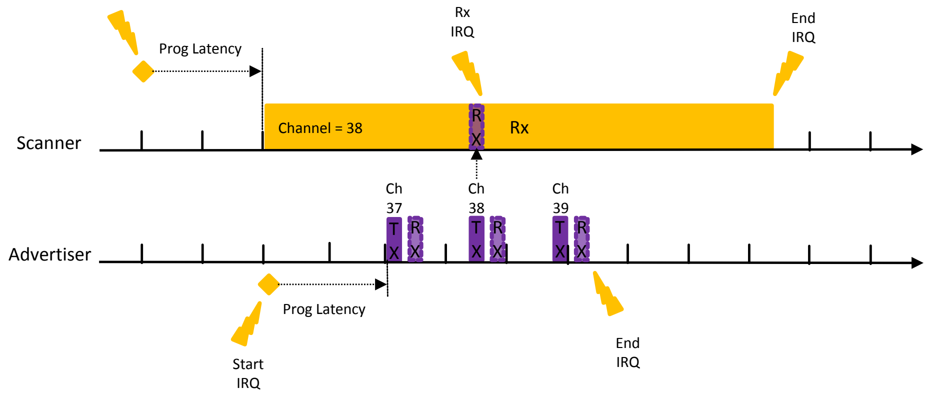


Figure 3-7 – RX interrupts, Advertising packet received

3.2.4 Wake up interrupt

The wake up interruption allows the FW to turn on the system and start a BLE activity. The wake up interrupt can occur asynchronously with a slot boundary. The Wake up has 2 steps:

- ✓ Wake up : start to compensate the low power clock drift **(wake up interrupt)**.
  Slot synchronization: system is ready **(slot interrupt)** the activity can start .

All those phases are shown in the figure below:

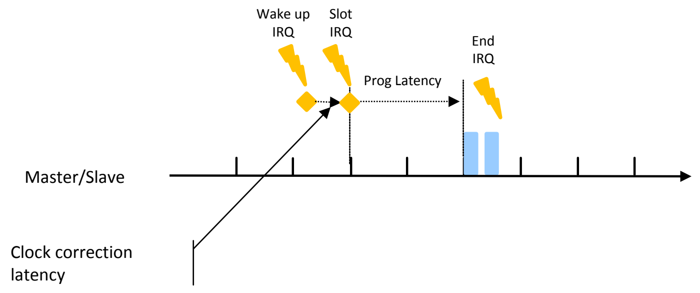


Figure 3-8 – Wake up interrupts

Upon wake up, HW base time counter has to be updated as per the passed sleep duration. For that, FW computes the correction values to apply on a base and fine counters by setting appropriate registers. The core applies the correction for the next slot interrupt. The maximum time to correct the clock value is 1 slot (default value) and is known in the FW as "clock correction latency".

The EA is in charge to check if the event should be programmed or if it is in the past and should be pushed in the cancelled queue.

3.2.5 Encryption interrupt

This interrupt occurs when the encryption engine performs cyphering process running AES-128 toolbox, and when the procedure ends.

The cyphering is started by a command, defined by an API (See [1] part E), with two parameters:

- **Plain data**: data to be encrypted
- **Key** : 128 bit key for the encryption.

When the interruption is generated, a kernel message is sent to the requester with the cyphered data.

3.2.6 Software interrupt

This interrupt indicates an EA cancelation, the FW increments the priority and the programming time before trying to reinsert the event in the EA wait queue.

3.2.7 Error interrupt

This interrupt indicates an HW error has been detected. The error type can be recovered by reading the ERRORTYPESTAT register (see [2]).

3.3 Interface with the Baseband

The LL SW interfaces with the Baseband by writing/reading its registers and the exchange memory. Depending on the physical interface between CPU and BB (e.g. AHB, SPI, etc.), the way for accessing the BB has to be adapted. An abstraction layer is therefore implemented so that LL SW is, as much as possible, independent from the physical interface. Only the LL-BB abstraction layer is modified depending on the physical interface.

The abstraction layer contains a few primitives allowing atomic read/writes, and copies of data from system RAM to exchange memory.

3.3.1 Baseband Abstraction Layer Primitives

The following functions are used by the LL SW to access the BB registers, exchange memory and RF registers:

- *REG_BLE_RD(addr)*
  - This macro reads a 32-bit from a BLE core register
- *REG_BLE_WR(addr, value)*
  - This functions writes a 32-bit value to a BLE core register
- *EM_BLE_RD(addr)*
  - This function reads a 16-bit value from the BLE Exchange Memory
- *EM_BLE_WR(addr, value)*
  - This functions writes a 16-bit value to the Exchange Memory
- *RF_BLE_RD(addr)*
  - This macro reads a value from an RF register
- *RF_BLE_WR(addr, value)*
  - This functions writes a value to an RF register
- *void em_ble_burst_rd(void *sys_addr, uint16_t em_addr, uint16_t len)*
  - This function reads *len* bytes from address *em_addr* in the Exchange Memory and writes them at address *sys_addr* in system memory
- *void em_ble_burst_wr(void const*sys_addr, uint16_t em_addr, uint16_t len)*
  - This function reads *len* bytes from address *sys_addr* in the system memory and writes them at address *em_addr* in the Exchange Memory

The macros and functions above assume that the addresses have the correct alignment. No alignment checking is performed. They have to be declared in a header file named *reg_access.h*, that depends on the physical interface between the LL SW and BB. This file is therefore not part of the LL SW, but it is part of the SW associated to the platform in which the CPU and BB are instantiated.

3.3.2 Registers

Registers are used for static settings and test mode purposes.

In the FW, these registers are accessed via macros. These macros which are python-generated are included automatically as register headers.

An excel sheet contains the registers describes in [2].

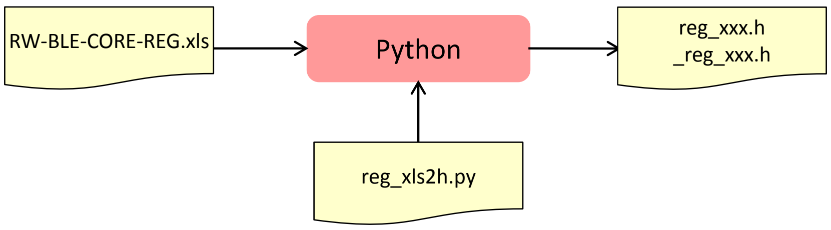


Figure 3-9 – Registers macros generation

<u>Macro register example</u>

reg_blecore.h : get register ble control.

```
__INLINE uint32_t ble_rwblecntl_get(void)
{
   return REG_BLE_RD(BLE_RWBLECNTL_ADDR);
}
```

3.3.3 Exchange Memory

The exchange memory is used for the dynamic settings: its mapping is defined in the file em_map.h.

For the EM, the accesses are also managed over macros. Those macros are generated as the registers are.

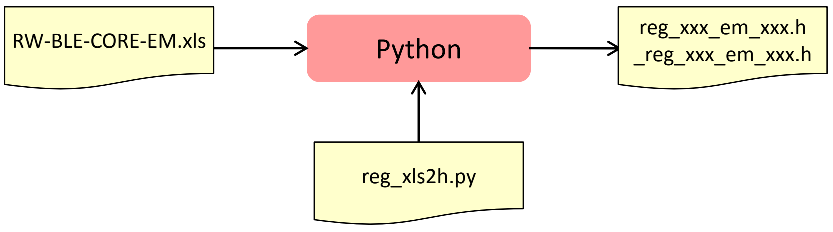


Figure 3-10 – Exchange memory macros generation

All the fields contain in the EM are described in [2].

<u>Macro EM example</u>

reg_ble_em_et.h: write the exchange table entry.

```
__INLINE void ble_extab_set(int elt_idx, uint16_t value)
{
   EM_BLE_WR(BLE_EXTAB_ADDR + elt_idx * REG_BLE_EM_ET_SIZE, value);
}
```

3.3.4 Control Structures

The CS contains all relevant information that gives instructions to the HW in order to perform a specific BLE action (CONNECTION, ADVERTISING, SCANNING, etc...). The CS is described in [2].

The CS is located in the exchange memory, and is allocated statically at build time. The number of allocated CS depends on the maximum number of links set in the FW.

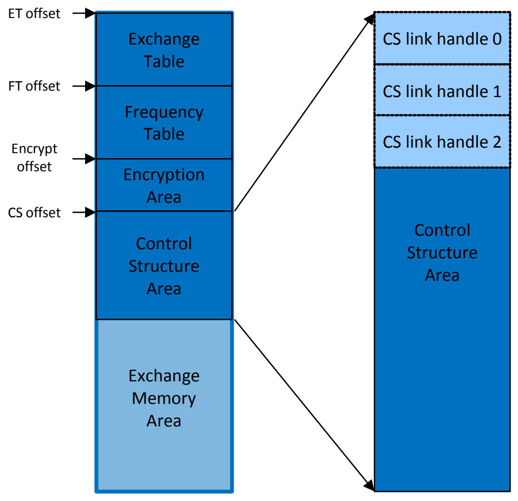


Figure 3-11 – Control structure allocation example

Example:

If the number of link is set to 3, then 4 CS are allocated 3 for the actives links and 1 for the non-connected activity.

To access to the CS the macros define in chapter 3.3.3 are used.

3.4 Host Controller Interface

In split model, in order to control the RW-BLE stack below the HCI, a hardware interface is required. It is not used in Full modes.

HCI is described in [3].

4 List Management

This section describes the list management model that is used for several purposes within the BLE FW, such as:

- BLE event scheduling.
- Kernel message queuing.
- Etc…

The list APIs are defined in the file ***co_list.h***.

A list is defined by this structure:

struct co_list	
*first	pointer to first element of the list
*last	pointer to the last element of the list

Table 4-1 – list definition

4.1 Initialization

At start up the entire lists are initialized by the API **co_list_init.** This API needs the pointer on the list to be reset.

The fields of the list are initialized.

list->first = NULL;

list->last = NULL;

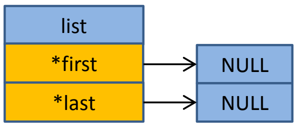


Figure 4-1 – List initialization

4.2 Push an element

Two API allow inserting an element in the list on the front **(co_list_push_front)** or on the back **(co_list_push_back).**

Those API need as input:

- List : List where the element are added.
- Element : The Element to be added.

The elements are casted by the following structure in order to create the chained list:

struct co_list_hdr	
*next	pointer to the next element of the list

Table 4-2 – list definition

It means that if it is required to create a list of elements, those are inserted using this structure on the top of the element.

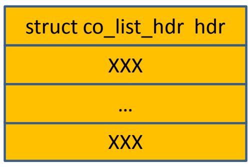


Figure 4-2 – Chain list header

Find an example where an element is chained in an empty list:

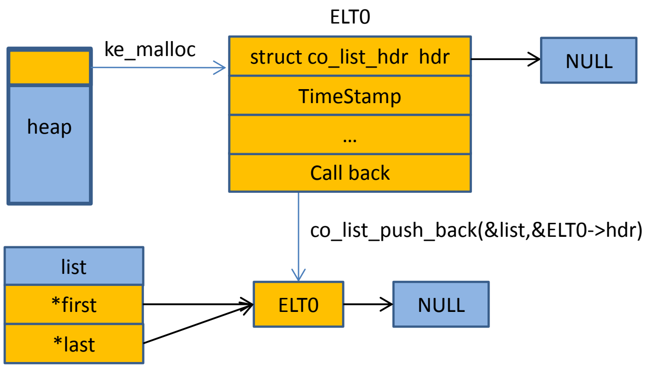


Figure 4-3 – Push back an element in a list

Find an example where an element is chained to the front of a non-empty list:

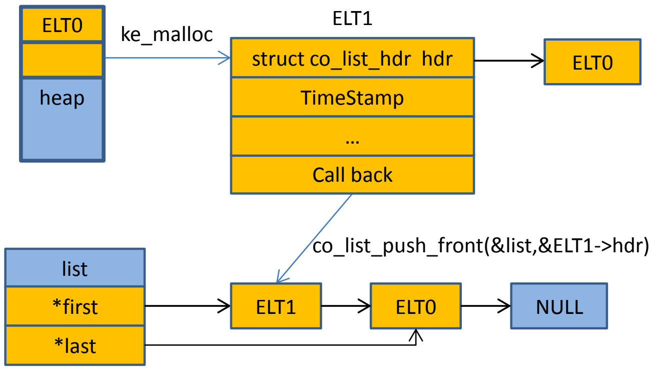


Figure 4-4 – Push front an element in a list

4.3 Pop an element

An API allows get an element in the list from the front **(co_list_pop_front).** This API need as input:

- List : List where the element are obtained.

This API returns a list pointer on the element, a cast is required before accessing to the element structure fields.

After a **co_list_pop_front** the element is removed from the list. In case of element checking, (which does not need its removal from the list), the API **co_list_pick** is used.

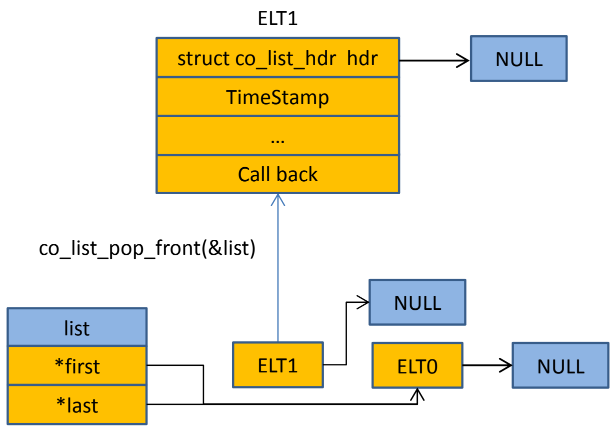


Figure 4-5 – Pop front an element in a list

5 Memory Management

The memory management for the element and environment will use the kernel memory APIs ***(ke_mem.h)***. The kernel can manage several heaps. More details in [4].

The RW BLE system is capable of switching off some parts of memory to save power. This is called deep sleep mode.

Pay attention to free an allocated memory to avoid memory leak.

5.1 Allocation

To allocate a memory area; the API **ke_malloc** needs two parameters:

- Size : in byte.
- Heap type : in which heap are allocated.

A void pointer is returned, it is mandatory to cast the API to avoid compilation issue.

The example below shows an allocation:

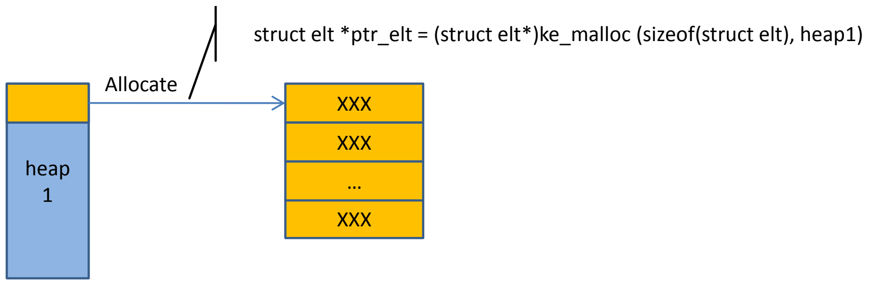

struct elt *ptr_elt = (struct elt*)ke_malloc (sizeof(struct elt), heap1)
Allocate
heap 1
XXX
XXX
…
XXX

Figure 5-1 – Kernel memory allocation

After the allocation, if a non-NULL pointer is returned, the structure fields can be filled.

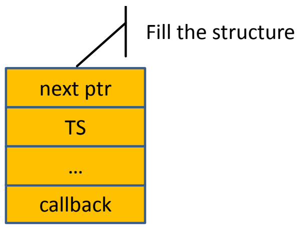


Figure 5-2 – Example: new element filled

5.2 Free

To free an allocated area; the API **ke_free** needs one parameter:

- Pointer : Pointer on the area to be freed.

The API retrieves the freed heap area.

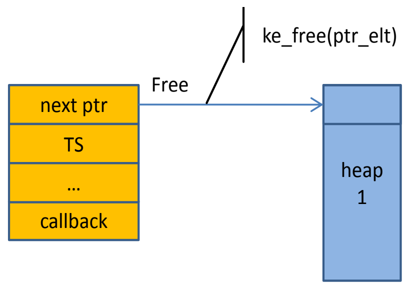


Figure 5-3 – Kernel memory free

6 Buffer Management

For the Tx/Rx data and control buffer management a chained list mechanism is used.

The Rx chained list is implemented as a ring buffer.

A pool of buffers and threshold values for interrupt are used and defined to prevent overflow. The values depend on the number of links, rate of events and the probability of packet retransmission in the link.

Assumptions:

- All the processes in the SW stack use the same mechanism to allocate and free the descriptor.
- SW stack shares same buffer area.
- Allocation is done statically at build time.

On the initialization phase the software:

- Create the ring chain list for the Rx descriptors.
- Initialize the pool of Tx descriptors.
- Initialize the pool of Rx and Tx data buffers.

6.1 Receive part

This part describes how the BLE software stack manages the buffer mechanism to receive a packet.

6.1.1 Rx descriptors

The Rx descriptor is defined in [2].

6.1.2 Rx ring chain list structure

The length of the pool is defined at compilation level; during the initialization the queue is created.

In parallel buffer allocation is done in Rx, you will notice in Figure 6-1 that the data area can be mapped anywhere.

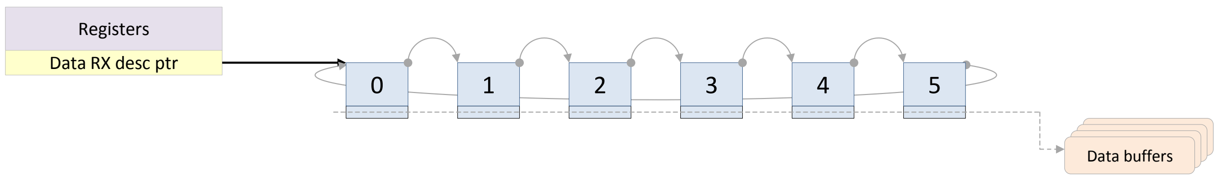


Figure 6-1 - Rx ring chain list

6.1.3 Rx procedure

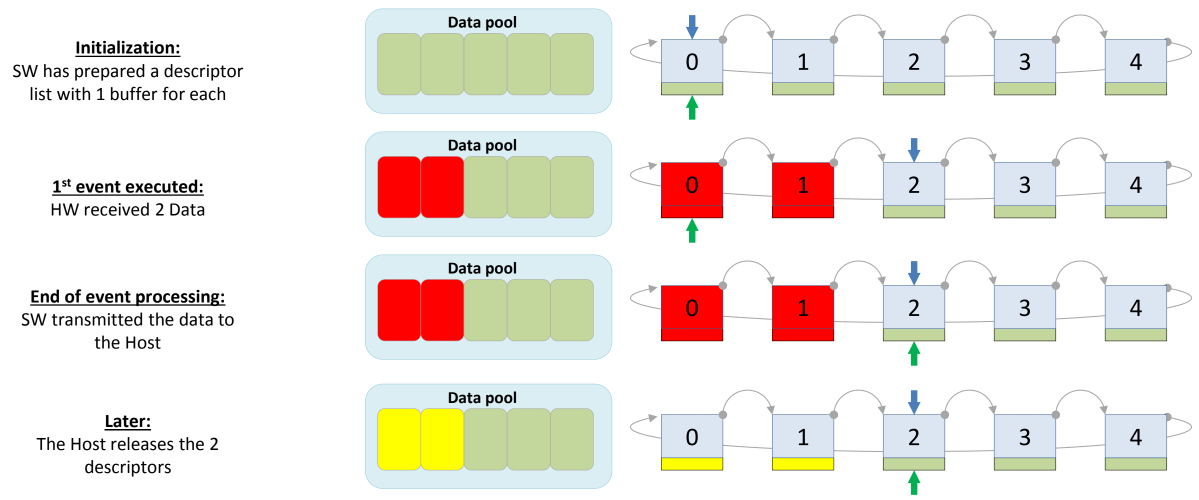
When the BLE stack runs, non-connected or connected modes, the SW action is scheduled on the IRQs: End of Event, Threshold or Receive.

The SW is in charge to initialize the pointer on the Rx ring list used by setting the ET_CURRENTRXDESCPTR-CURRENTRXDESCPTR and also to set:

- CS-RXTHR: Control structure value of the threshold for the RX interrupt.
- CS-RXMAXBUF: Defines the maximum size (in bytes) of the allocated Rx Buffers for a connection.
- CS-RXMAXTIME: Defines the reception maximum time (in µs) for a connection.

When either the threshold, or the receive or the end of event interruption rises, the link layer driver access to the header and status fields of the first descriptor and check if the data needs to be processed or not (error case).

The link layer driver is in charge to maintain a pointer on the next descriptor to be processed and to free the processed descriptor by resetting the status bit RxDESC-RXDONE in the descriptor if not data is received, else the destination of the data is in charge to free the descriptor.


Initialization:
SW has prepared a descriptor list with 1 buffer for each
1st event executed:
HW received 2 Data
End of event processing:
SW transmitted the data to the Host
Later:
The Host releases the 2 descriptors
Data pool
0
1
2
3
4

🟨	free buffer	⬜	free descriptor
🟩	allocated buffer	🟥	used descriptor
🟥	filled buffer	▬	point to filled buffer
⬇	HW write pointer	▭	point to empty buffer
⬆	SW read pointer	▭	point to NULL

Figure 6-2 – Reception example

6.2 Transmit part

This part describes how the BLE software stack manages the buffer mechanism to send packets.

6.2.1 Tx descriptors

The Tx descriptor is defined in [2].

6.2.2 Tx chain list structure

The length of the pool is defined at compilation level. The queue is created during the initialization.

Two queues are initialized:

- Descriptors queue

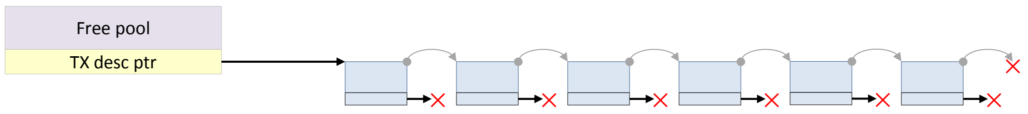


Figure 6-3 – Free descriptor chain list

This queue is used to allocate a new descriptor when a data should be transmitted.

- Index queue

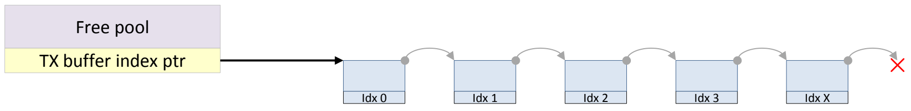


Figure 6-4 - Free buffer index chain list

This queue is used to check the available data buffer, each index represent a free buffer area.

The buffer area is represented by a base address pointer and a size offset, to look over the index queue and the buffer size offset are used.

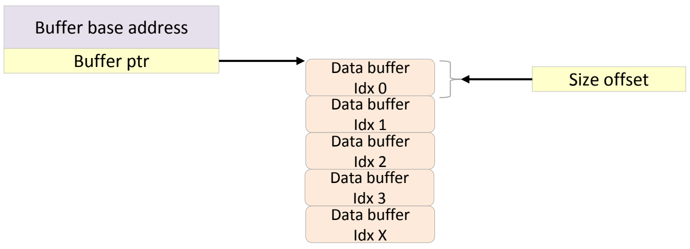


Figure 6-5 – Buffer area

6.2.3 Tx procedure

When an action is started (non-connected or connected events), the SW is in charge to prepare the packet to send and also pre-allocate the possible answer or the next packets. At the end, the SW sets the field CS-TXDESCPTR for its next use.

The allocation of the Tx descriptor and buffer is done by the link layer manager (ADVERTISING, SCANNING), the link layer controller (LLCP) or HCI (DATA) in split mode. In fullemb mode the host stack replace the HCI to allocate the Tx descriptor and buffer.

On a schedule interruption, the SW checks if it needs to program a new Tx packet, if yes, it takes the necessary amount of descriptor in the free pool, sets the TxDESC-MD field in case where more than one descriptor is allocated and clear the TxDESC-TXDONE.

When the SW has prepared a packet it sets the CS-TXDESCPTR to notify the HW that the packet is ready to be processed.

When the SW wants to send empty packet, it sets the field CS-TXDESCPTR to NULL. By this way the HW is indicated it has to send a null length packet.

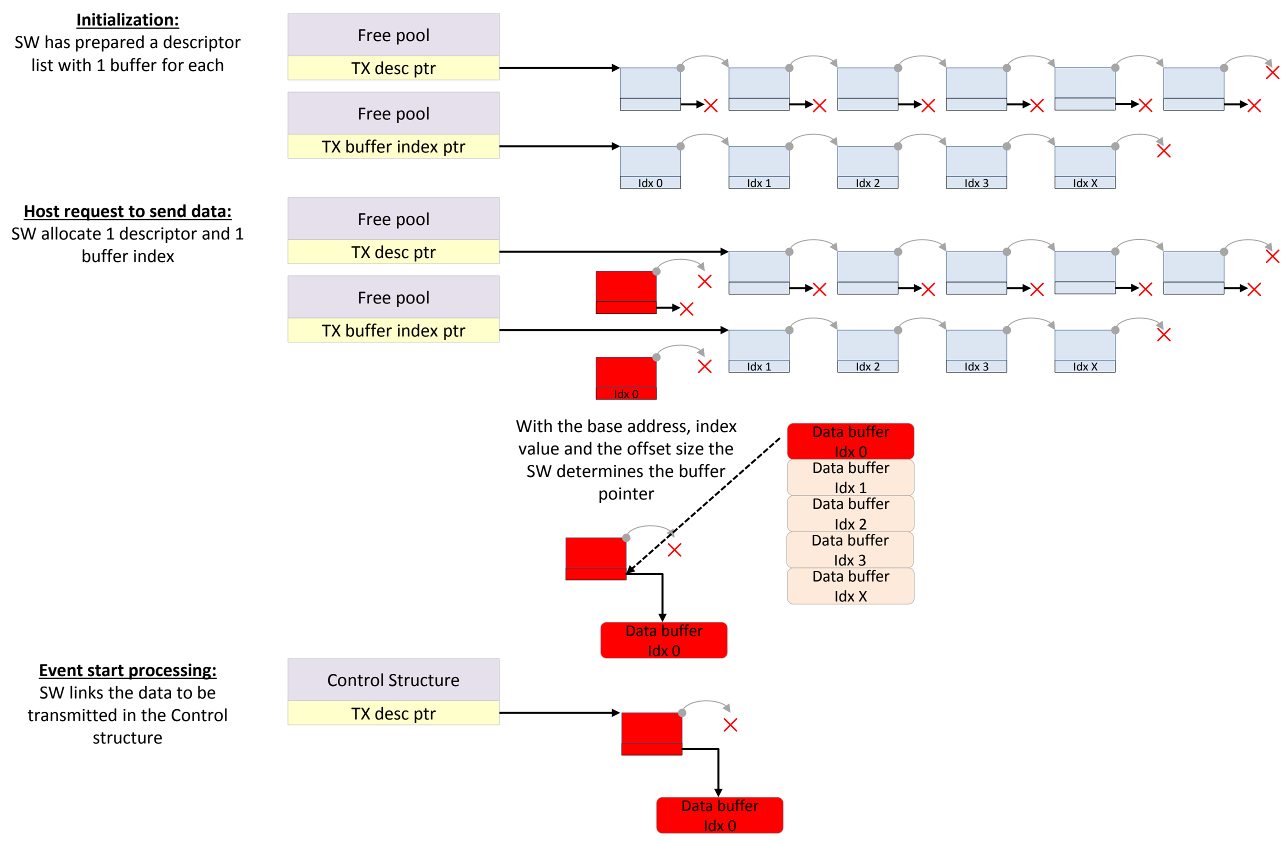


Figure 6-6 – Non fragmented transmission Tx procedure

The freeing is done at the end of event or Rx interruption.

6.2.3.1 Tx procedure with controller fragmentation

If the host request to send data with a length greater than the supported, the controller will send the data in several chunks.

The controller fragmentation mechanism allocate several descriptors pointing on multiple chunks of data buffer, the descriptors pointing on the firsts chunks have the MD (More Data) bit set to ***1*** and the flag free buffer set to ***false***. The latest descriptor, which is pointing on the latest chunk to send, has the MD set to ***0*** and the free buffer flag set to ***true***.

In this case the freeing of the data buffer is done only when the latest chunk is acknowledged (on end of event or Rx interruption).

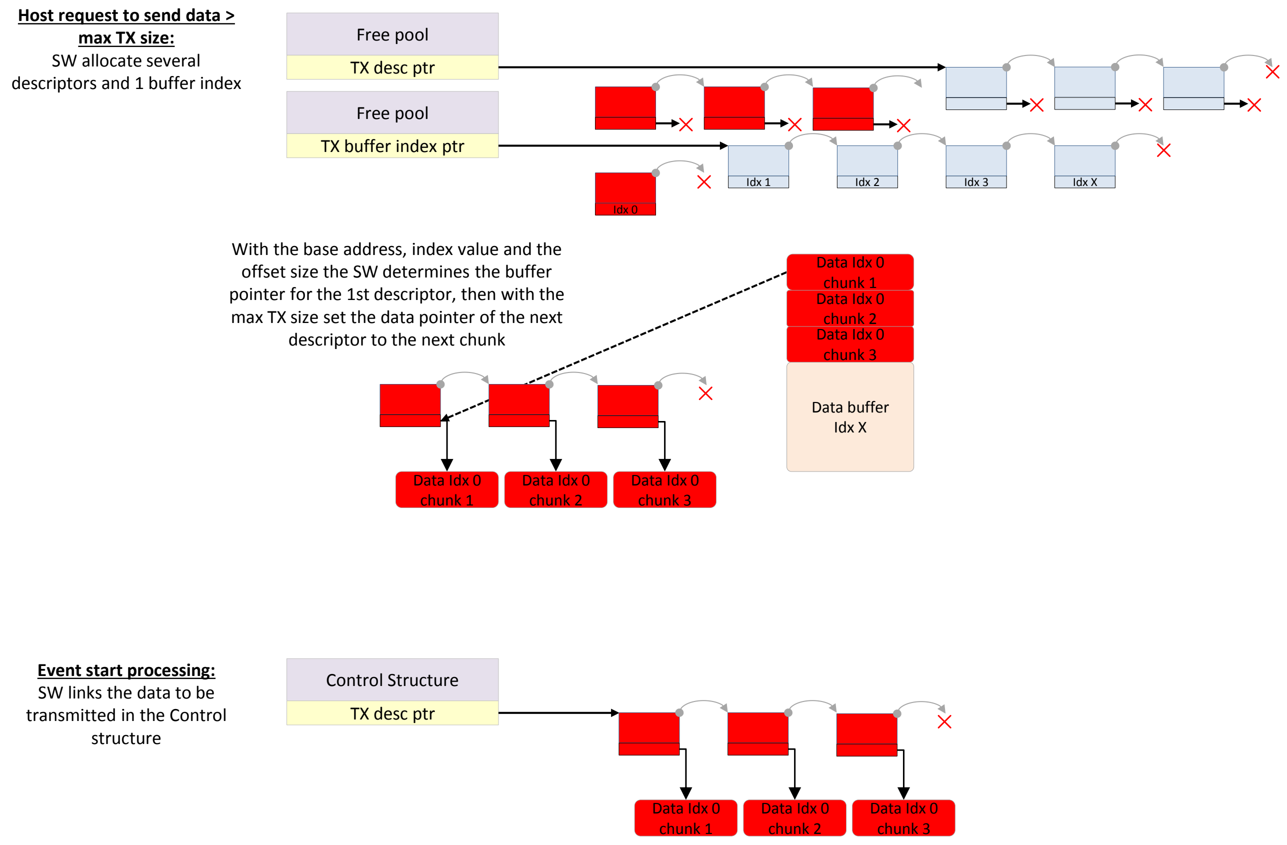

Host request to send data > max TX size:
SW allocate several descriptors and 1 buffer index
Free pool
TX desc ptr
Free pool
TX buffer index ptr
Idx 0
Idx 1
Idx 2
Idx 3
Idx X
With the base address, index value and the offset size the SW determines the buffer pointer for the 1st descriptor, then with the max TX size set the data pointer of the next descriptor to the next chunk
Data Idx 0 chunk 1
Data Idx 0 chunk 2
Data Idx 0 chunk 3
Data buffer Idx X
Event start processing:
SW links the data to be transmitted in the Control structure
Control Structure
TX desc ptr

Figure 6-7 – Fragmented Tx procedure

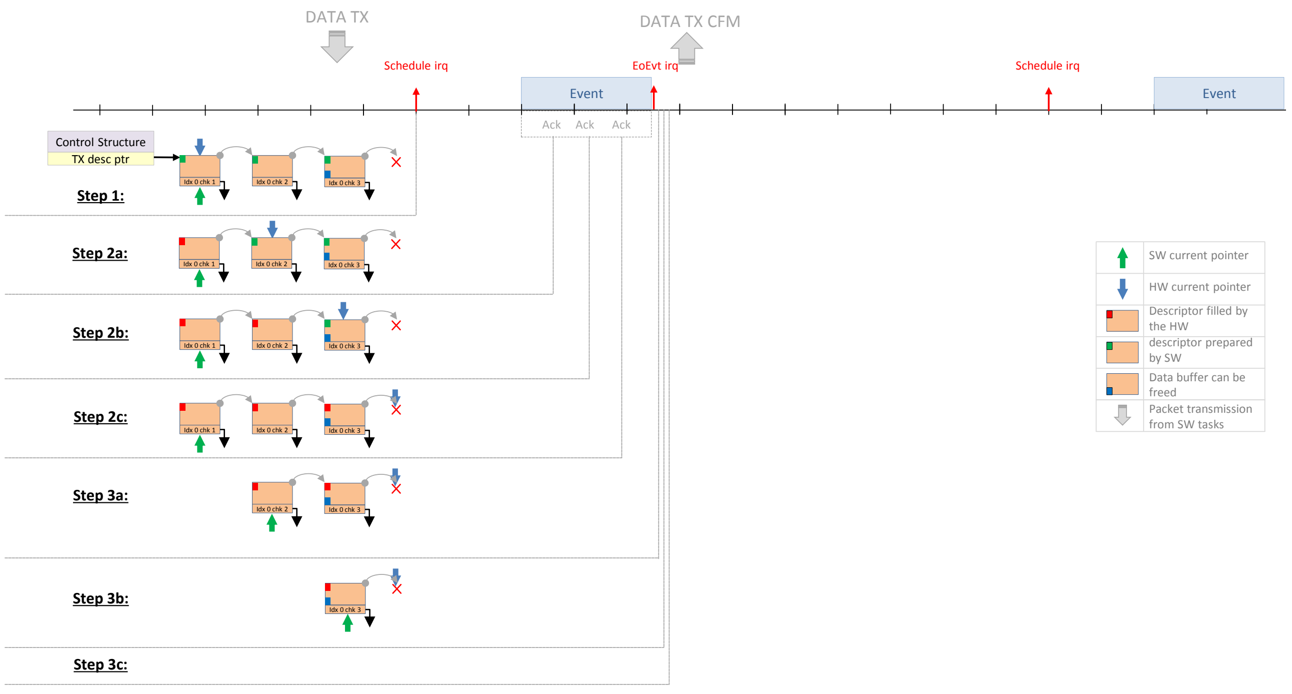


Figure 6-8 – HW/SW interaction for a fragmented Tx procedure

Step 1: On Schedule interrupt the SW set the descriptor pointer filed of the control structure on the first descriptor of the list.

Step 2a: When the event start the HW sends the data chunk linked to the 1st descriptor, if the packet is acknowledged by the peer, the Tx done bit field bit is set, then the next pointer field is checked.

Step 2b: If a 2nd descriptor is present, the HW sends the data chunk linked to this descriptor, if the packet is acknowledged by the peer, the Tx done bit field bit is set, then the next pointer field is checked.

Step 2c: If a 3rd descriptor is present, the HW sends the data chunk linked to this descriptor, if the packet is acknowledged by the peer, the Tx done bit field bit is set, then the next pointer field is checked, if NULL the HW stops the event and raises an End of Event interruption.

Step 3a: When the event end the HW raise an interruption the SW checks the TX done status bit and frees only the descriptor, like the flag free data buffer is not set, then goes to the next descriptor.

Step 3b: The SW checks the TX done status bit and frees only the descriptor, like the flag free data buffer is not set, then goes to the next descriptor.

Step 3c: The SW checks the TX done status bit and free the descriptor and the data buffer, like the flag free data buffer is set, then a confirmation is sent to the upper layers.

7 Event Arbiter

The Event Arbiter is the real time scheduler. See [6].

8 Low Energy Controller Functionalities

The LE LL uses the active clock accuracy during connection or advertising event, else it uses the sleep clock accuracy.

The LE LL follows the Little Endian format to create the packet sent over the air, only the CRC is transmitted, by the HW, most significant bit first.

All packets on an LE LL are not affected by the automatic flush timer.

The LE LL transport and L2CAP logical link between two devices is set up using the Link Layer Protocol. The Link Layer provides the delivery order of data packets. No more than one Link Layer channel exists between two devices.

The LE LL always provides the impression of full-duplex communication channels. This implies that all L2CAP communications are bi-directional.

The LE LL performs data integrity checks and resends data until it is successfully acknowledged, or a timeout occurs.

8.1 Role Combination and states

The table below shows all the combinations supported for the RW-BLE SW stack (see [1] volume 6 part B Chapter 1.1).

State		**Advertiser**	**Scanner**	**Initiator**	**Connected**	
		Non-connected	**Non-connected**	**Non-connected**	**Role Master**	**Role Slave**
Advertiser	**Non-connected**	Prohibited	Not Supported	Not Supported	Supported	Supported
Scanner	**Non-connected**	Not Supported	Prohibited	Not Supported	Supported	Supported
Initiator	**Non-connected**	Not Supported	Not Supported	Prohibited	Supported	Supported
Connected	**Role Master**	Supported	Supported	Supported	Supported	Supported
	Role Slave	Supported	Supported	Supported	Supported	Supported

Table 8-1 - Roles and states combination

To make the analogy with the FW, when a link layer is in the connection state, it may operate in Master and Slave role at the same time.

In summary, the RW-BLE SW stack does not support two operations in non-connected states (Advertising, Scanning and Initiating) at the same time.

8.1.1 States transition

The Link Layer allows only one state to be active at a time. It can operate in five different states:

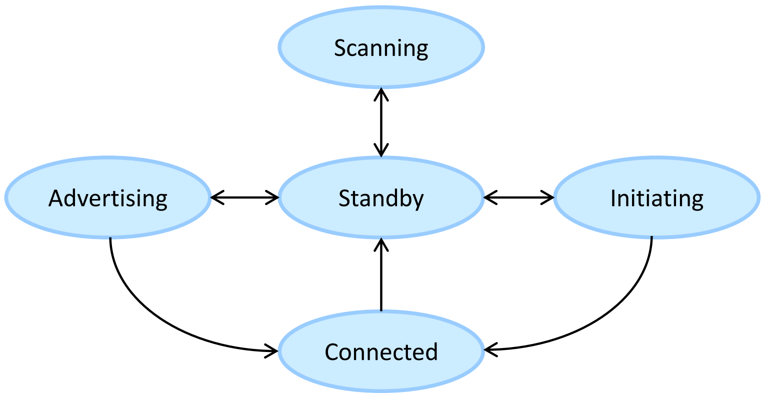


Figure 8-1 - States transition

Certain combinations of states and roles are prohibited see Table 8-1 - Roles and states combination.

8.2 Features support

The signification of the supported features depends on the source.

The bit positions for each Link Layer Feature are:

Bit position	**Link layer feature**
0	Encryption
1	Connection Parameters Request Procedure
2	Extended Reject Indication
3	Slave-initiated Features Exchange
4	LE Ping
5	LE Data Packet Length Extension
6	LL Privacy
7	Extended Scanner Filter Policies
8 - 63	Reserved

Table 8-2 - Bit features

8.3 Filtering

The Link Layer may perform device filtering based on the device address of the peer device. Link Layer Device Filtering is used by the Link Layer to minimize the number of devices to which it responds.

- ✓ **White list**: The set of devices that the Link Layer uses for device filtering.
- ✓ **Advertising filter policy**: determines how the advertiser's Link Layer processes scan and/or connection requests.
- ✓ **Scanner filter policy**: determines how the scanner's Link Layer processes advertising packets.
- ✓ **Initiator filter policy**: determines how an initiator's Link Layer processes advertising packets.

Filtering operations are configured by SW and done by HW see [2].

8.3.1 Resolving Address List (Enhanced Privacy)

The enhanced privacy (version 1.2) also called controller privacy is used to hide device identity using a private resolvable address (see 8.3.1.1). When enabled in the controller, this feature will determine if a peer device hidden with a resolvable private address is known or not. Even if a device is found or not in the resolving list, filtering is applied according to non-connected mode parameter using white list mechanism.

The private resolvable address is generated using an identity key (IRK), this IRK is unique to a bond and is provided to a peer device during pairing and can be resolved only by device that have this IRK. This address shall be regenerated after a configurable timer.

Note: A device is called “private” when it never exposes its identity in non-connected mode.

To retrieve a private peer device, host has to populate the resolving address list (see 8.3.1.2) with peer device identity, peer IRK and local IRK exchange during pairing phase.

8.3.1.1 LE Address

There are two types of Bluetooth LE addresses

1. Static Address
    - Two most significant bits are equal to 1
    - All the other bits are not “all 0s” nor “all 1s”
2. Private Address
    1. Non-resolvable Address
        - Two most significant bits are equal to 0
        - All the other bits are not “all 0s” nor “all 1s”
    2. Resolvable Address
        - 2 MSB are equal to 01
        - 22 MSB remaining are random
        - 24 LSB hash is calculating with AES using IRK and 24 MSB of the address

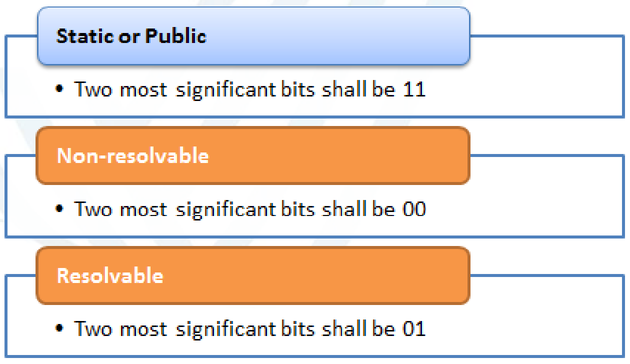

Static or Public
• Two most significant bits shall be 11
Non-resolvable
• Two most significant bits shall be 00
Resolvable
• Two most significant bits shall be 01

Figure 8-2 - LE Address types

8.3.1.2 Resolving Address List

The resolving address list (RAL) is populated by host. It contains a peer device identity, corresponding local and peer device IRK (exchange during a pairing).

If peer or local IRK has not been exchange, the missing IRK shall be replaced by zero values. In that case, it means that device should use its identity in order to be retrieve by resolving mechanism (see [1] Volume 6 Part B chapter 6.5 Table 6.1).

Number of device into the resolving address list should be sized in order to allow address resolutions and address generation within IFS.

The address resolution is transparent for the software because it's fully managed by the hardware. When address resolution succeeds, pointer to element of resolving list used to resolve peer address is set in RX descriptor (RX-RXRALPTR).

A bit is present in each element of the resolving address list in order to inform if peer identity is present in white list or not. This feature is used to prevent HW to check the white list when a known device has been found.

Address renewal generation is performed by HW, software has just mark which address to renewal timeout expires.

A bit is present for black list management that should be set by software when a connection is established and cleared when link disconnected. This is used to reject any connection-less packets from a connected device.

In Scan mode, if an undirected advertisement packet is received, an HCI_le_advertising_report_event is triggered with peer device identity instead of the resolving private address.

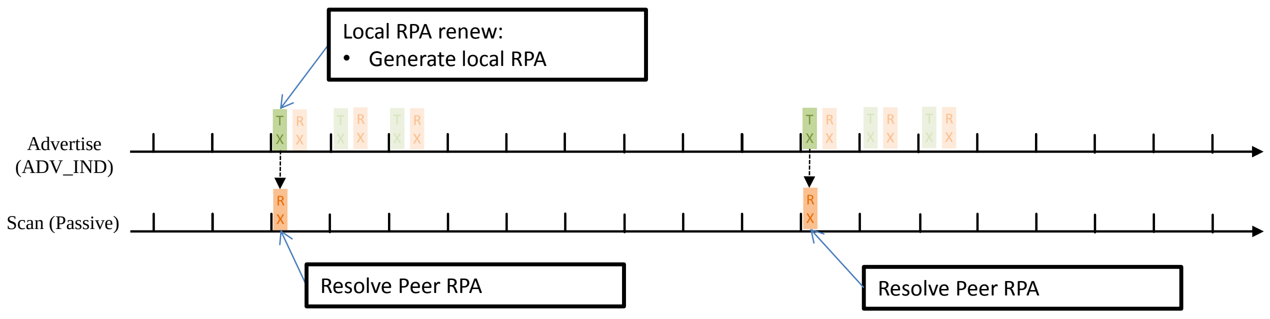


Figure 8-3 – Enhanced privacy - Passive Scan

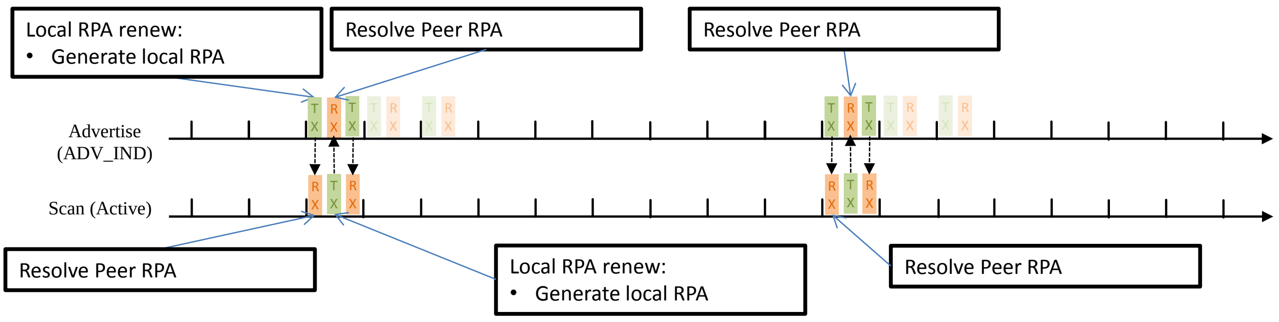


Figure 8-4 – Enhanced privacy – Active Scan

If a direct advertising packet is received by scanner, and a RPA is present in initiator address, an HCI_le_direct_advertising_report_event is triggered with peer device identity if resolved or advertiser address and RPA present in initiator address.

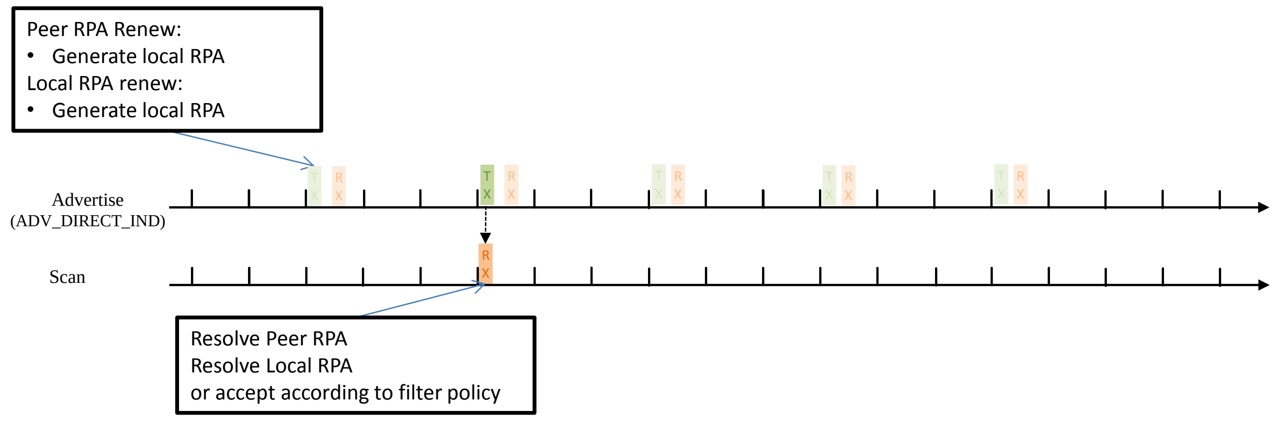


Figure 8-5 – Enhanced Privacy – Scan for Direct Advertising Packet

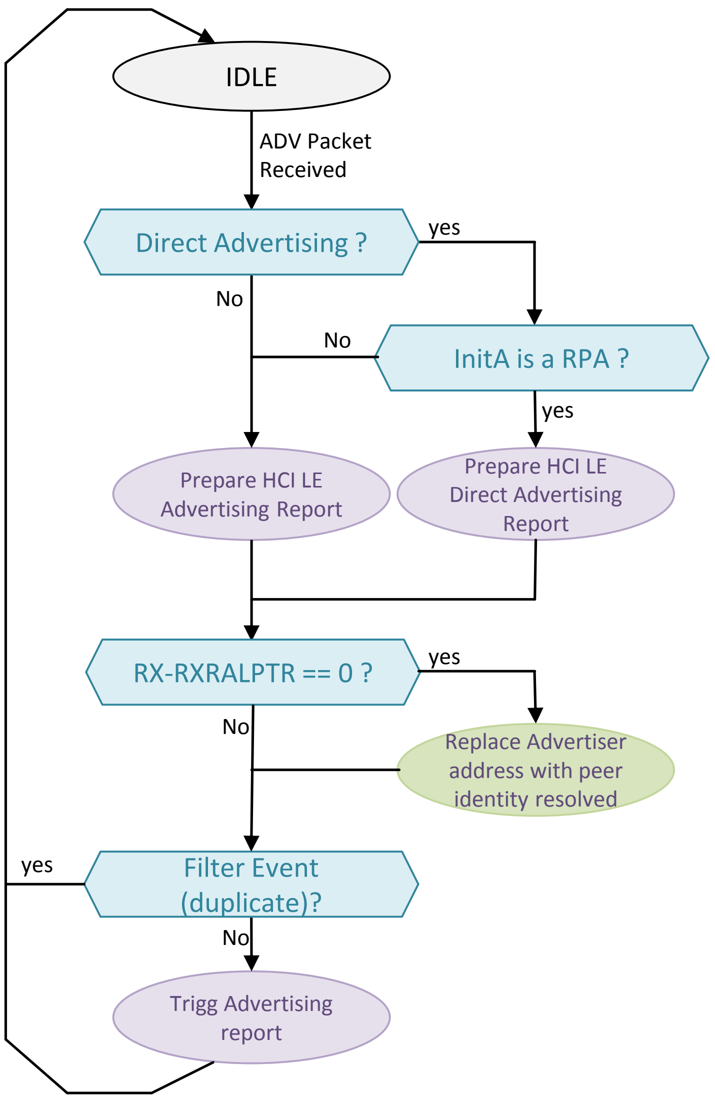

IDLE
ADV Packet Received
Direct Advertising ?
yes
No
InitA is a RPA ?
No
yes
Prepare HCI LE Advertising Report
Prepare HCI LE Direct Advertising Report
RX-RXRALPTR == 0 ?
yes
No
Replace Advertiser address with peer identity resolved
Filter Event (duplicate)?
yes
No
Trigg Advertising report

Figure 8-6 Scan mode state machine when advertising report received

When a connection is established in enhanced privacy mode and peer device is resolved, an HCI_LE_Enhanced_Connection_Complete_Event is issued with peer device identity and advertiser and initiator address used for connection establishment.

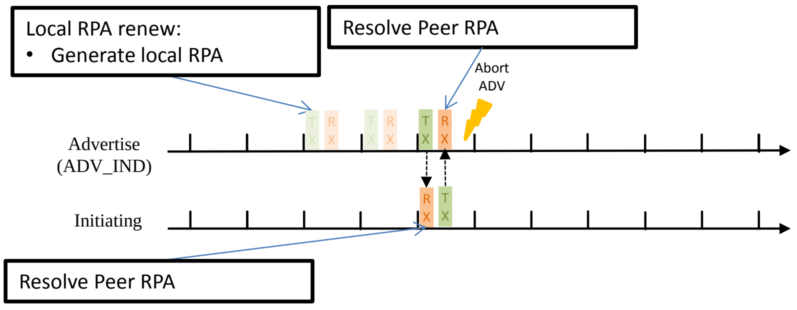


Figure 8-7 – Enhanced Privacy – Initiating a connection

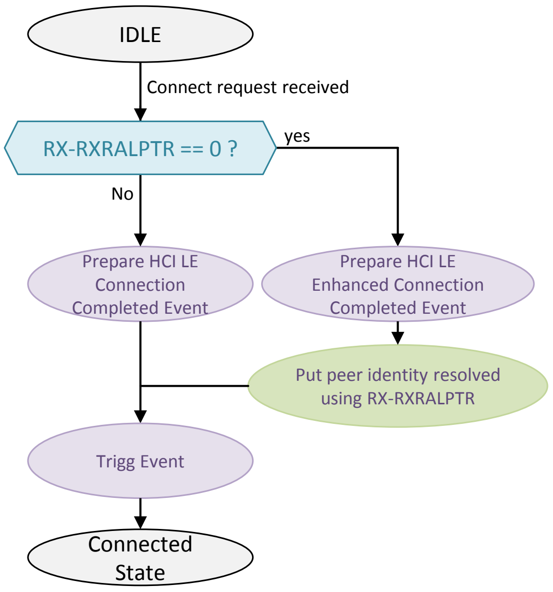

IDLE
Connect request received
RX-RXRALPTR == 0 ?
yes
No
Prepare HCI LE Connection Completed Event
Prepare HCI LE Enhanced Connection Completed Event
Put peer identity resolved using RX-RXRALPTR
Trigg Event
Connected State

Figure 8-8 Advertising state machine when connection request received

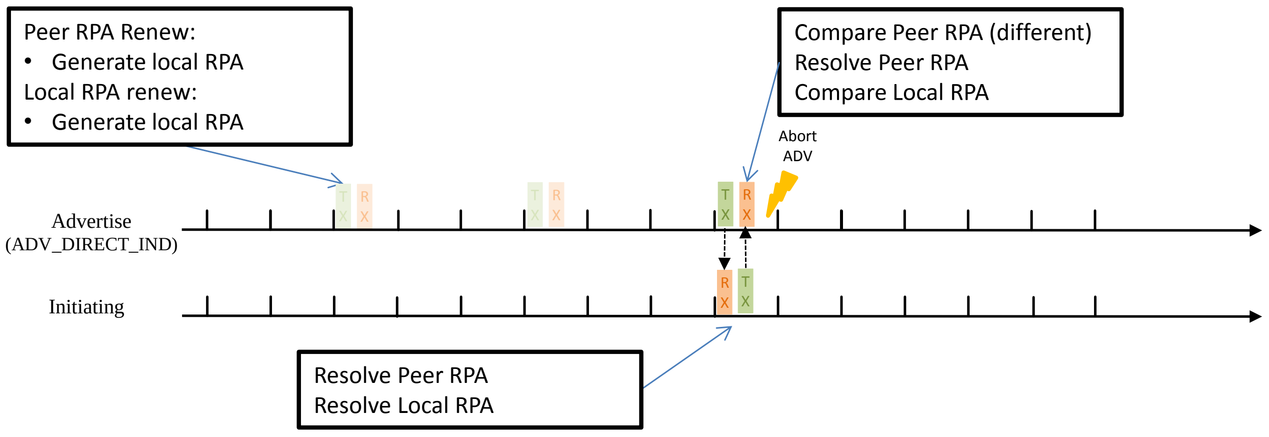


Figure 8-9 – Enhanced Privacy – Initiating a connection with direct advertiser

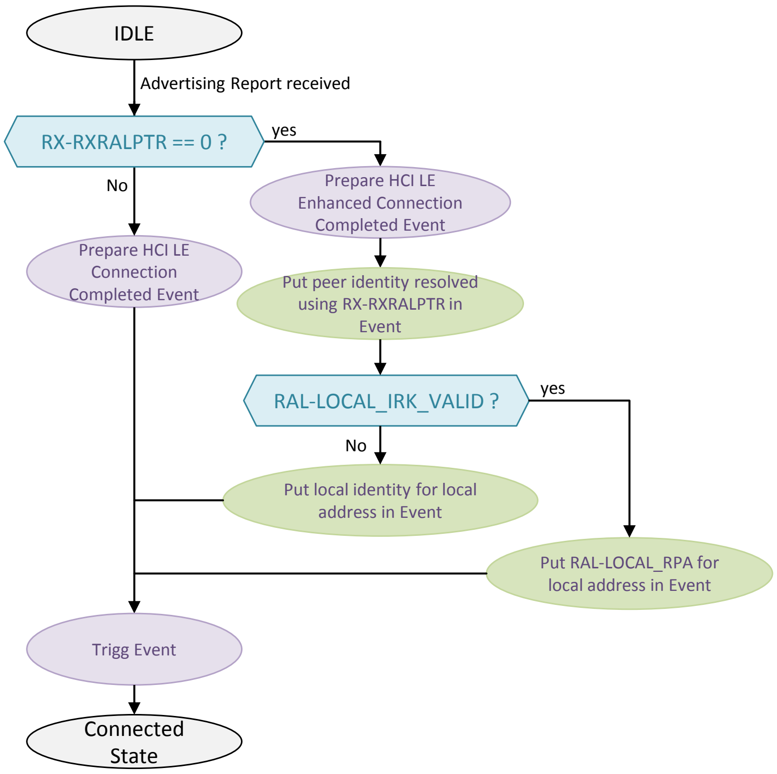

IDLE
Advertising Report received
RX-RXRALPTR == 0 ?
yes
No
Prepare HCI LE Enhanced Connection Completed Event
Prepare HCI LE Connection Completed Event
Put peer identity resolved using RX-RXRALPTR in Event
RAL-LOCAL_IRK_VALID ?
yes
No
Put local identity for local address in Event
Put RAL-LOCAL_RPA for local address in Event
Trigg Event
Connected State

Figure 8-10 Initiating State Machine when advertising report received

Note: if resolving address list is not able to recognize peer device, normal white list mechanism is used for filtering.

8.3.2 White list

Two White Lists are managed by the FW and used by the HW (Private and Public).

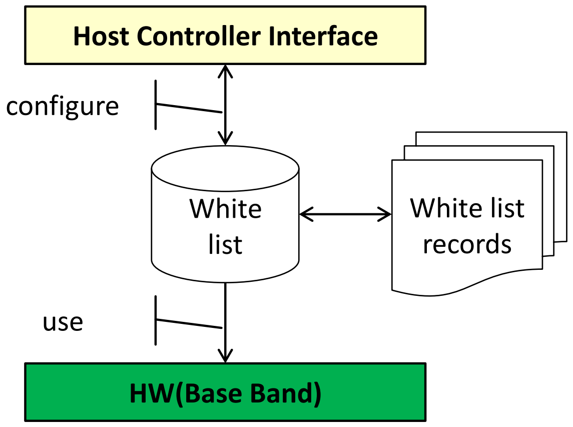


Figure 8-11 - While list overview

The White Lists contained in the LL are not persistent across power cycles or Controller resets.

For devices supporting device filtering, the controller’s LL supports adding White List records to the White List and deleting all the White List records.

The SW provides APIs to the host to add, remove and set the devices addresses and the filtering policies (describes in [1] Volume2 Part E chapter 7.8).

The HCI commands used by the host to access to the White List are:

- ✓ LE Read White List Size Command
- ✓ LE Clear White List Command.
- ✓ LE Add Device To White List Command.
- ✓ LE Remove Device from White List Command.

The algorithms described in chapters hereafter are managed by the HW depending on the policy set by the host; this management cannot be done by the SW due to the short response time.

8.3.3 Advertising filter policy

The figures bellows describe the filtering action, set by HCI command, done by the HW on SCAN_REQ or CONNECT_REQ PDU reception in Advertising state.

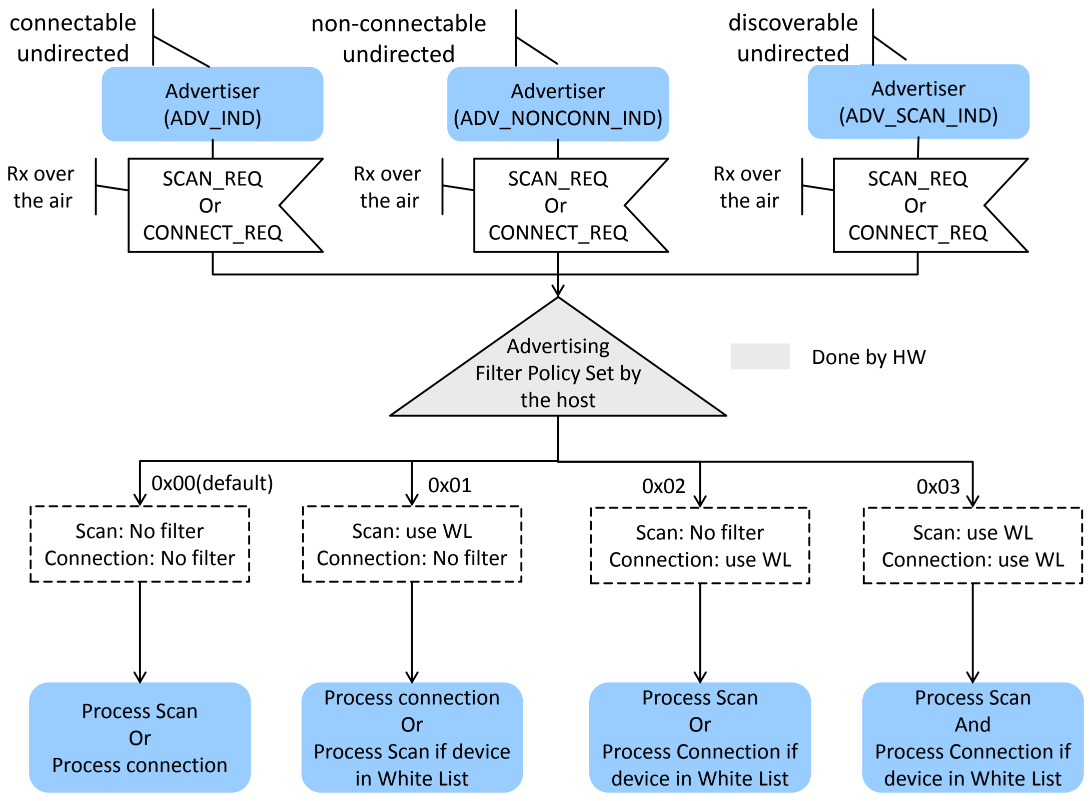

connectable undirected
non-connectable undirected
discoverable undirected
Advertiser (ADV_IND)
Advertiser (ADV_NONCONN_IND)
Advertiser (ADV_SCAN_IND)
Rx over the air
SCAN_REQ Or CONNECT_REQ
Rx over the air
SCAN_REQ Or CONNECT_REQ
Rx over the air
SCAN_REQ Or CONNECT_REQ
Advertising Filter Policy Set by the host
Done by HW
0x00(default)
0x01
0x02
0x03
Scan: No filter Connection: No filter
Scan: use WL Connection: No filter
Scan: No filter Connection: use WL
Scan: use WL Connection: use WL
Process Scan Or Process connection
Process connection Or Process Scan if device in White List
Process Scan Or Process Connection if device in White List
Process Scan And Process Connection if device in White List

Figure 8-12 - Advertising filter policy

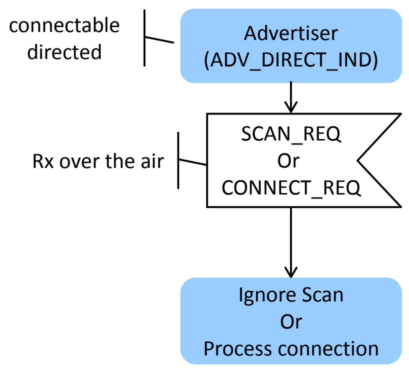


Figure 8-13 – Directed Advertising filter policy

8.3.4 Scanning filter policy

The figure bellow describes the filtering action, set by HCI command, done by the HW on ADV_XXX PDU reception in Scanning state.

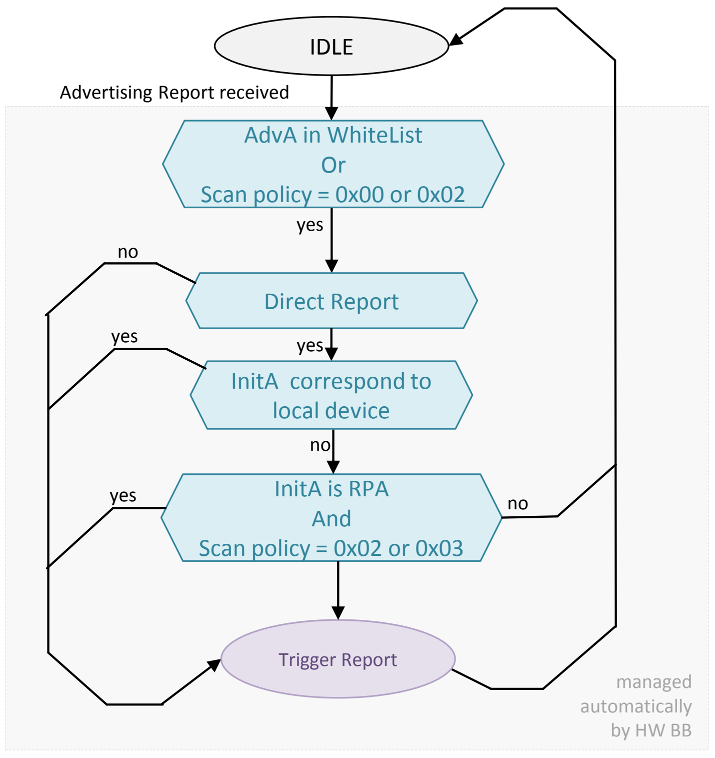


Figure 8-14 - Scanner filter policy

The Directed advertising packets which are not addressed for this device or initiator address is not an RPA are ignored.

8.3.5 Initiator filter policy

The figure bellow describes the filtering action, set by HCI command, done by the HW on ADV_XXX PDU reception in Initiating state.

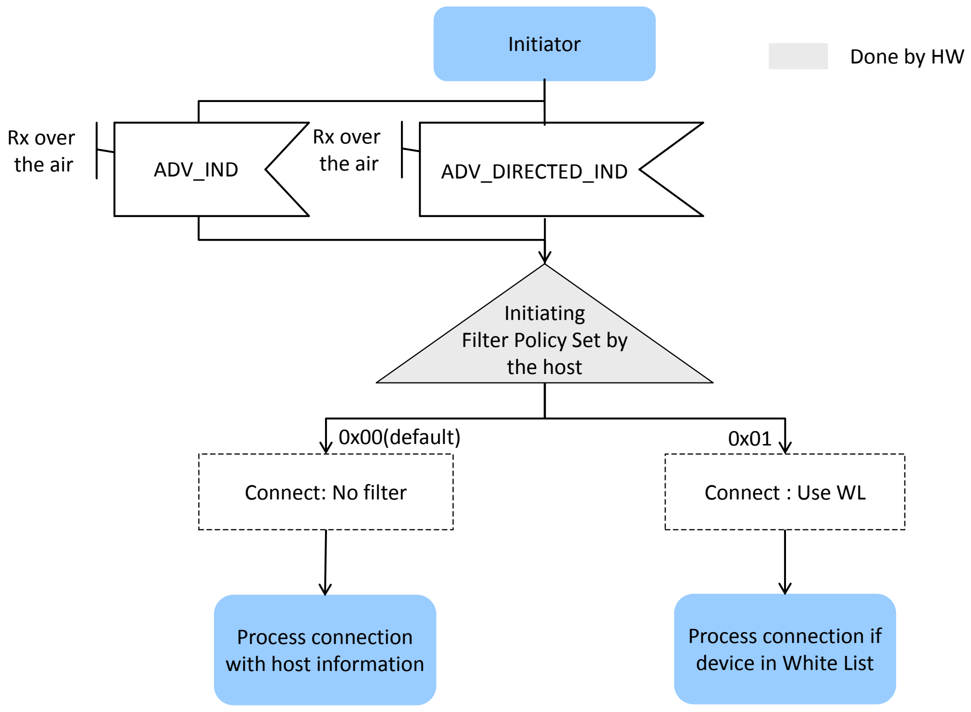


Figure 8-15 - Initiator filter policy

8.4 Link Layer partitioning

The link layer is split in several blocks:

1. Link Layer Controller (LLC) – Link oriented function and multi-instantiated.
2. Link Layer Manager (LLM) – Non-connected oriented function and mono-instantiated.
3. Link Layer Driver (LLD) – baseband driver function.

All blocks, except the LLD, are defined as kernel task. Those blocks are used to do the BLE protocol functionalities.

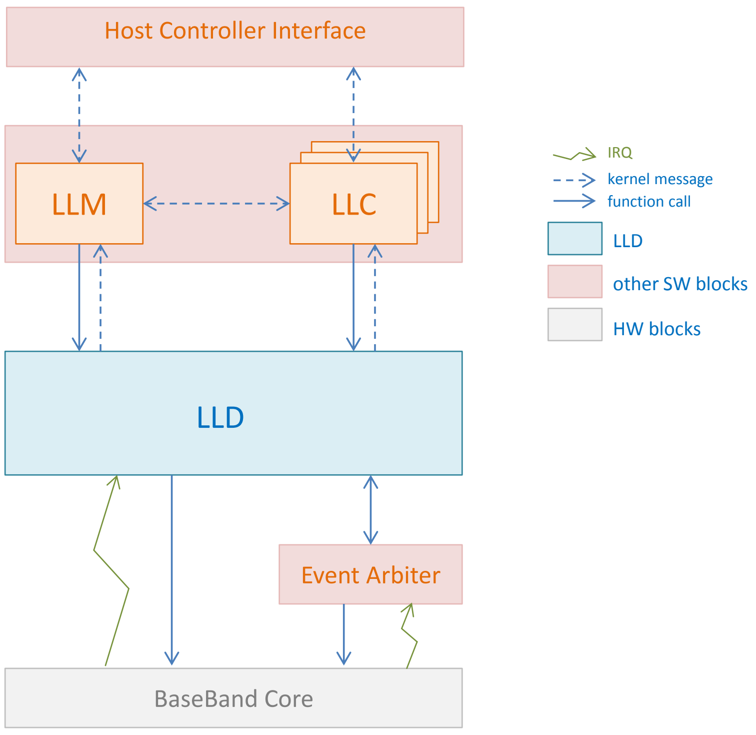


Figure 8-16 - Link Layer Architecture Overview

8.4.1 Link Layer Manager

The LLM is used for the advertising, scanning and initiating modes, and common functionalities not dedicated for one link (e.g: read local name, read local features…) The LLM process is mono-instantiated and works on OS time.

It is also in charge to change the channel map based on information passed by the host. The information is sent to the peers over the LLC.

At the initialization, a function is called to create and set the LLM task in initialization state.

Once Initialization is done, the LLM task goes in idle mode and waits a request from the host or the peer.

The LLM manages the HCI commands not related to a link; it takes care of the parameters and returns the status according to the specification.

The chapter 8.5 describes the LLM functionalities, states through MSCs.

8.4.2 Link Layer Controller

The LLC is used to control active link, all the PDU dedicated to the control and data go through it. The LL Controller process is multi-instantiated and works on OS time.

At the initialization, a function is called to create and set the LLC task in initialization state.

Once Initialization is done, the LLC task is in connected mode and waits a request from the host or the peer.

The LLC manages the HCI commands related to one link; it takes care of the parameters and returns the status according to the specification.

It is in charge of the procedures:

- Connection update: used to update connection parameters (interval, slave latency and supervision timeout).
- Encryption: used for security.
- Feature exchange: used to know peer supporting features in order to start authorized procedures.
- Version exchange: use to exchange the company identifier, link layer version (BT 1.2, 2.0,…).
- Termination: used for voluntary termination of a connection.
- Data transfer: used by the higher layers to communicate with the peer.
- Connection parameters request: used to request an update connection parameters (interval, slave latency and supervision timeout).
- LE Ping: used to verify message integrity on the link by forcing the remote device to send a data packet that contains a valid MIC.

The chapter 8.6 describes the LLC functionalities, states through MSCs.

8.4.3 Link Layer Driver

The LLD is used to manage the scheduling. It works on Real time (Interruption), and kernel events.

On Master side, one descriptor is defined by link; in this case it easier to manage the multi-link topology.

The LLD communicates with the link layer manager by message, and with the baseband by an API defined in the chapter 5.5.1.

The chapter 8.5 and 8.6 describe the LLD functionalities through MSCs.

8.4.3.1 Event environment

To maintain a link, or execute non–connected state the link layer driver save the information for every link and non-connected state in a descriptor defined in the figure below.

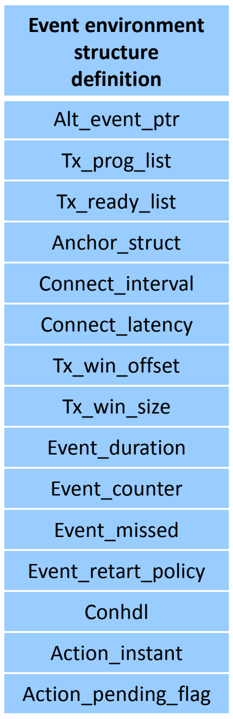


Figure 8-17 – Event environment: structure

Parameters	Description
Alt_event_ptr	Pointer to the alternate event used for connection update mechanism
Tx_prog_list	List of TX descriptors programmed for transmission (i.e. chained with the CS)
Tx_ready_list	List of TX descriptors ready for transmission (i.e. not yet chained with the CS)
Anchor_struct	Base time counter value of the Anchor point, in multiple of slot (625us)
	Fine time counter value of the Anchor point, in multiple of microsecond (range [0 to 624])
Connect_interval	Connection interval for this event, in multiple of 1.25ms
Connect_latency	Slave latency for this event (range [0 to 499])
Tx_win_offset	Delay before starting the transmit window for a connection or an update, in multiple of 1.25 ms(range [0 to connInterval])
Tx_win_size	Transmit window size for a connection or an update.in multiple of 1.25 ms (range [1.25 to min(10, (connInterval - 1.25))]).
Event_duration	Bandwidth used by the event, in multiple of slot (625us)
Event_counter	Event counter is used to synchronize Link Layer control procedures
Event_missed	Event missed is used to compute the window size.
Event_restart_policy	Restart event policy is used to reschedule or not the event
Conhdl	Connection handle is used to get the link dedicated control structure
Action_instant	Instant in event counter unit, where an action is performed
Action_pending_flag	Indicate that an action is pending.

Table 8-3 – Event environment: fields' description

8.4.3.2 Window widening

The slave resynchronizes onto the anchor point on every received packet regardless CRC error. This is done by CS-RXWINSZ field where the value of the window is set and used by the HW.

This window takes in account the sleep clock accuracy from the peer (Master), the local (Slave) as well as the value of the anchor point and the missing event(s).

According to the slow clock accuracy the value will be set in microseconds or multiple of 625us.

It is possible to lose the connection when the window size is higher than half the connection interval.

In Master connection state, the HW uses a dedicated register field (RWBLRCNTL-RXWINSZDEF) to keep the synchronization, where the value is set by the SW.

8.4.3.3 Channel selection

The SW is in charge to update the channel index and to provide the hopping offset to the Core over the Control structure.

The selection is done by an HW algorithm, with the information given by the SW. The hopping offset is computed by a random generator and fit within the [5 : 16] range. The index follow the formula:

index = (old_index + hopping offset) mod 37.

In case the index is not a "good" channel, a remapping algorithm is used; this algorithm is performed by the HW. The SW provides the HW the number of good channel used and the channel map in ascending order.

Those two values are set in the control structure.

8.4.3.4 Slave latency

Slave latency allows a slave to use a reduced number of connection events. The connection slave latency parameter defines the number of consecutive connection events that the slave device is not required to listen. (see [1] Volume 6 Part B chapter 4.5.1).

The LLD computes the next programming event by taking care of the latency used, it requires to:

- Update the channel index: **next_ch = (last_ch + (latency) * hop) % 37 (like the HW does automatically the hopping, the software compute the next channel one interval before).**
- Update the counter: **counter = latency + 1**.
- Update the time of the event: **next_basetimecnt = basetime_cnt + interval * (Latency+ 1).**

The calculation of the counter and next base time takes care of the counter wrapping.

8.5 Non connected states

The link layer is in charge to enter/escape from the scanning or advertising states.

8.5.1 Scan mode

The scanning state is used to detect the neighbor devices. In order to enter in scan mode, the host sends a command with the type of scanning:

- Passive: No extra request is sent.
- Active: A scan request packet is sent to obtain more information.

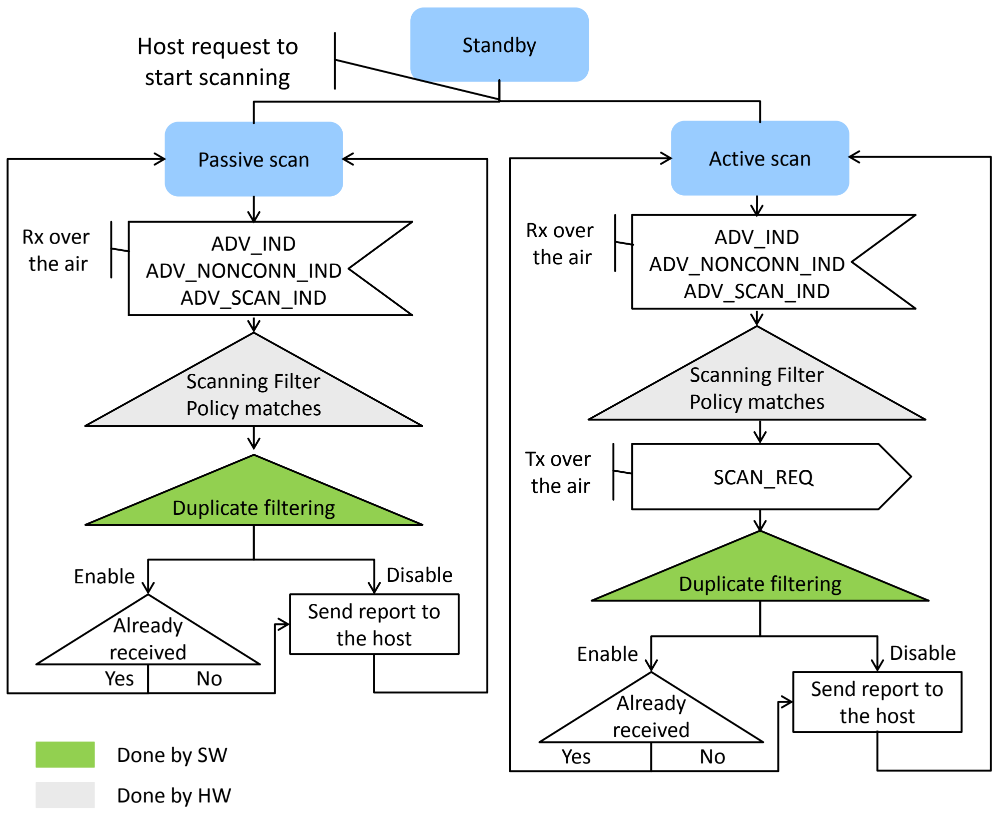


Figure 8-18 – Start scanning overview

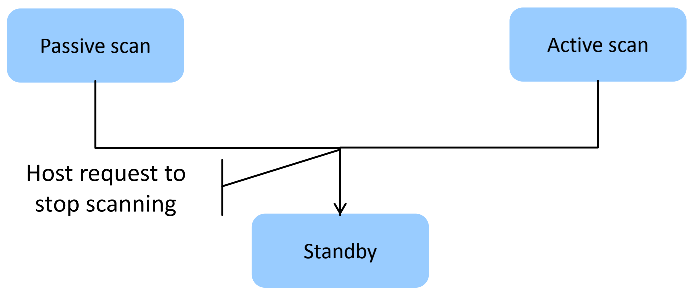


Figure 8-19 – Stop scanning overview

8.5.1.1 Scan mode MSC

The following MSCs present an overview of the layers' interactions during a start, stop scan, and advertising detection:

We have the following rules for the MSCs:

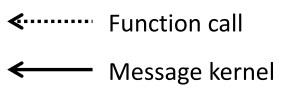


Figure 8-20 – MSC rules.

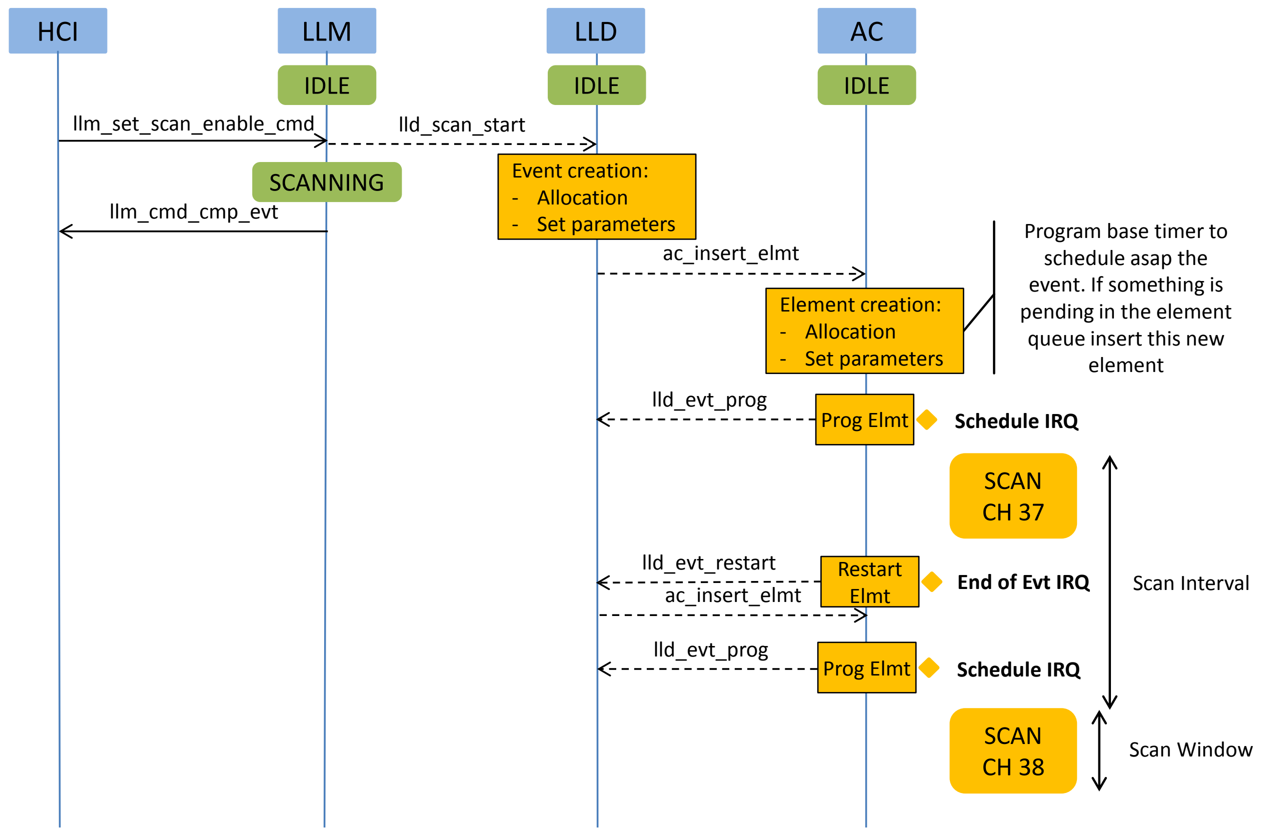


Figure 8-21 – Start Scan MSC

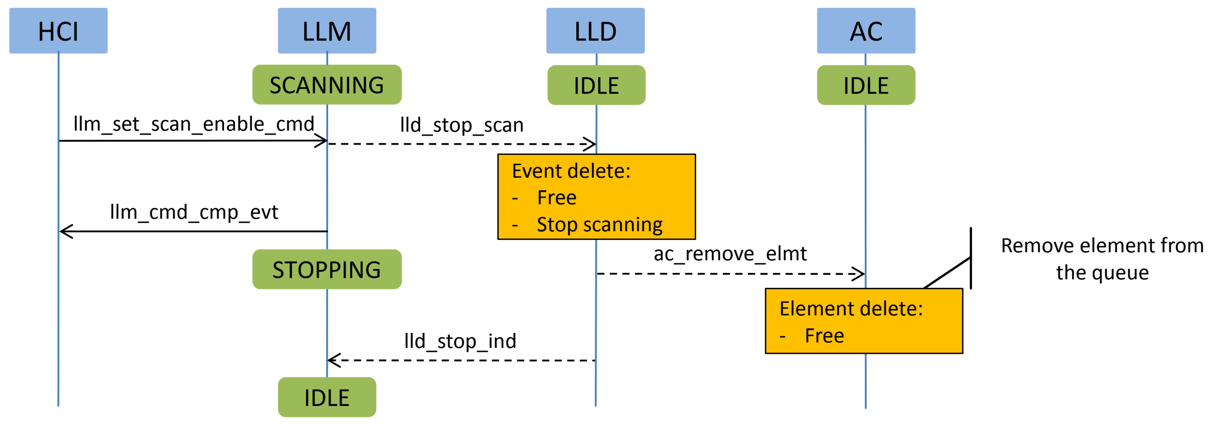


Figure 8-22 – Stop Scan MSC

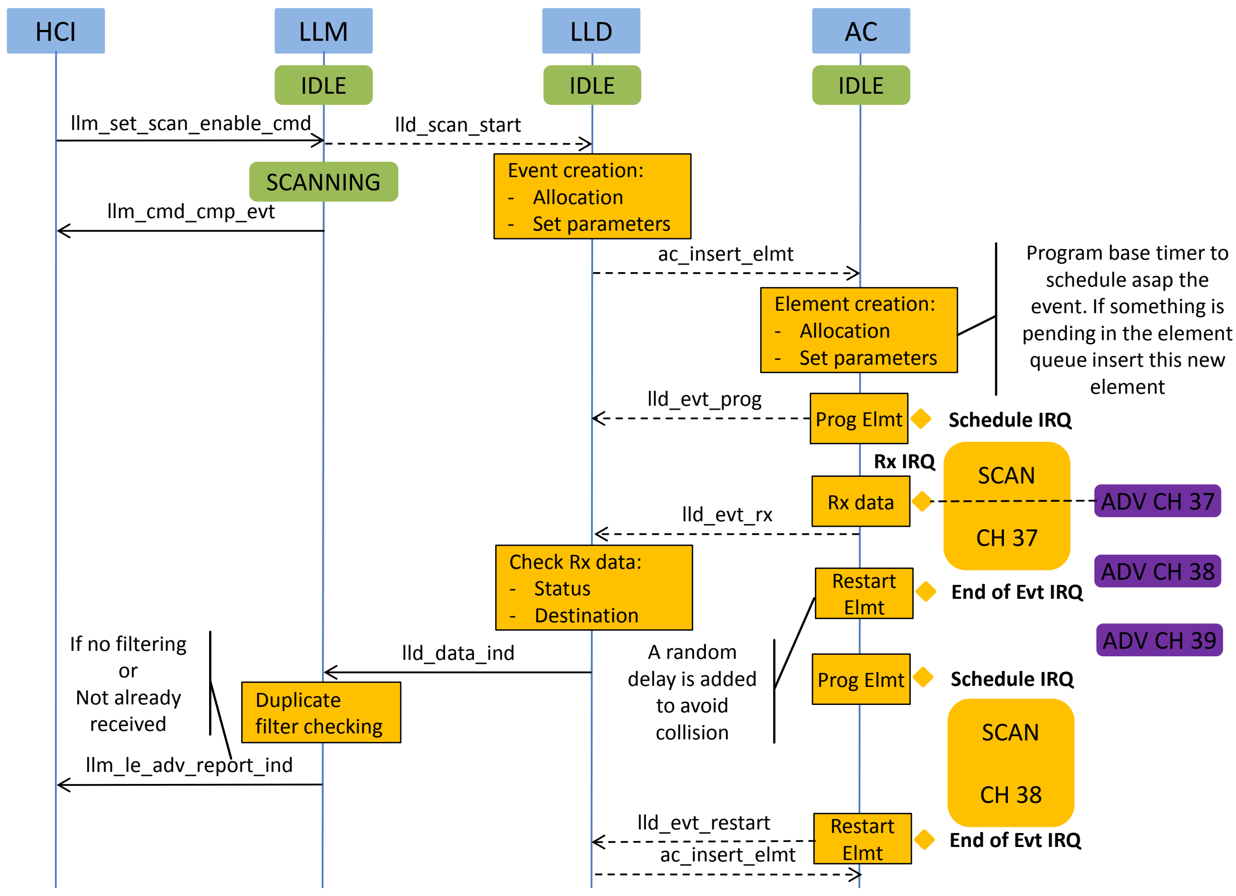


Figure 8-23 – Scan with ADV detection MSC

8.5.2 Advertise mode

The advertising state is used to be discoverable and/or connectable. In order to enter in advertise mode, the host sends a command with the type of advertising (set previously by the host), the advertising data and parameters, over APIs (defined [1] part E).

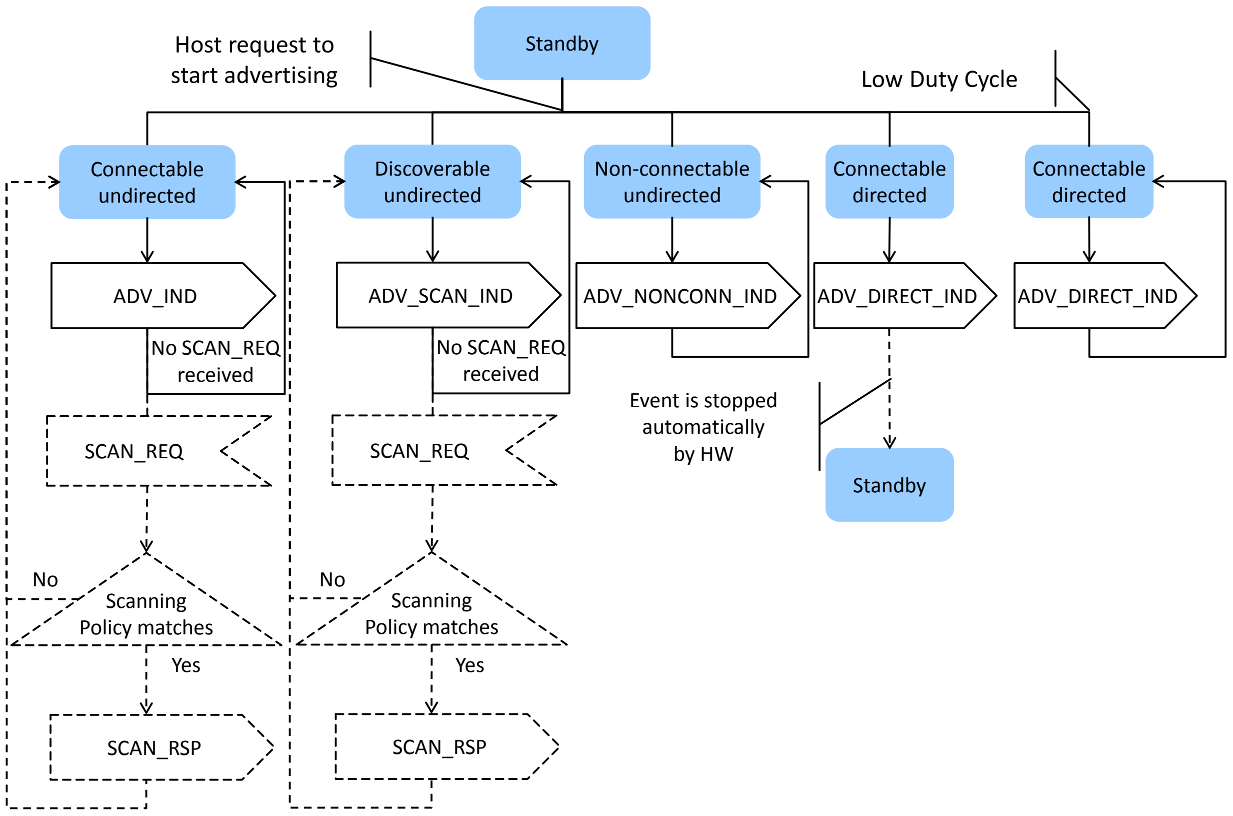


Figure 8-24 – Start Advertise overview

To stop an advertising event several ways can be chosen:

- By a stop command: Connectable undirected, Discoverable undirected and Non-connectable undirected, Connectable directed (Low Duty cycle).
- By a connection request: Connectable undirected, Connectable directed (High and Low Duty cycle).
- By HW timer: Connectable directed.

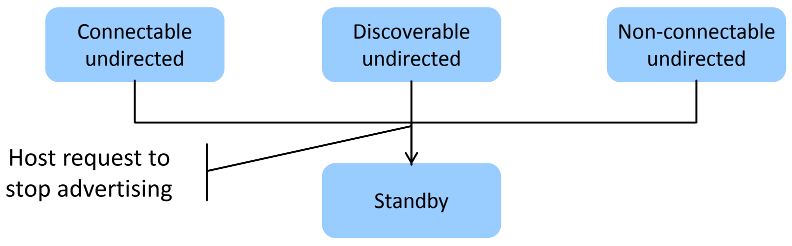


Figure 8-25 – Stop advertising by command overview

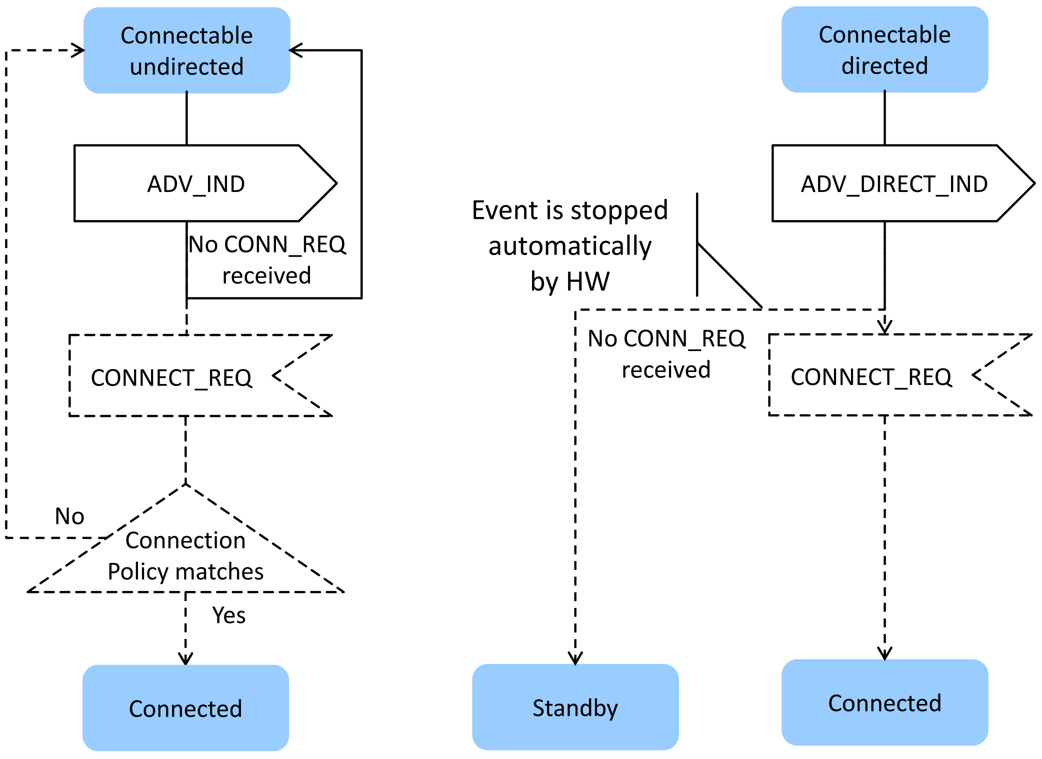


Figure 8-26 – Stop advertising by connection request overview

8.5.2.1 Advertise mode MSC

The following MSCs present an overview of the layers' interactions during a start, stop advertising:

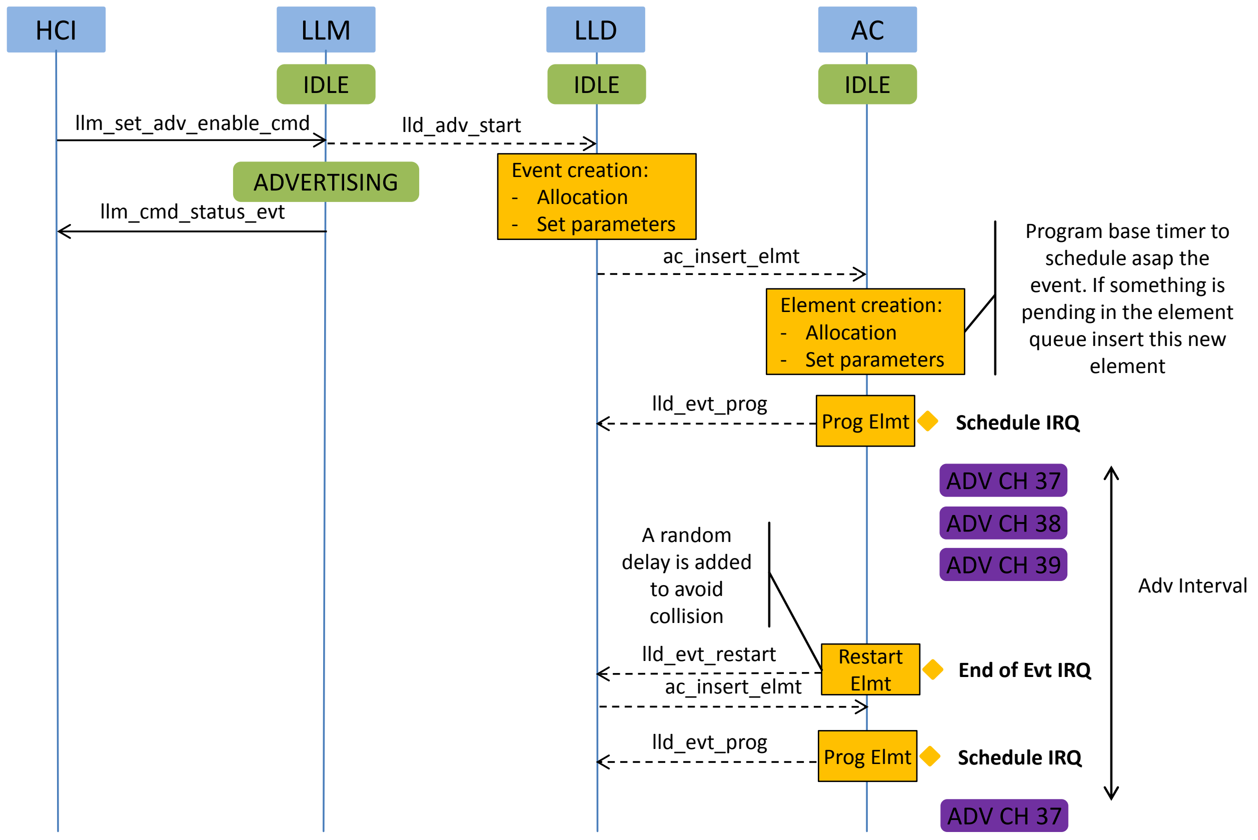


Figure 8-27 – Start Advertise MSC

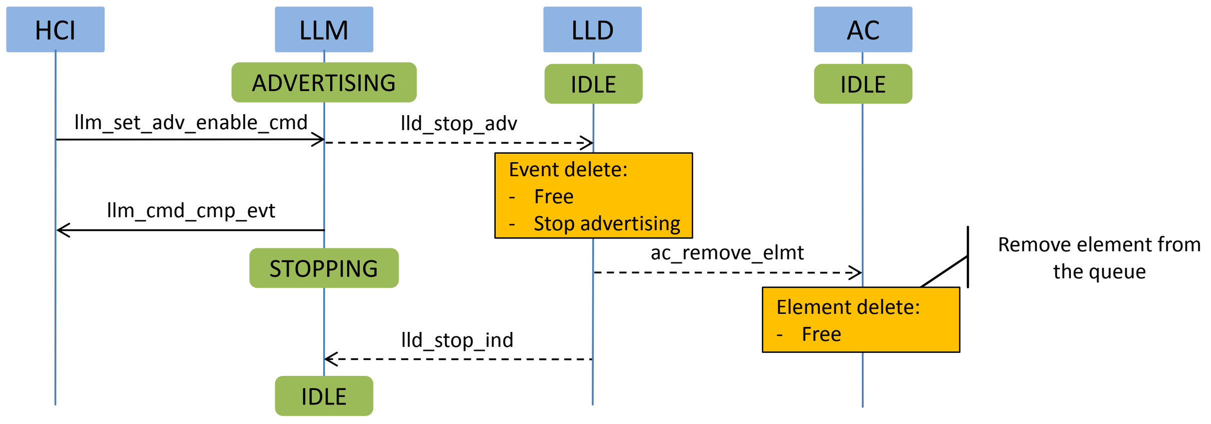


Figure 8-28 – Stop Advertise MSC

8.5.3 Initiating mode

The initiating state is used to start a BLE connection. In this state we have 3 sub-states:

- **Scanning** : Allows detect the Advertising packet from peer device.
- **Connecting** : In this state the connect request is sent and a connection timer is started.

- **Connected** : when the 1st empty packet is received, the connection timer is stopped and link supervision timer is started.

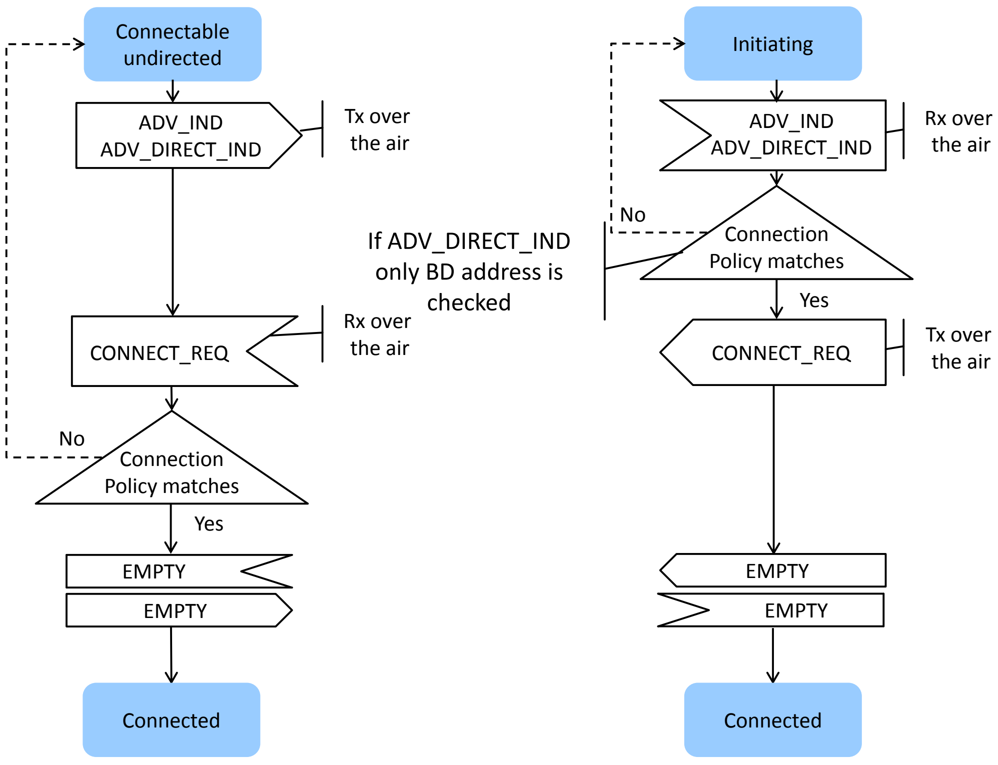


Figure 8-29 - Initiating overview

8.5.3.1 Initiating mode MSC

The following MSCs present an overview of the layers' interactions during a connection initiating:

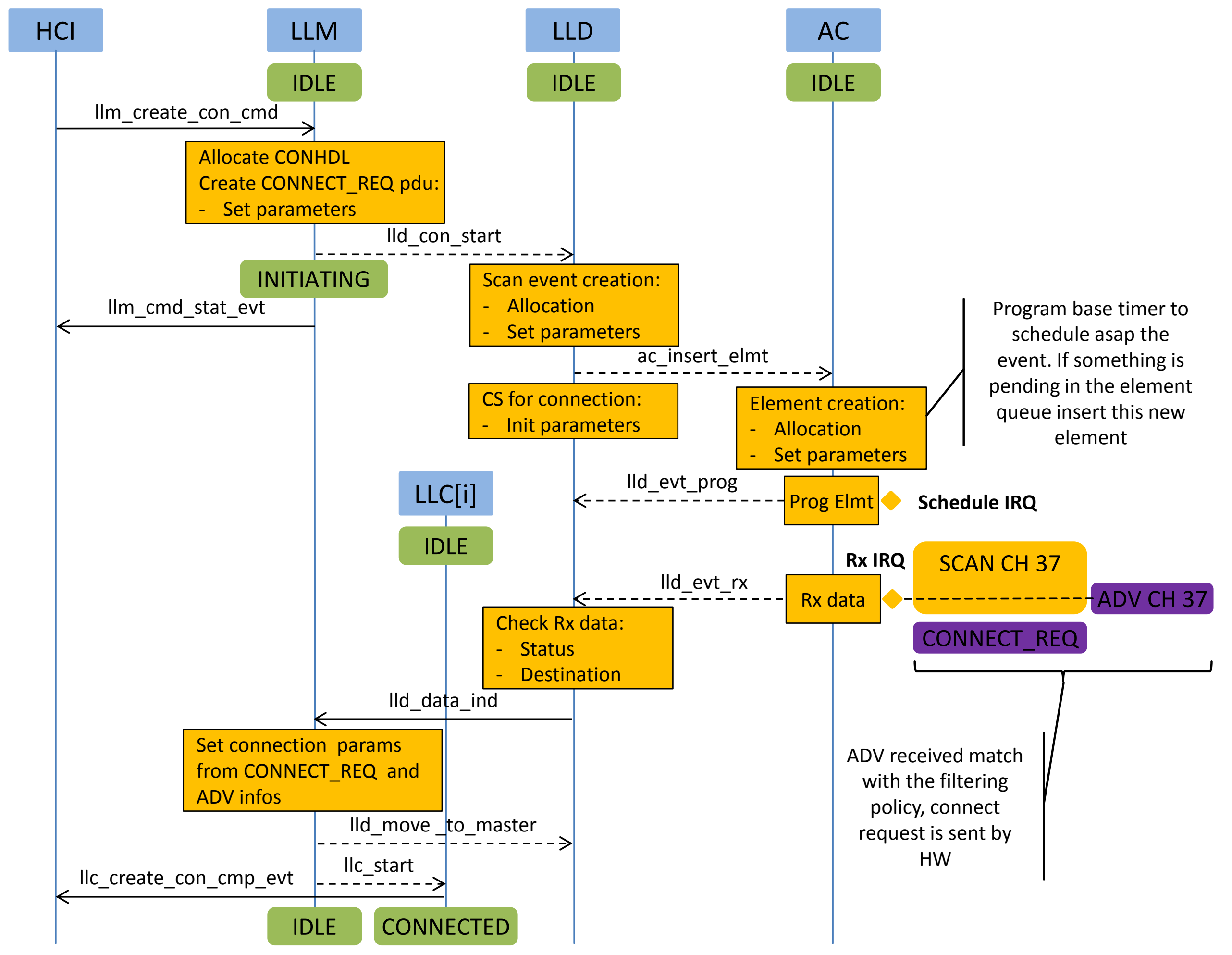


Figure 8-30 - Initiating MSC Initiator side

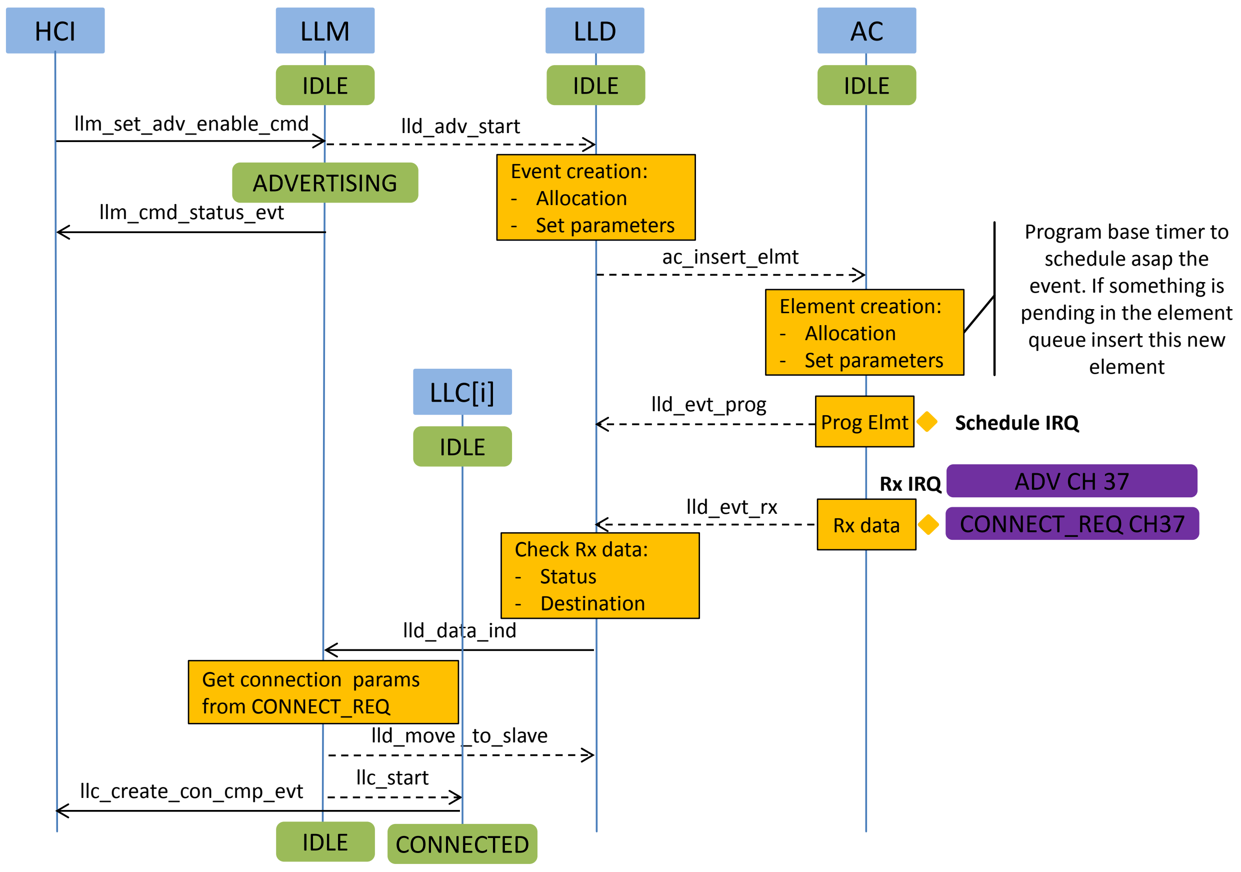


Figure 8-31 - Initiating MSC Responder side

8.6 Connection state

The Link Layer in the connected state transmits only Data Channel PDUs during connection events.

The timing of connection events is determined by three parameters:

• **Connection event interval**	: multiple of 1.25 ms in the range of 7.5ms to 4.0s.
• **Slave latency**	: sets by the initiator's LL LE from the range given by the Host.
• **Connect event length min and max**	: determine the duration of an event.

The connection interval allows setting the periodicity of the event.

The latency allows the slave to skip connection interval(s) to save power if nothing has to be sent.

The Connect event length allows allocating a bandwidth for an event.

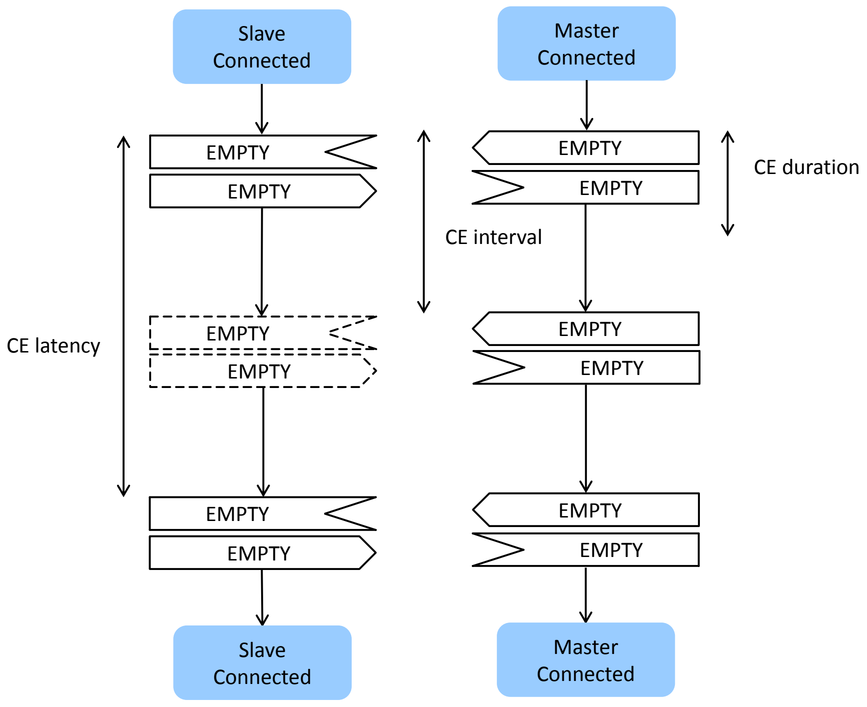


Figure 8-32 – Connected state overview

Several procedures are supported by the LL:

- Send data
- Connection Update
- Channel Map Update
- Features Exchange
- Version Exchange
- Start Encryption
- Restart Encryption
- Connection Parameters Update
- LE ping
- Disconnection

8.6.1 Send Data MSC

Device sends data to a peer:

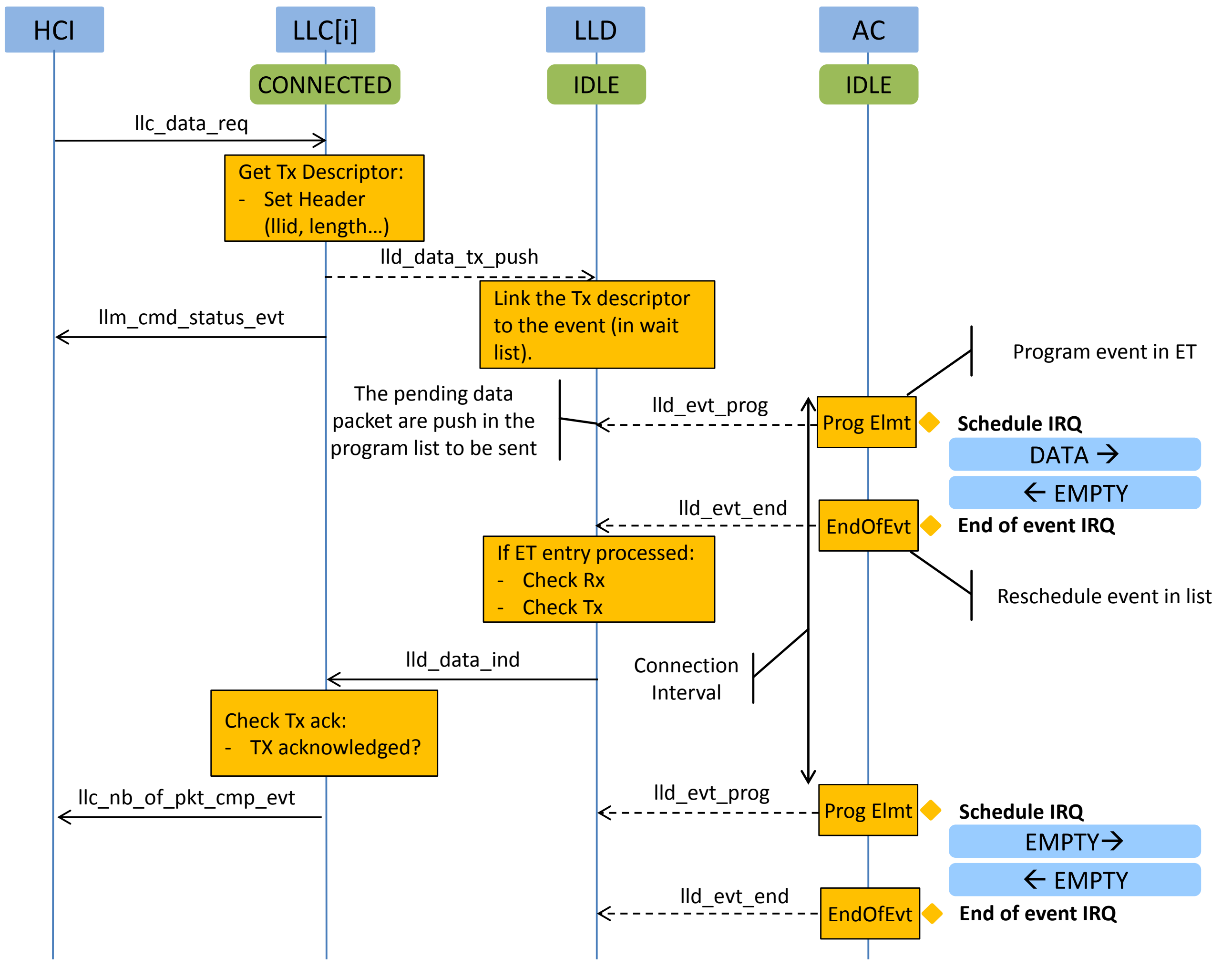


Figure 8-33 – Data sender MSC

Device receives data from a peer:

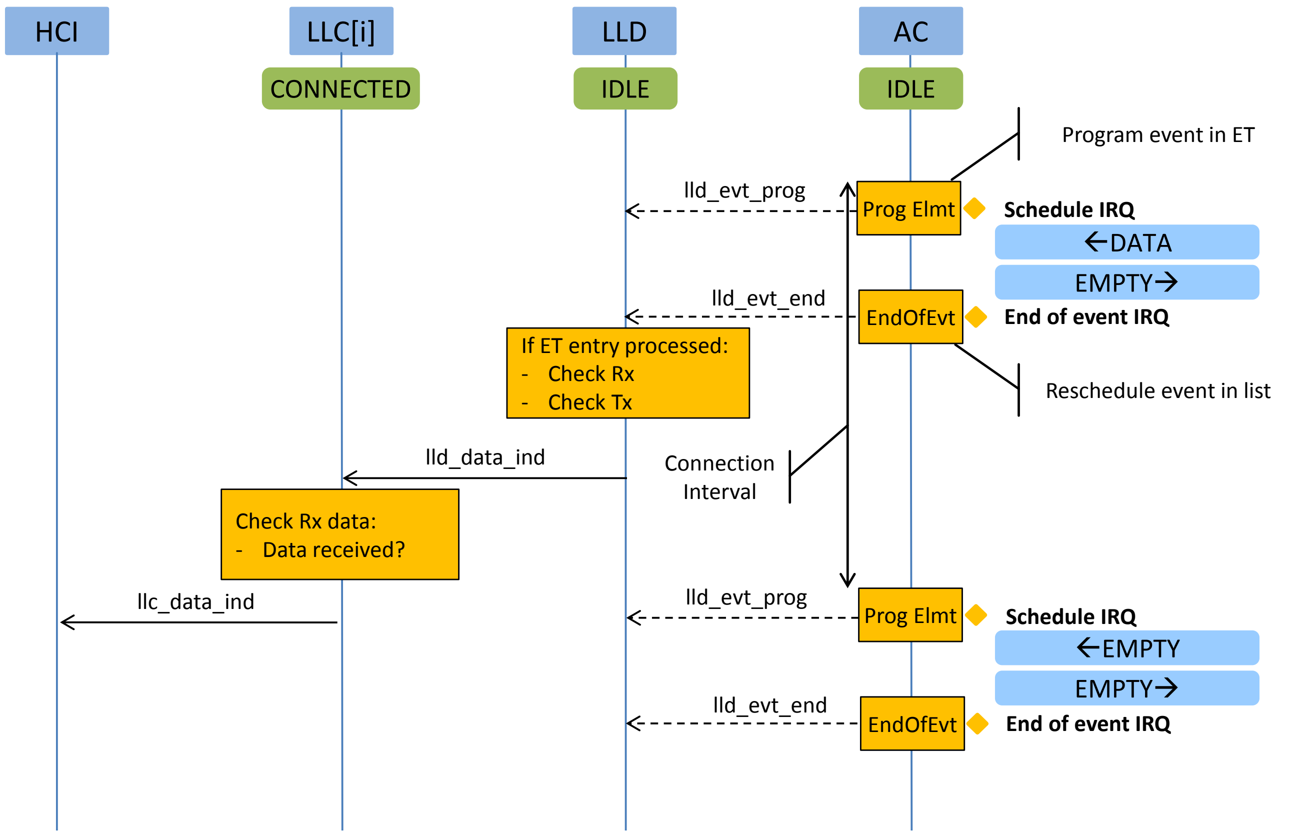


Figure 8-34 – Data receiver MSC

8.6.2 Connection Update Parameters MSC

A device (Master or Slave) can request a link parameters update to (see [1] Chapter Volume 6 Part B 5.1.1):

- Change the interval.
- Change the latency.
- Change the LSTO.

When a Master initiates the update parameters, the following procedure is executed:

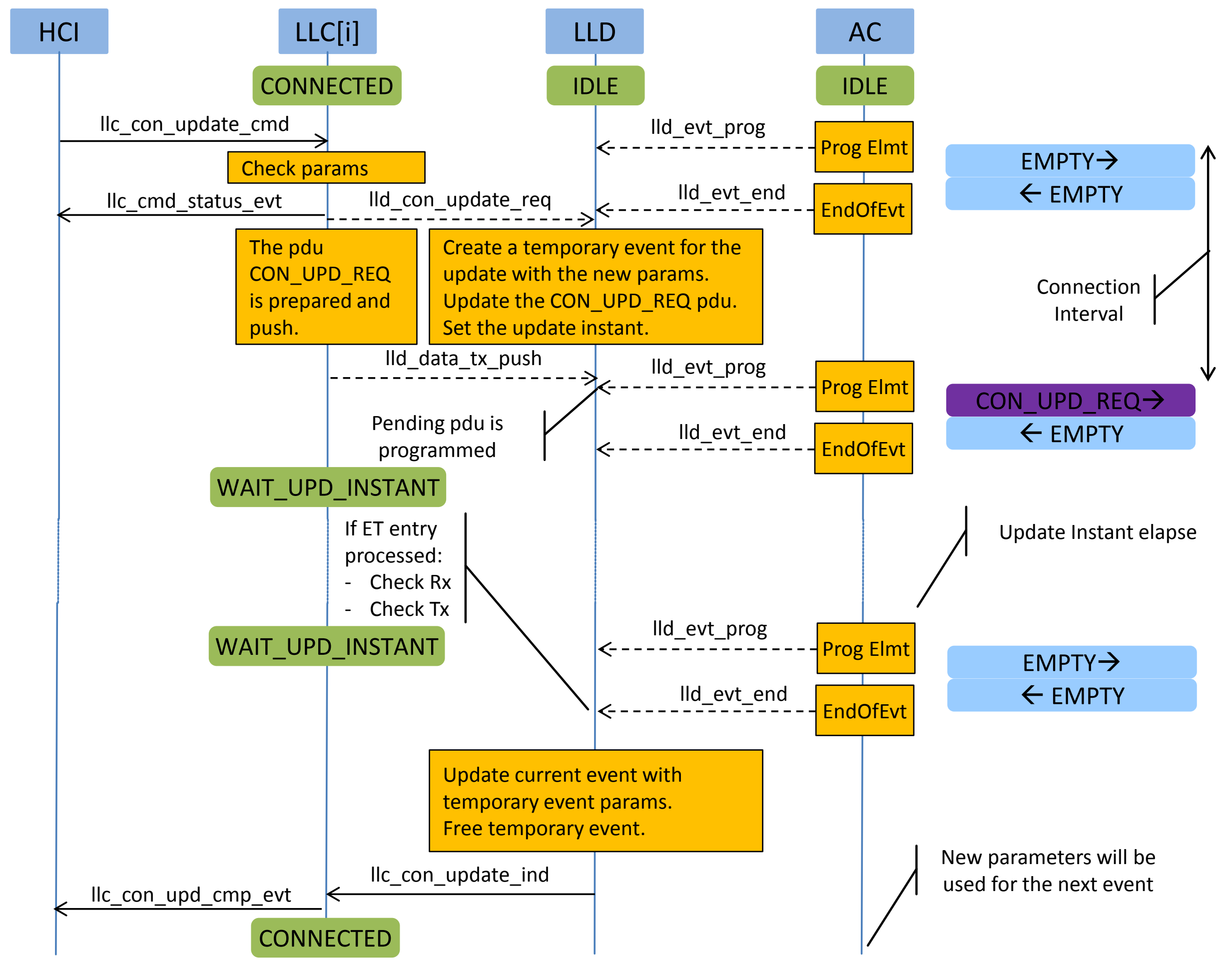


Figure 8-35 – Parameters Update Master initiator MSC

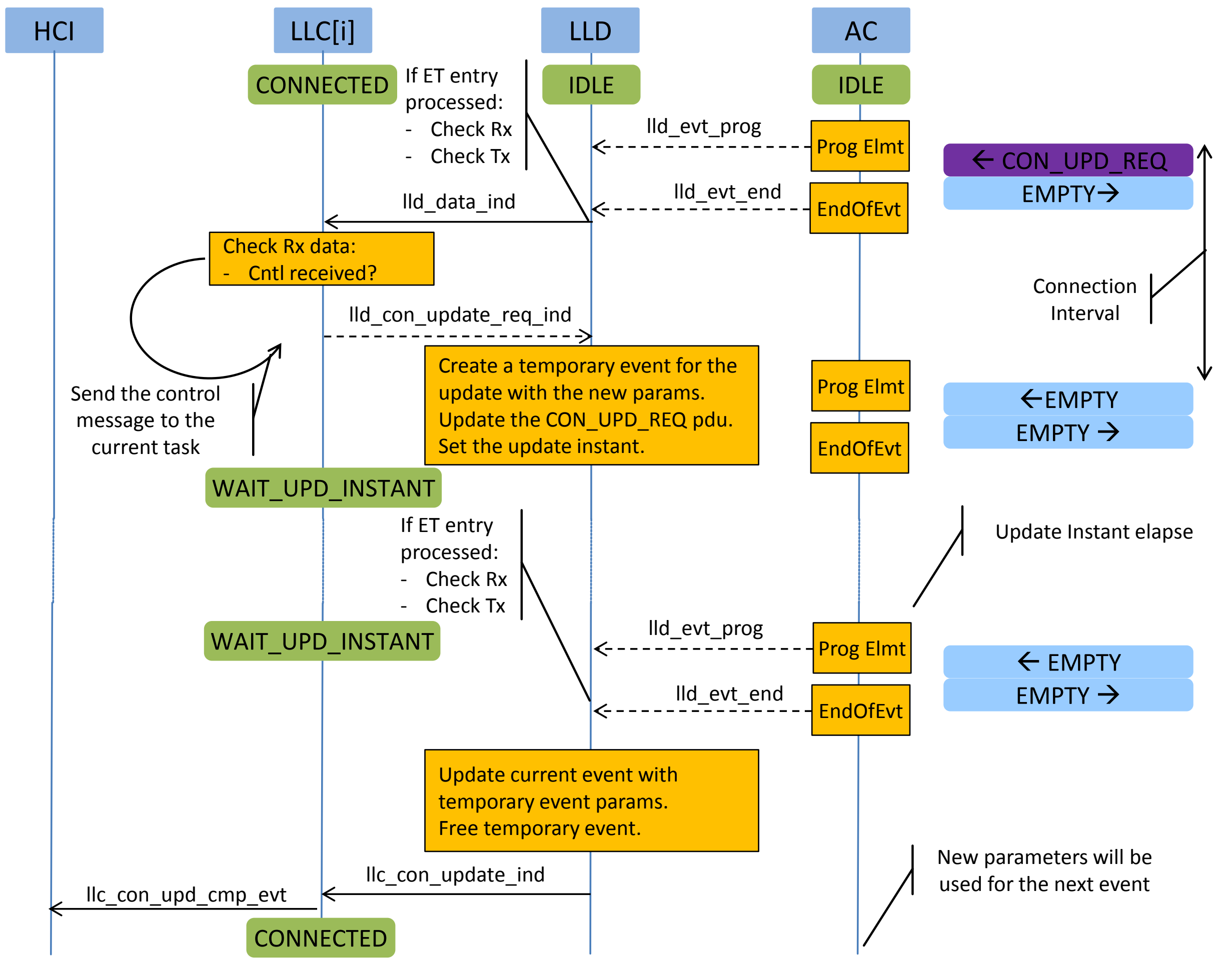


Figure 8-36 – Parameters Update Slave responder MSC

When the slave requests a parameters update, it uses the L2CAP LE signaling channel, this is managed by the upper layers.

8.6.3 Channel Map Update MSC

The Master can request to change the channel map by link: the procedure is describes below (see [1] Volume 6 Part B Chapter 5.1.2):

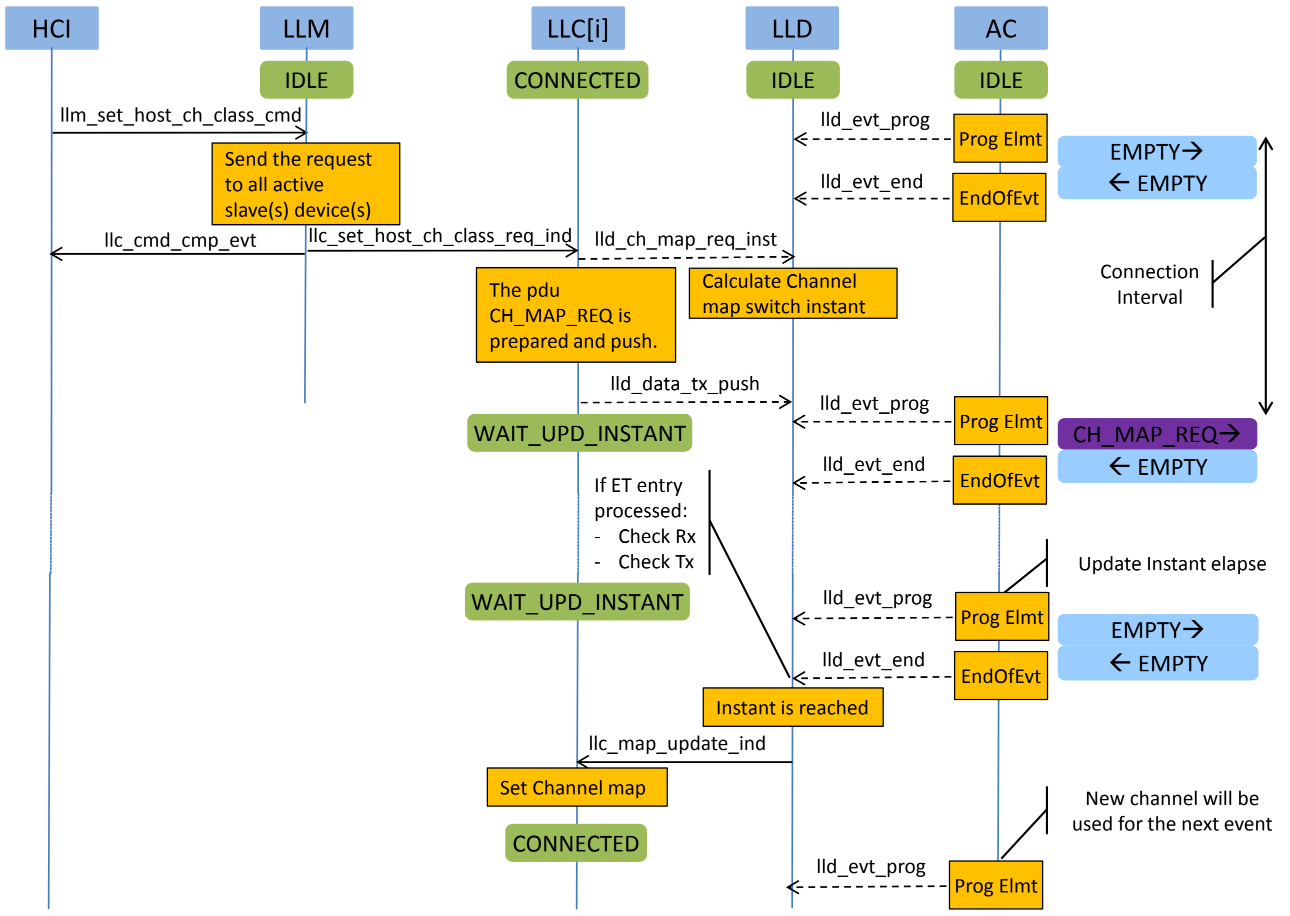


Figure 8-37 – Channel Map Update Initiator MSC

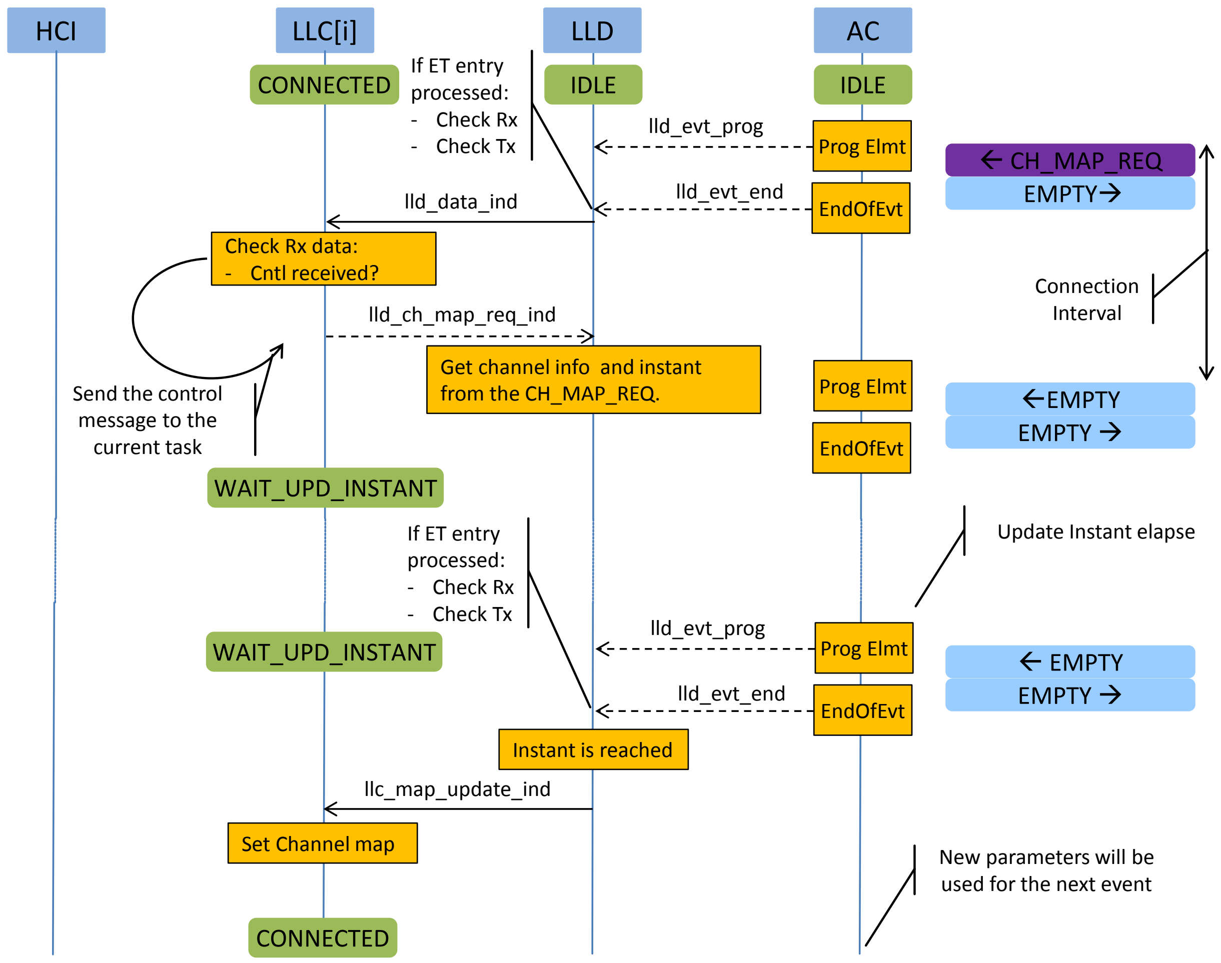


Figure 8-38 – Channel Map Update responder MSC

8.6.3.1 Assessment, Reassessment and Channel Map Updated FlowCharts

The LLM maintains a timer to check if a new channel map is applied.

The decision to start (or not) an assessment timer is done on Master side on each connection confirmation (only the Master can request to change the channel map).

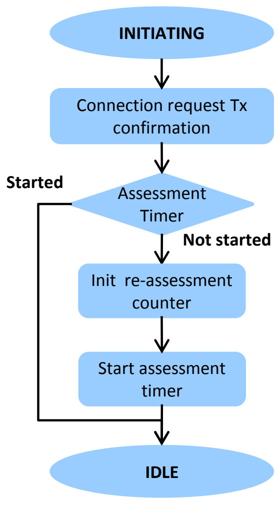


Figure 8-39 - Link Layer Manager: Start assessment timer

When the number of connected links is null, the timer is stopped.

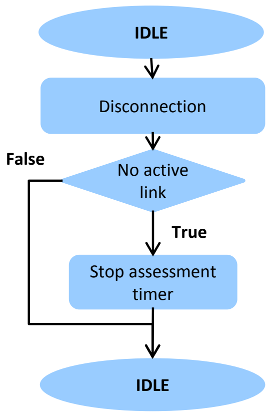


Figure 8-40 - Link Layer Manager: Stop assessment timer

On assessment timer expiration, the LLM checks if a reassessment is required (channel re allocation) and if the channel map of the Slaves devices has to be changed, a new channel map is computed and all the Slaves are informed with the channel map update procedure.

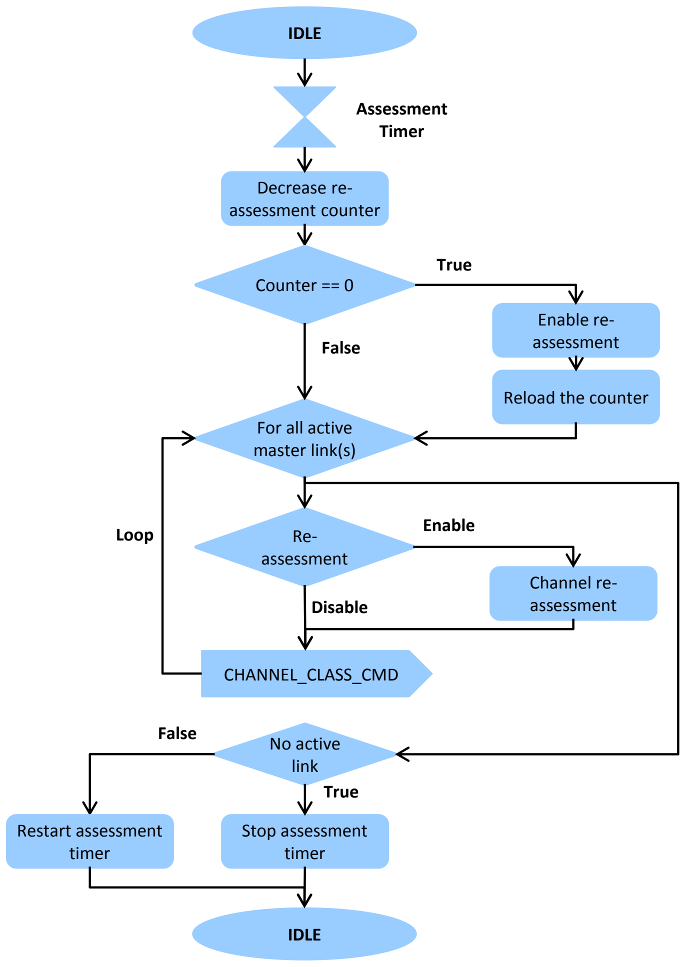


Figure 8-41 - Link Layer Manager: Assessment timer management

The LLC receives from the upper layer a channel map update request: it is in charge to determine the instant of channel map update (with a minimum of 6 connection events before the instant occurs).

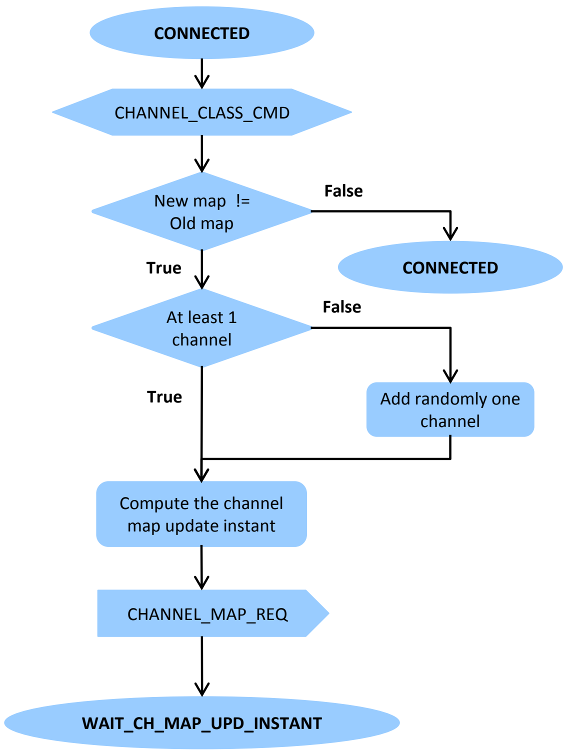


Figure 8-42 – Link Layer Controller: Assessment or channel map update request

The LLC is also in charge of the channel classification.

The classification is done on the connected channels only by a SW algorithm, but not on the advertising channels.

If the index value is “1” then the channel is considered as “good” and the Core uses it. If the index value is “0” then the channel is considered as “bad” and the Core does not use it.

The decision to set the channel index to “good” or “bad” comes from the local assessment, using the RSSI of the packet received, the error and also, if it is available, local information coming from the host (e.g BR/EDR channel even channel map part).

The SW is in charge to maintain the CS fields used for the channel management:

- Mapping of used and unused channels.
- Number of used channels.
- Number of bad packet received.

8.6.4 Features Exchange MSC

Master and Slave can request to check the features supported by the peer (see [1] Volume 6 Part B Chapter 5.1.4):

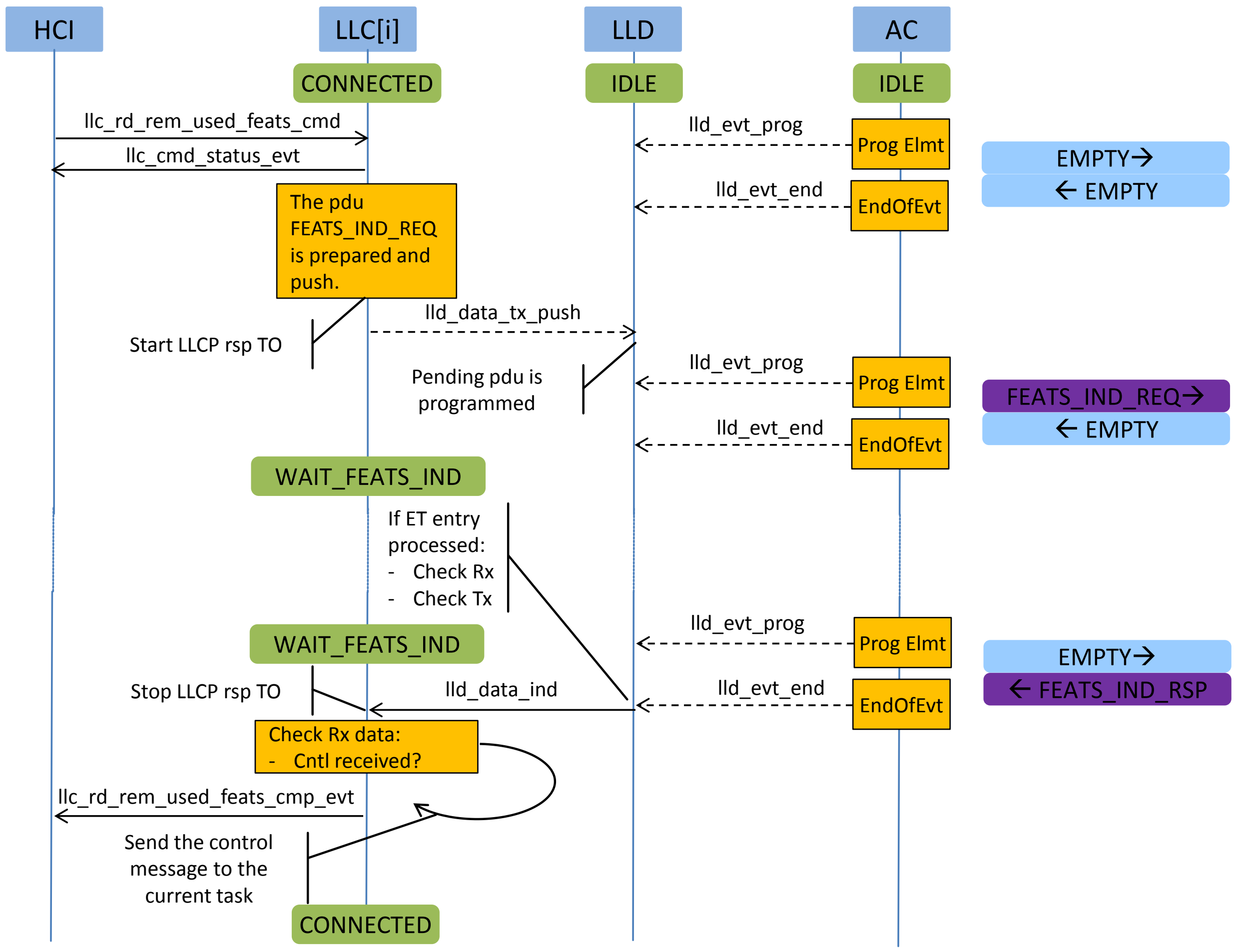


Figure 8-43 – Features Request Master Initiator MSC

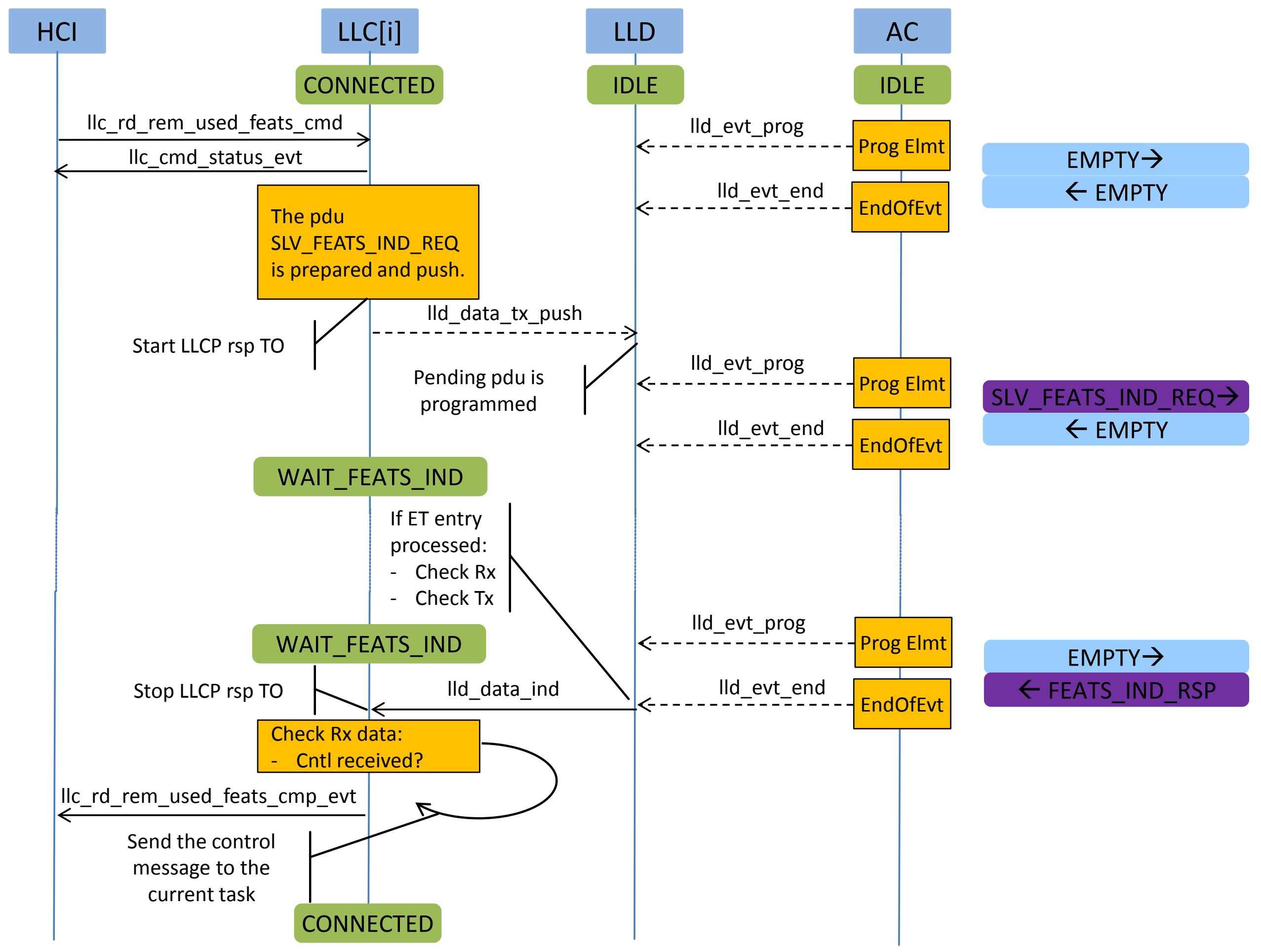


Figure 8-44 – Features Request Slave Initiator MSC

8.6.5 Version Exchange MSC

This procedure can be executed by Master or Slave device (see [1] Volume 6 Part B Chapter 5.1.5):

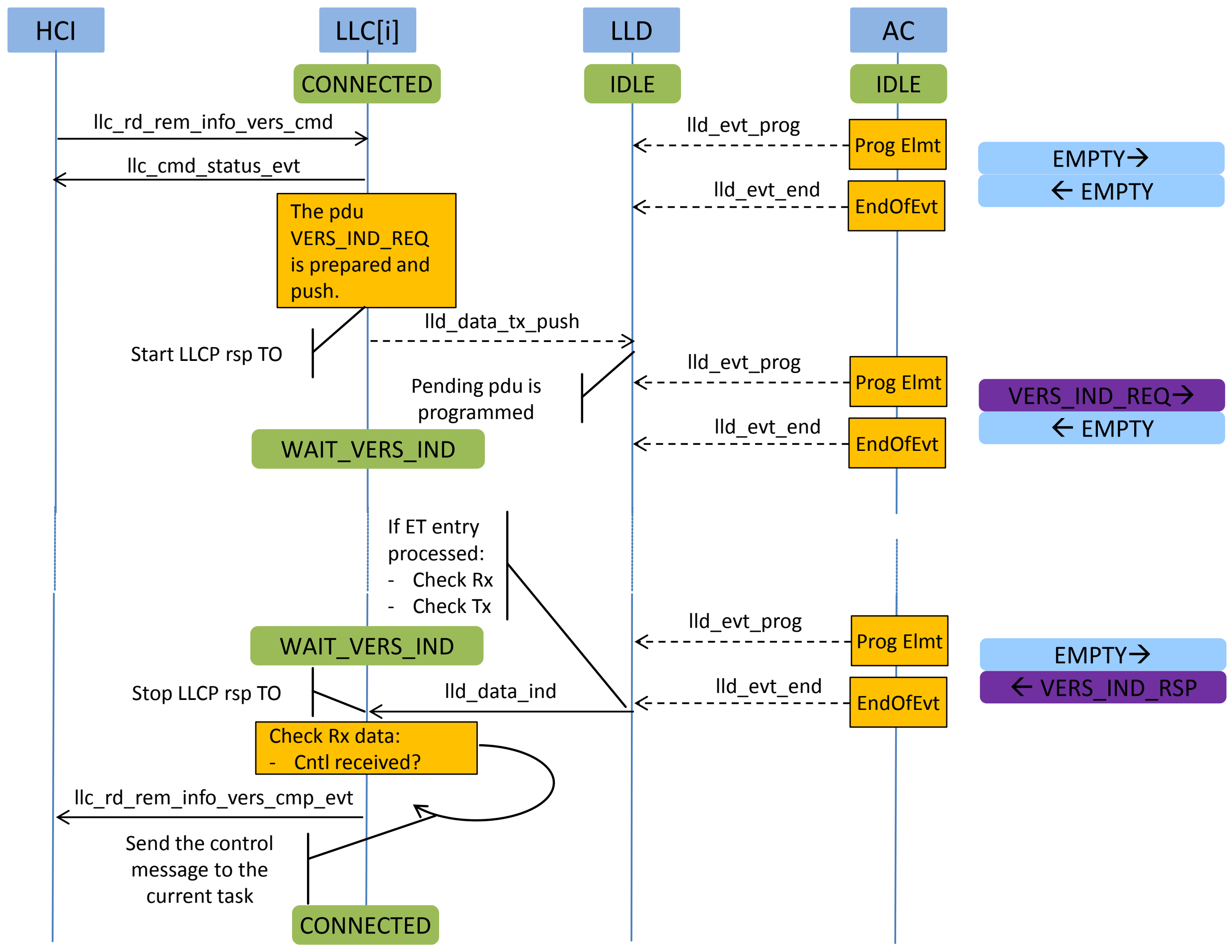


Figure 8-45 – Version Request MSC

8.6.6 Start Encryption MSC

Only the Master initiates an encryption procedure. The slave can request to have a secure link over L2CAP channel.

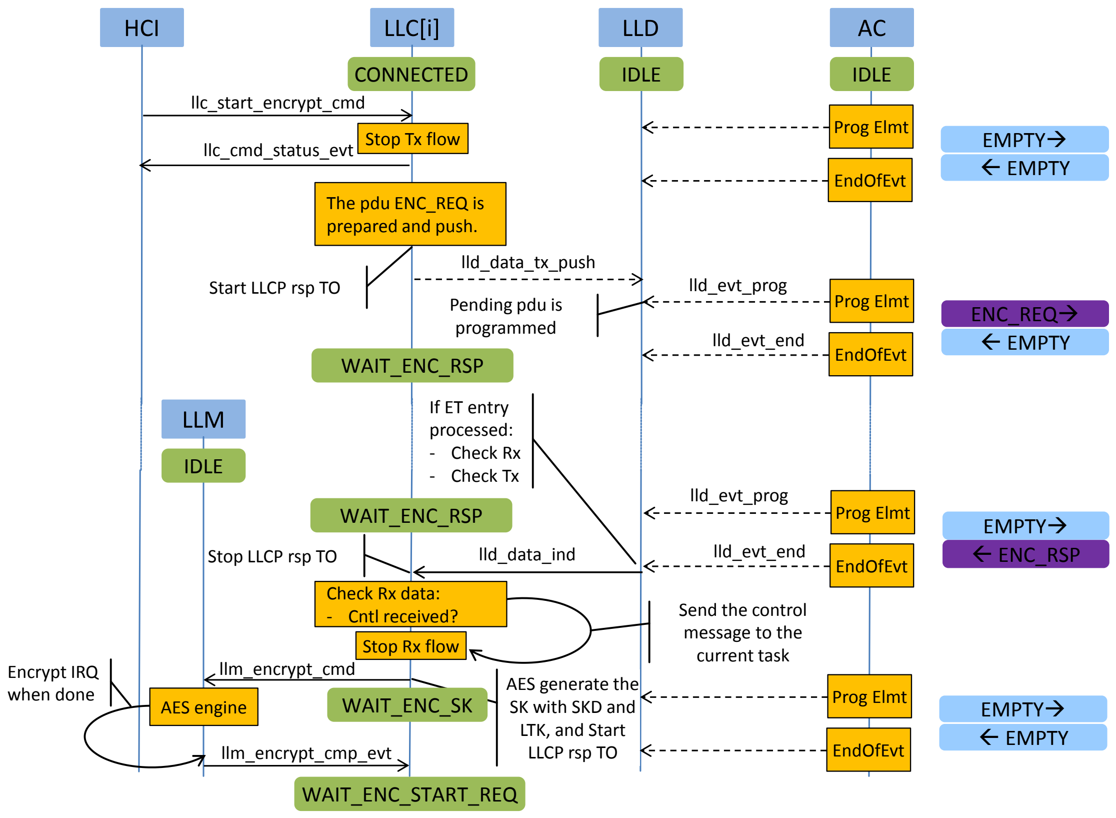


Figure 8-46 – Master Encryption phase 1 MSC

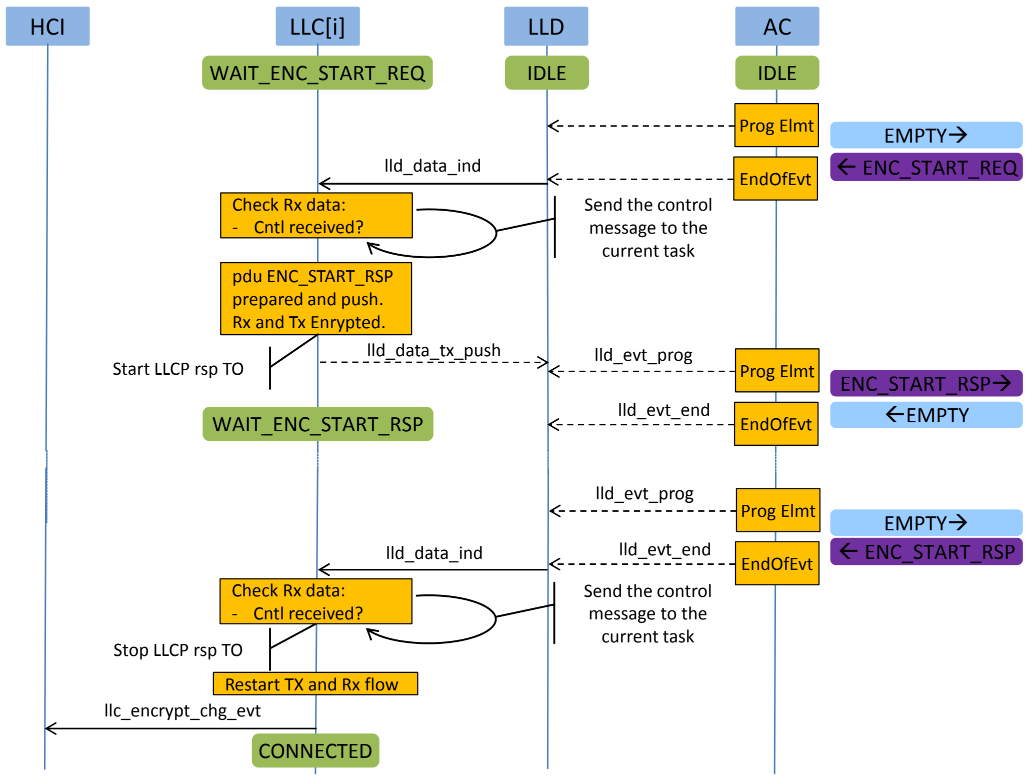


Figure 8-47 – Master Encryption phase 2 MSC

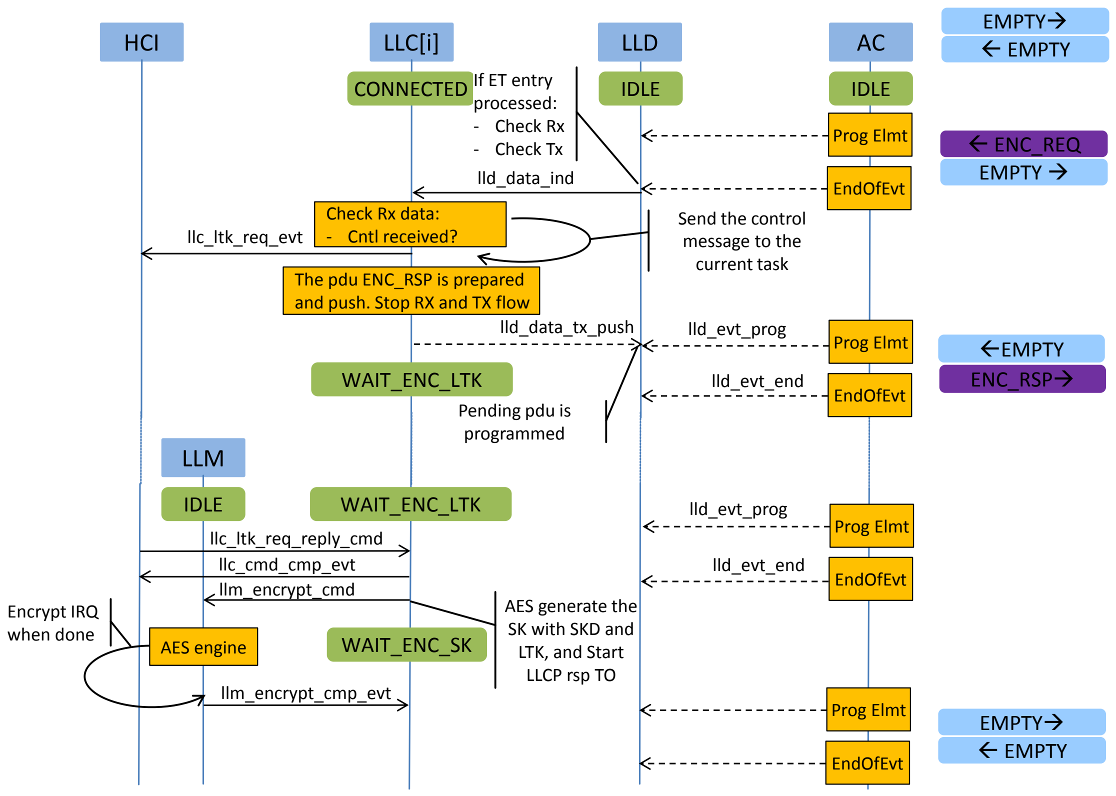


Figure 8-48 – Slave Encryption phase 1 MSC

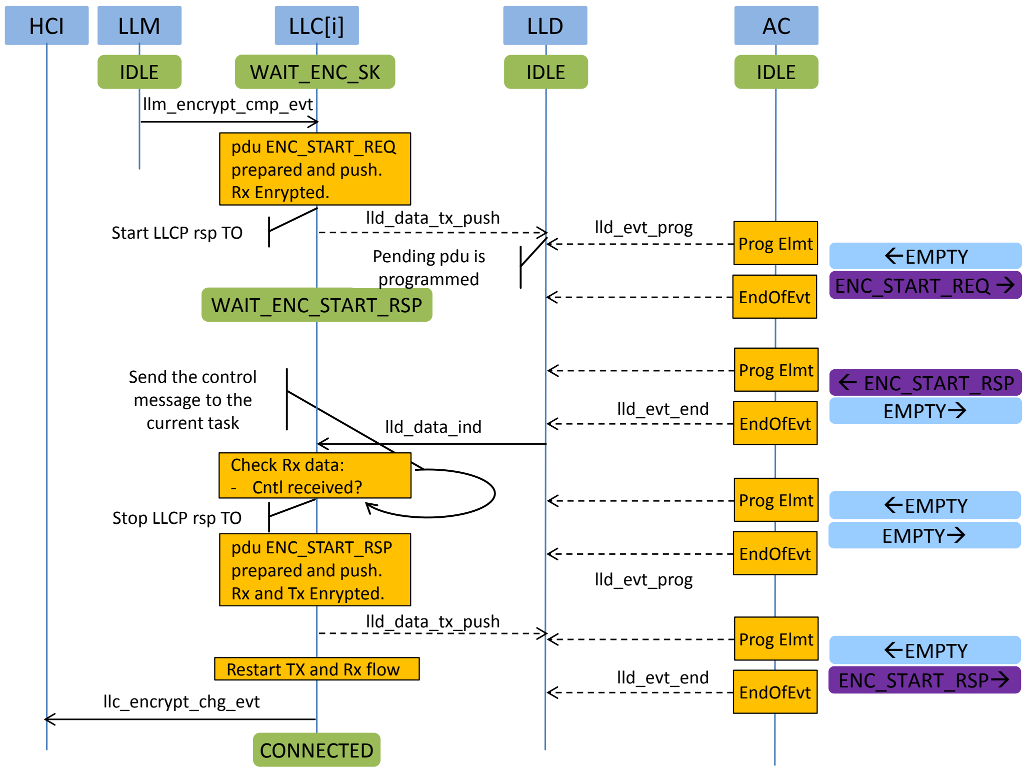


Figure 8-49 – Slave Encryption phase 2 MSC

8.6.7 Restart Encryption MSC

This procedure can be executed by Master only (see [1] Volume 6 Part B Chapter 5.1.3.2):

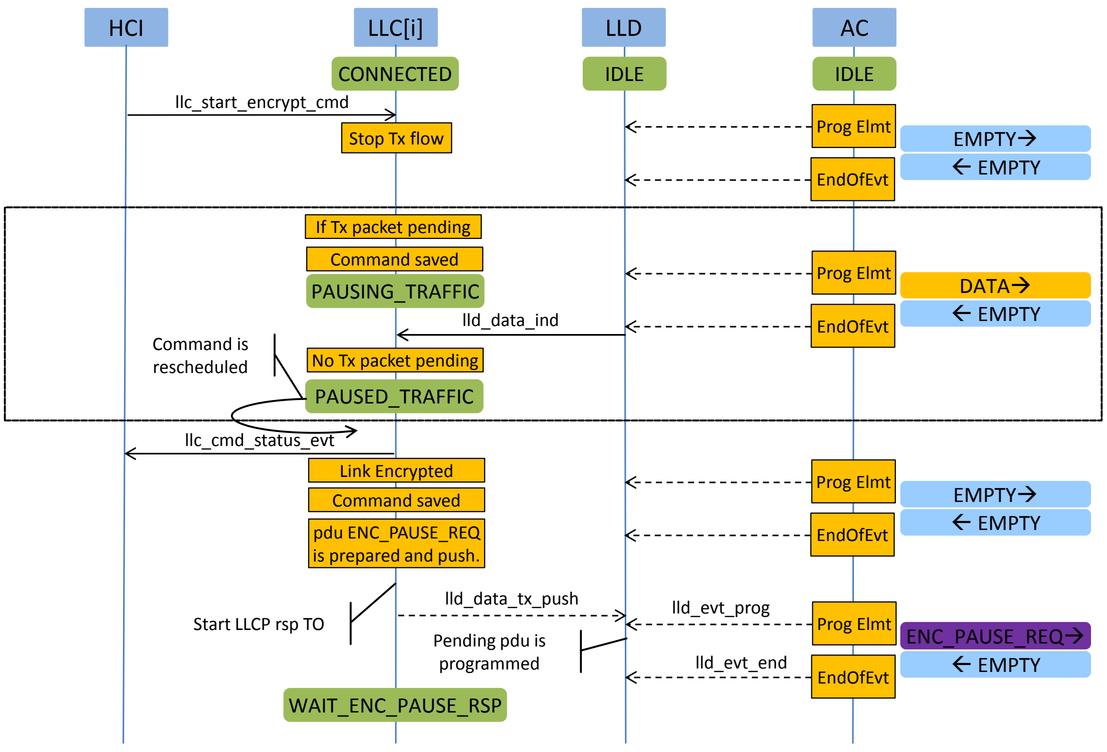


Figure 8-50 – Master Refresh key phase 1 MSC

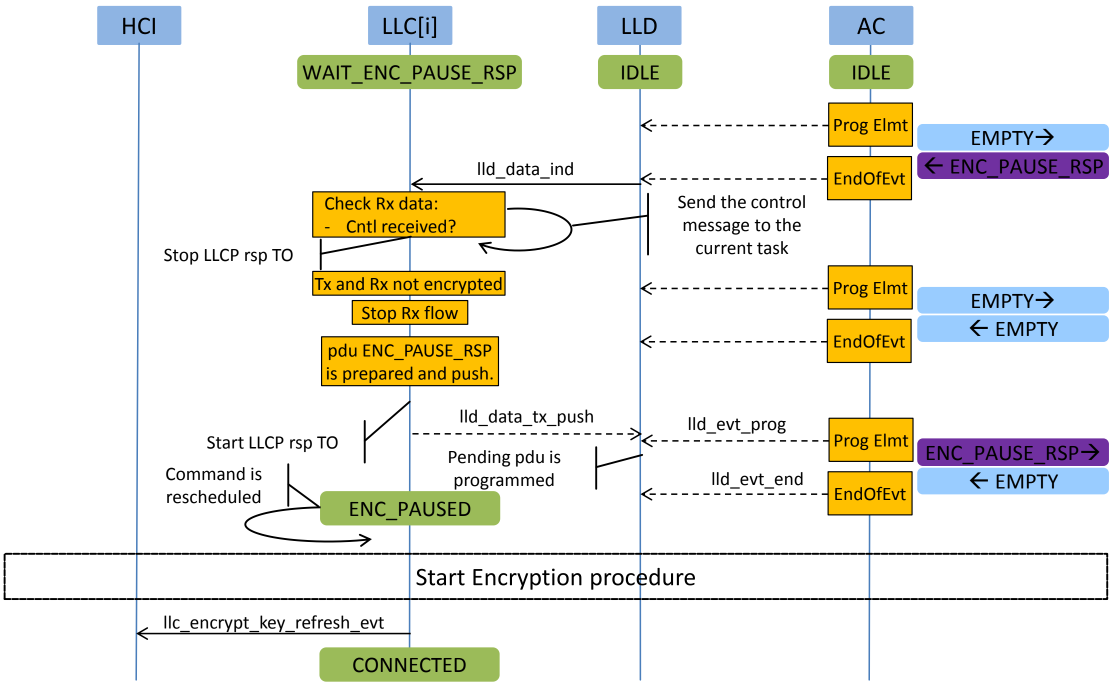


Figure 8-51 – Master Refresh key phase 2 MSC

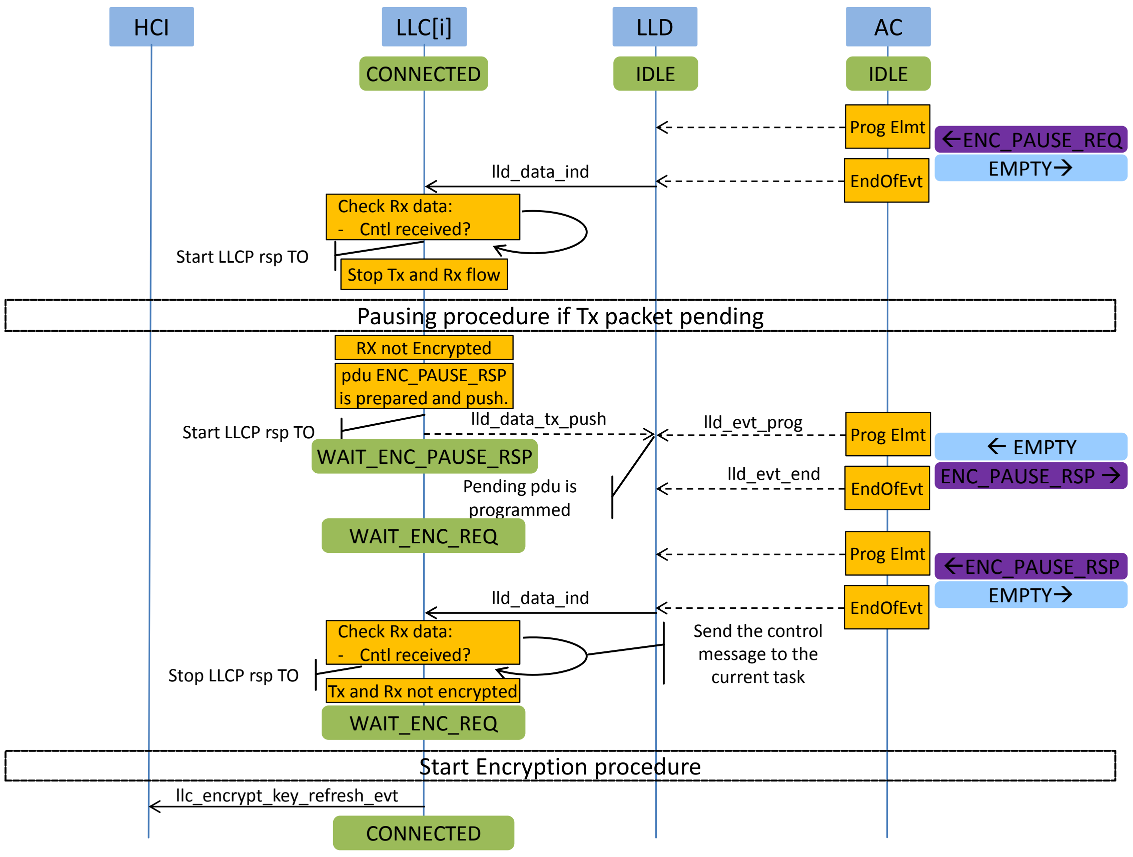


Figure 8-52 – Master Refresh key phase 1 and 2 MSC

8.6.8 Connection Parameters Update

The purpose of this procedure is to be able to move the anchor point or change connection parameters. At the end of this procedure, the parameters update procedure is launched by the master.

8.6.8.1 Connection Parameters Update MSC

The master or slave initiates a Connection Parameters Request procedure at any time after entering the Connection State. (See [1] Volume 6 Part B Chapter 5.1.7).
The procedure can be initiated:

1. By command if the host updates: connection Interval, connection Slave Latency, connection Supervision Timeout or anchor point moving.

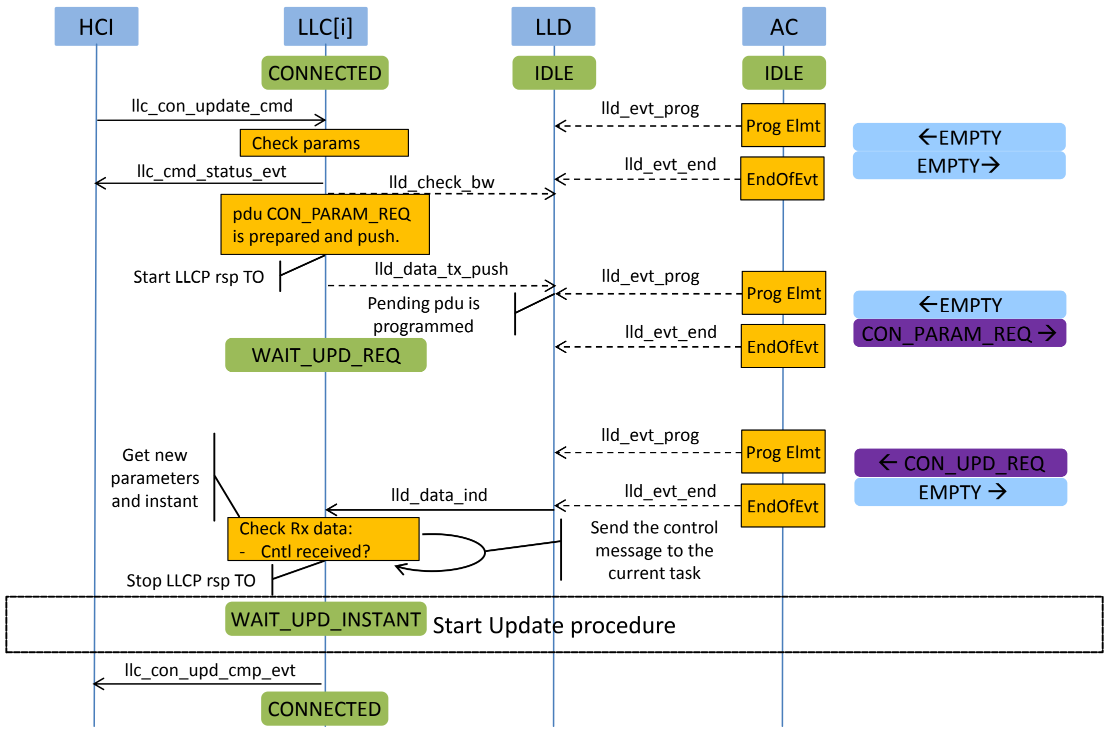


Figure 8-53 – Slave Host initiates parameters update MSC

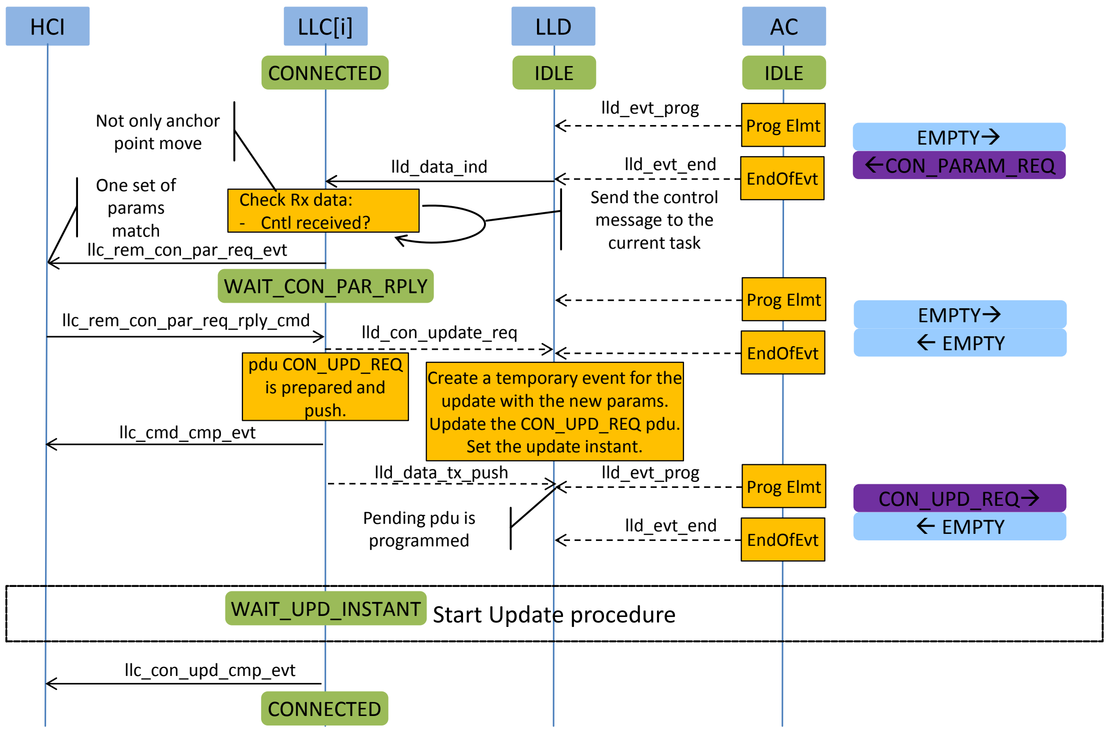


Figure 8-54 – Master Host responds parameters update MSC

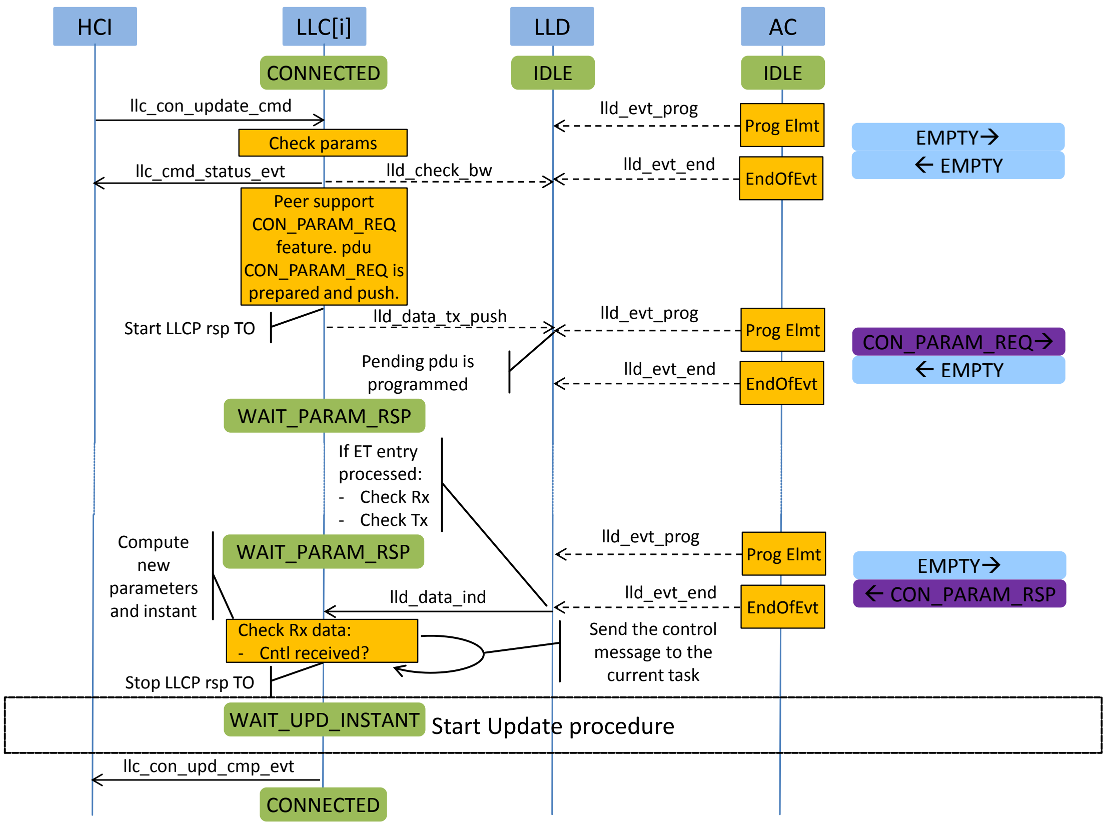


Figure 8-55 – Master Host initiates parameters update MSC

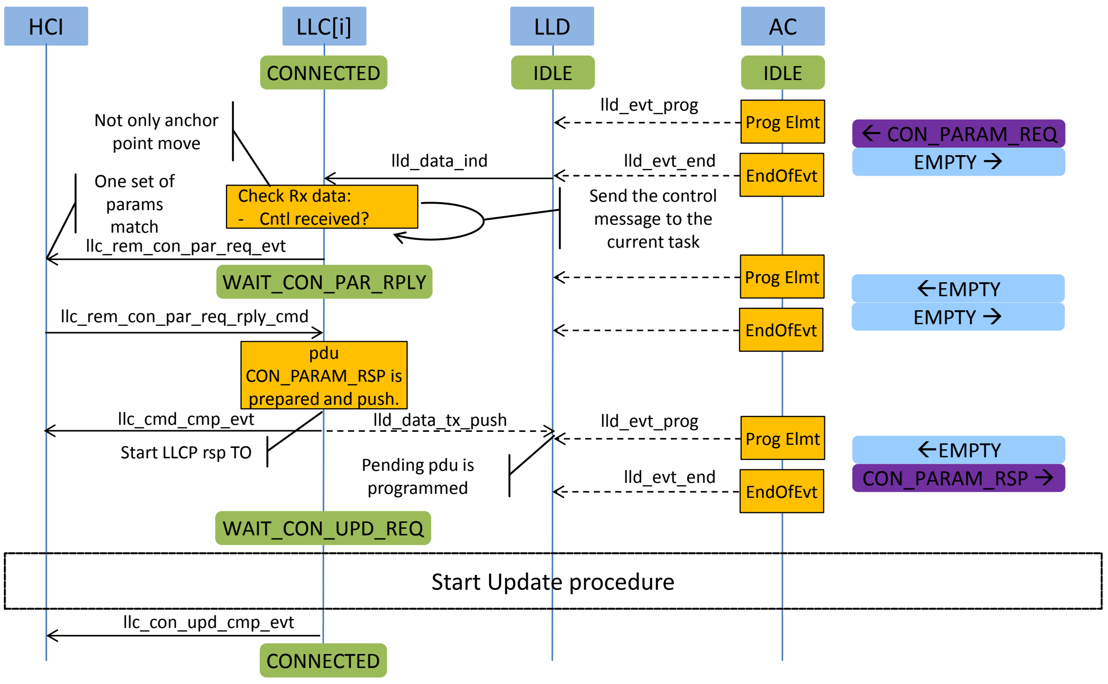


Figure 8-56 – Slave Host responds parameters update MSC

If the parameters received in the CON_PARAM_REQ or CON_UPD_REQ are the same, than the current connection, a move anchor point is requested and the host doesn't need to be informed, the link layer will automatically perform the procedure.

2. Autonomously by the Link Layer: move anchor point of the BLE connection.

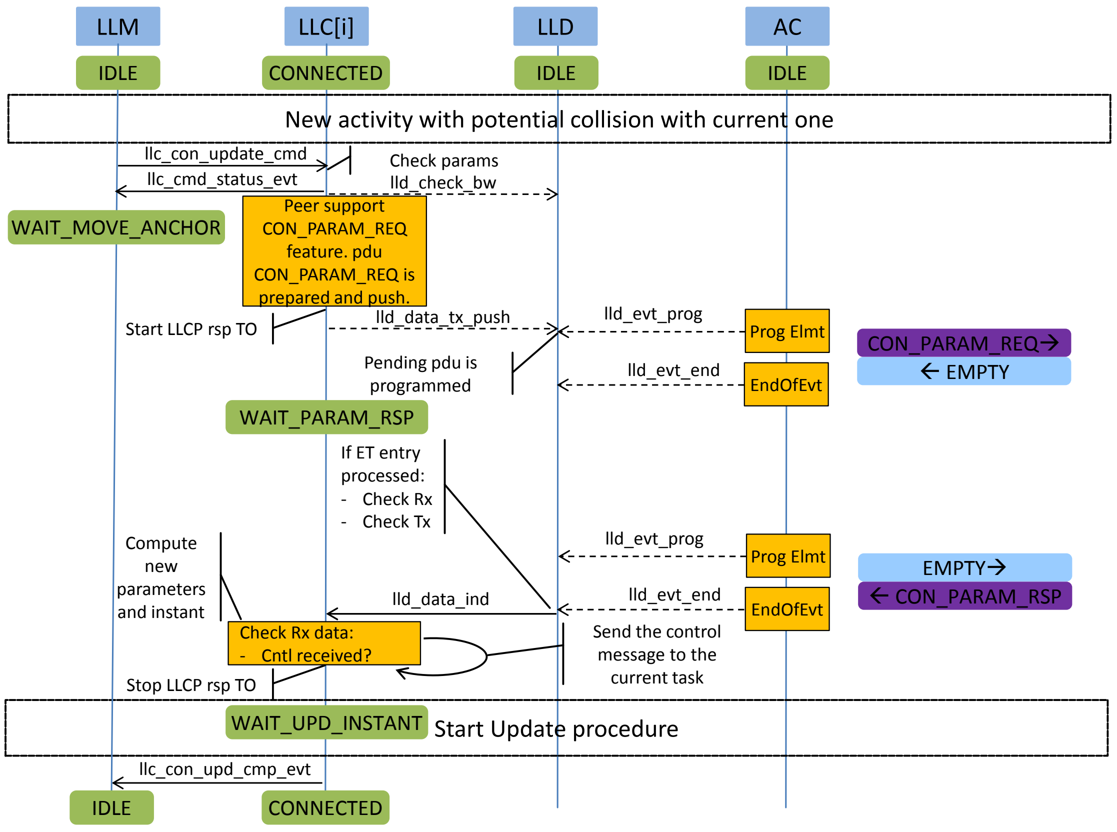


Figure 8-57 – Master LL initiates parameters update MSC

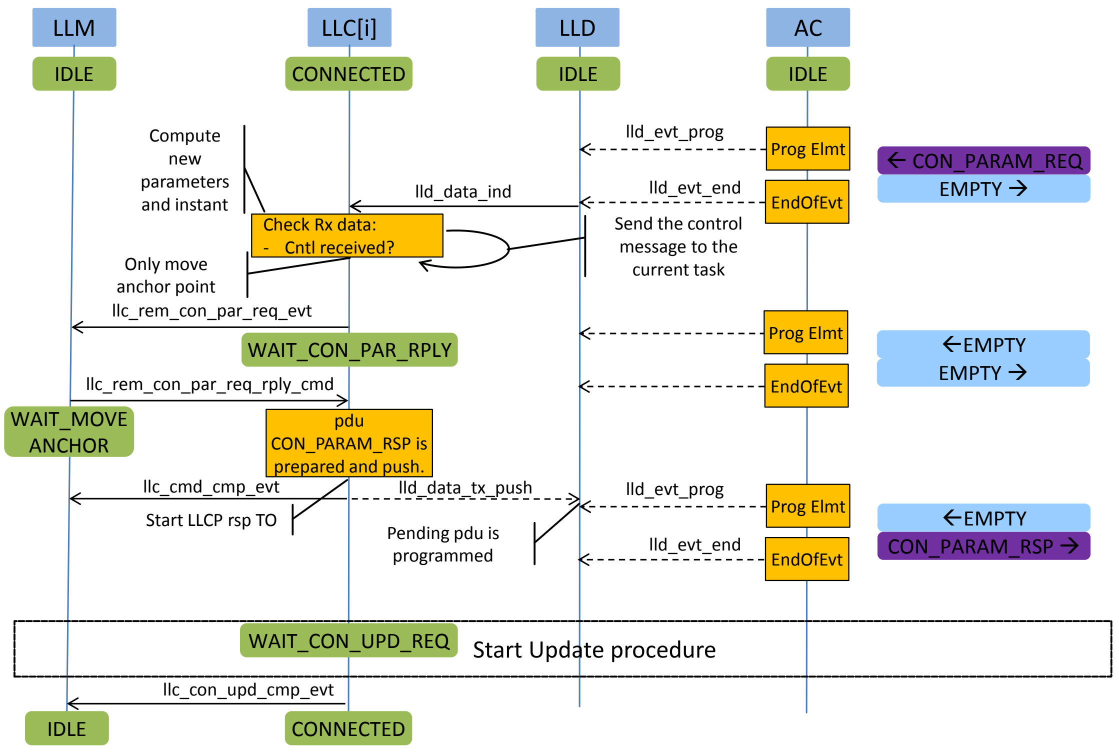


Figure 8-58 – Slave LL responds parameters update MSC

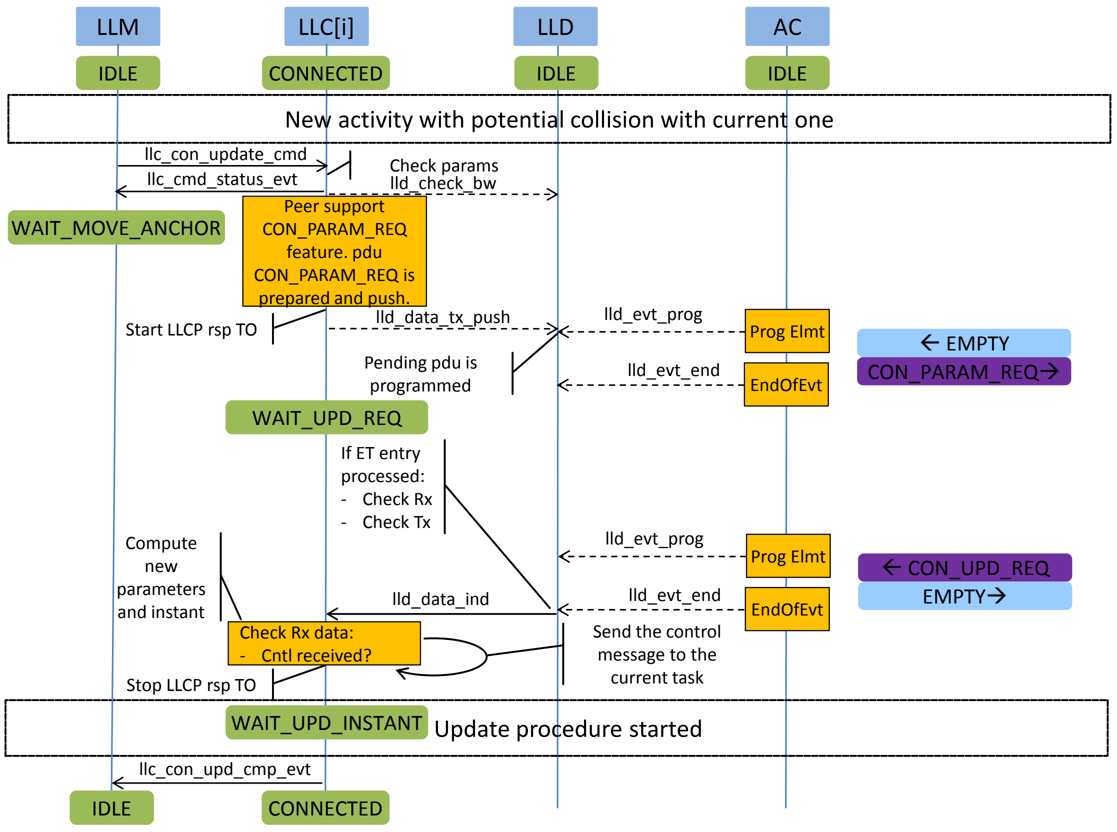


Figure 8-59 – Slave LL initiates parameters update MSC

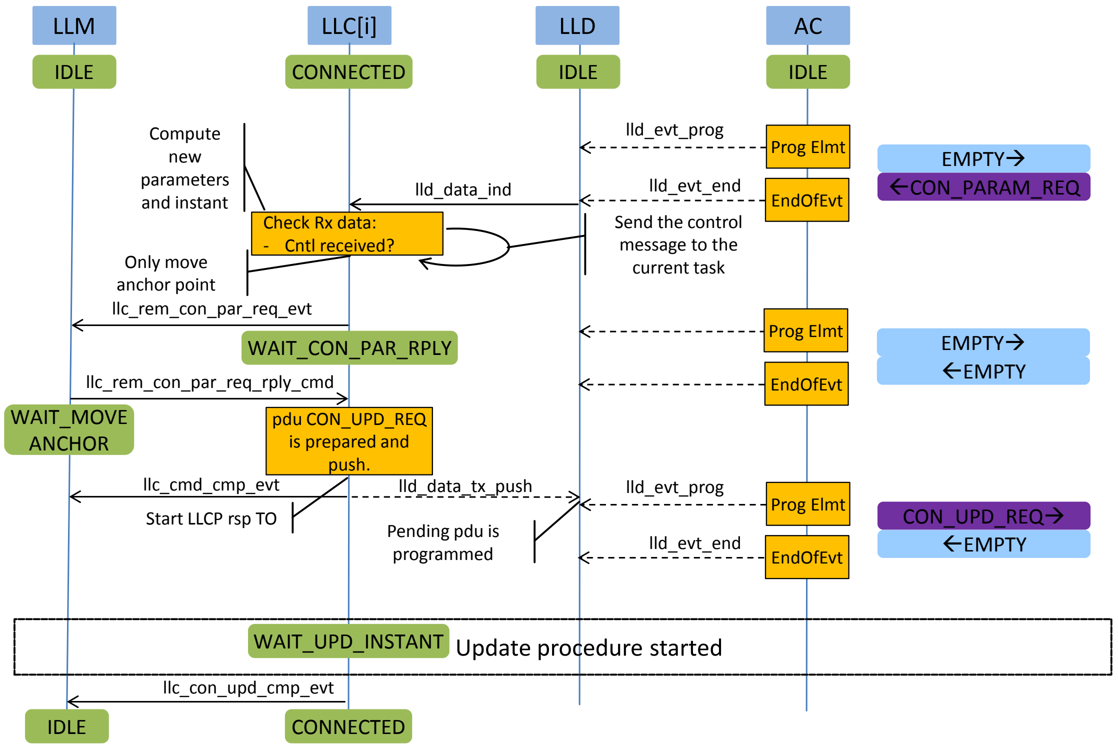


Figure 8-60 – Master LL Responds parameters update MSC

The master or slave can reject a Connection Parameters Request procedure if the parameters cannot match with the context of the current connection(s).

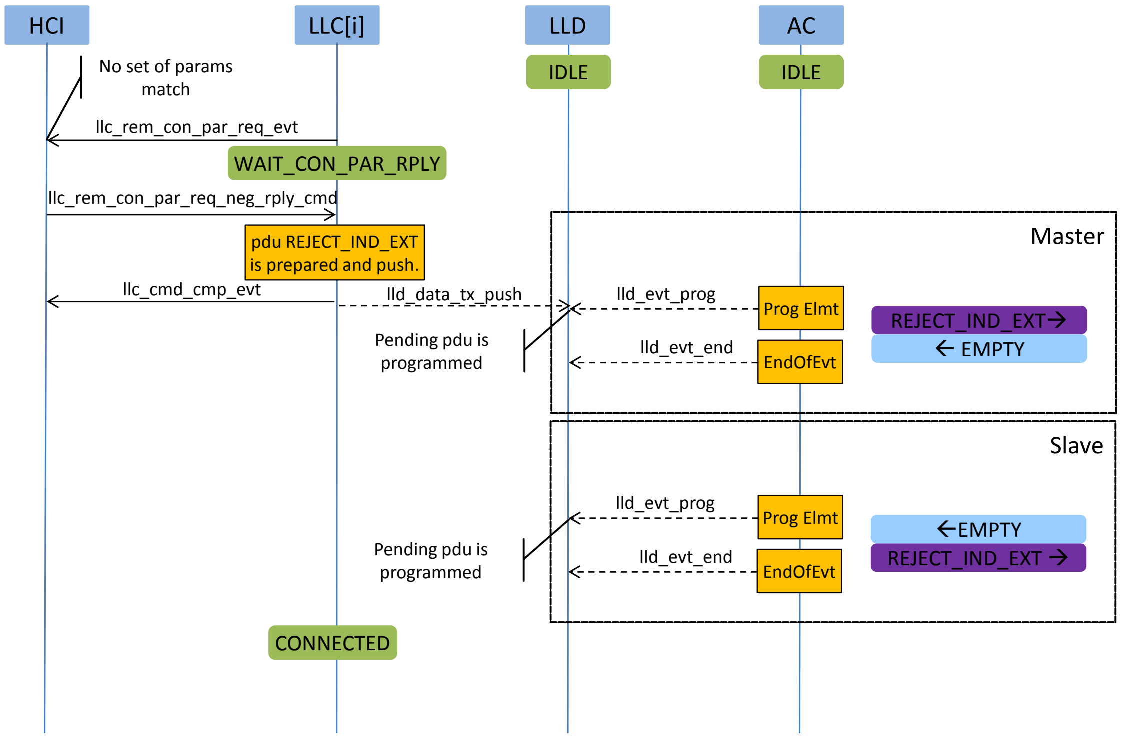


Figure 8-61 – Master/Slave Reject the parameters update MSC

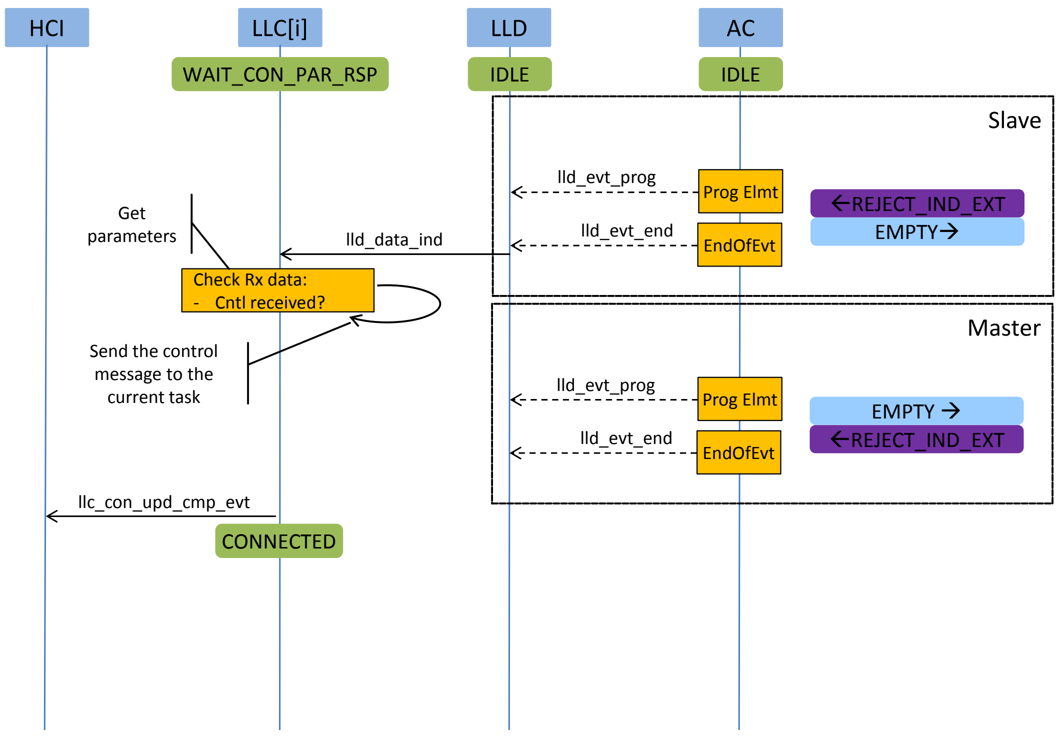


Figure 8-62 – Master/Slave Received the parameters update Reject MSC

If the procedure is started by the LL, the reject procedure is the same.

8.6.8.2 Connection Parameters Update FlowChart

1. **Host can request a parameter update; never mind the role.**

Only the LLC and the LLD will be impacted, as this procedure is link oriented. The LLC is in charge to check the parameters received by the command, and to prepare the PDU and to set the link states.
It is informed by the LLD of the real time activity (if the update instant is passed or if there is a TO).

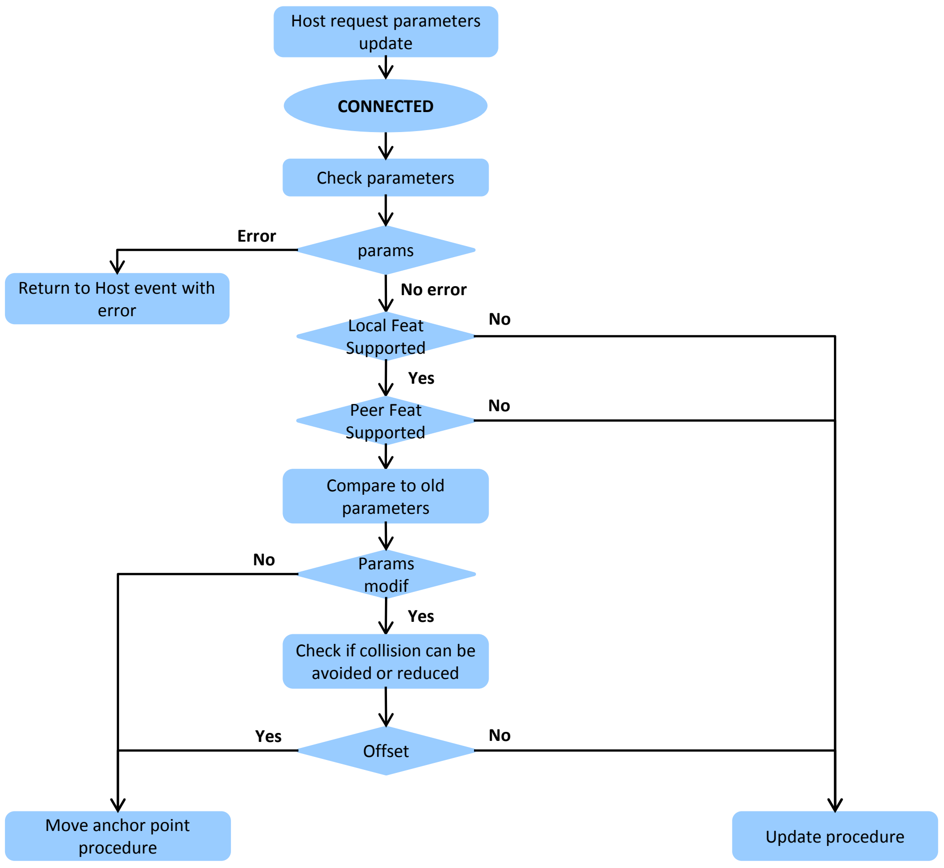

Host request parameters update
CONNECTED
Check parameters
Error
params
Return to Host event with error
No error
Local Feat Supported
No
Yes
Peer Feat Supported
No
Compare to old parameters
No
Params modif
Yes
Check if collision can be avoided or reduced
Yes
Offset
No
Move anchor point procedure
Update procedure

Figure 8-63 – Master/Slave LLC parameters update command management

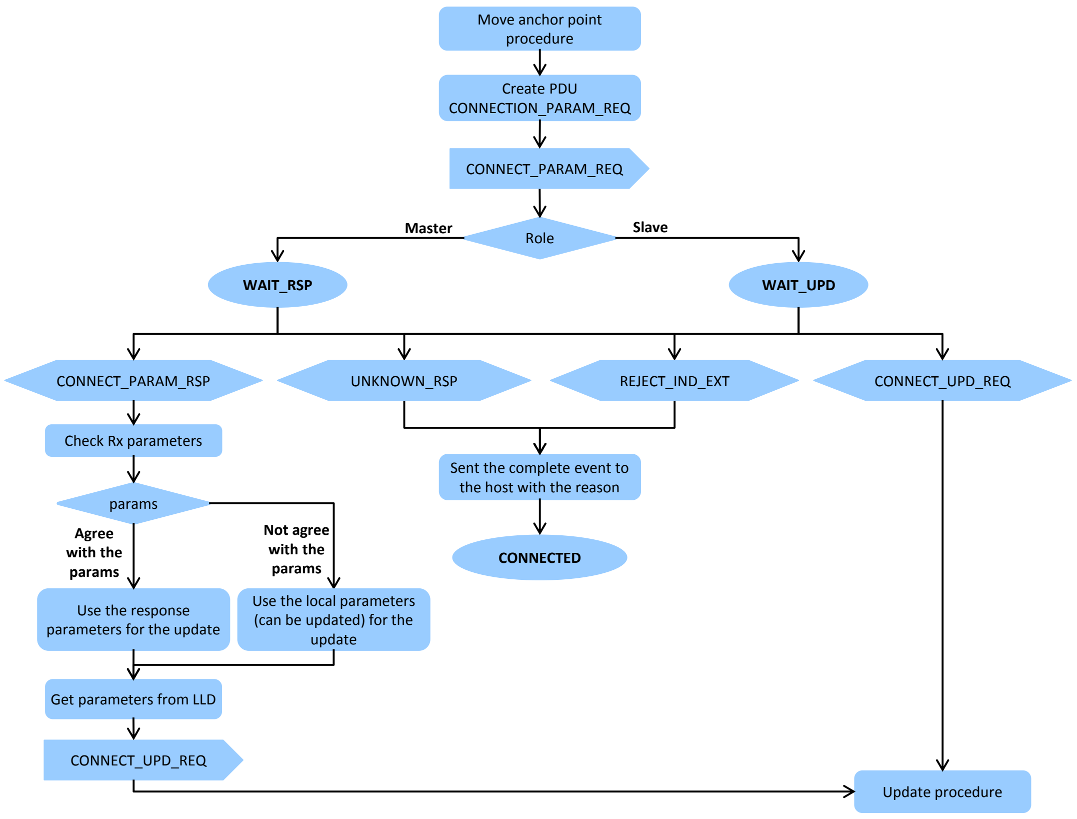

Move anchor point procedure
Create PDU CONNECTION_PARAM_REQ
CONNECT_PARAM_REQ
Role
Master
Slave
WAIT_RSP
WAIT_UPD
CONNECT_PARAM_RSP
UNKNOWN_RSP
REJECT_IND_EXT
CONNECT_UPD_REQ
Check Rx parameters
params
Agree with the params
Not agree with the params
Use the response parameters for the update
Use the local parameters (can be updated) for the update
Sent the complete event to the host with the reason
CONNECTED
Get parameters from LLD
CONNECT_UPD_REQ
Update procedure

Figure 8-64 – Master/Slave LLC PDU CONNECT_PARAM_REQ management

The CONNECT_UPD_REQ pdu parameter window offset is set accordingly to the selected offset.

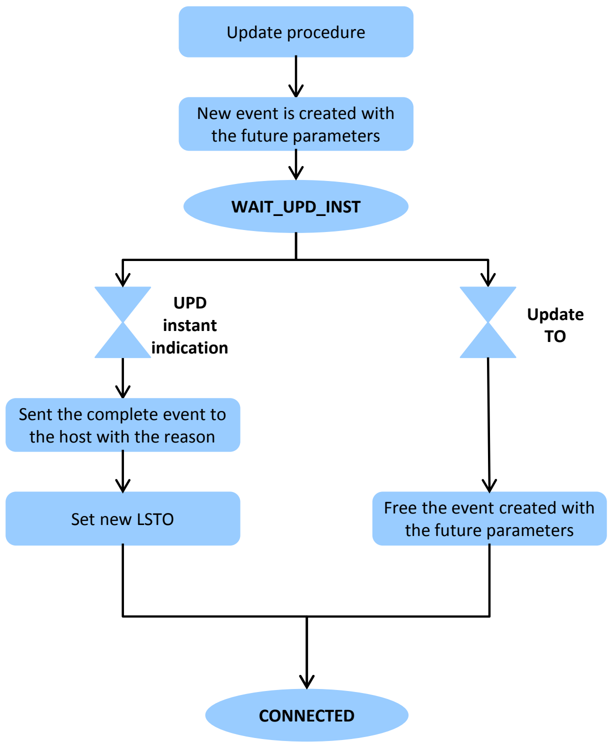

Update procedure
New event is created with the future parameters
WAIT_UPD_INST
UPD instant indication
Update TO
Sent the complete event to the host with the reason
Set new LSTO
Free the event created with the future parameters
CONNECTED

Figure 8-65 – Master/Slave parameters instant management

The LLD is in charge to check the bandwidth used by other links to determine the new parameters to use and to program the next event.

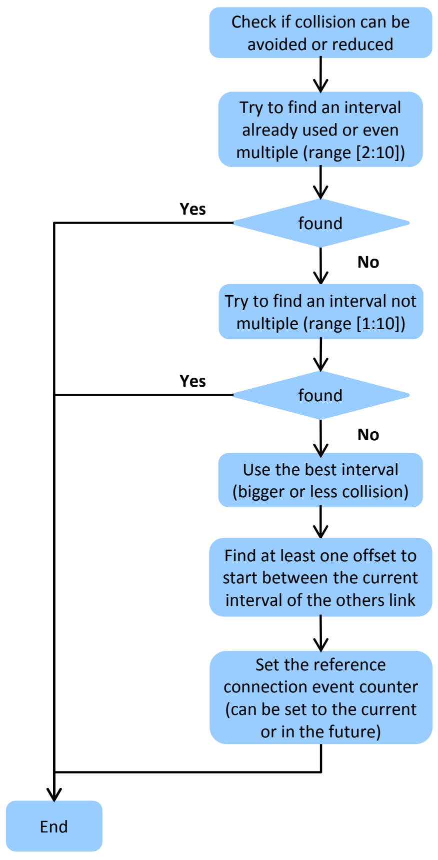

Check if collision can be avoided or reduced
Try to find an interval already used or even multiple (range [2:10])
found
Yes
No
Try to find an interval not multiple (range [1:10])
found
Yes
No
Use the best interval (bigger or less collision)
Find at least one offset to start between the current interval of the others link
Set the reference connection event counter (can be set to the current or in the future)
End

Figure 8-66 – Master/Slave LLD select interval and offset for a parameter update procedure

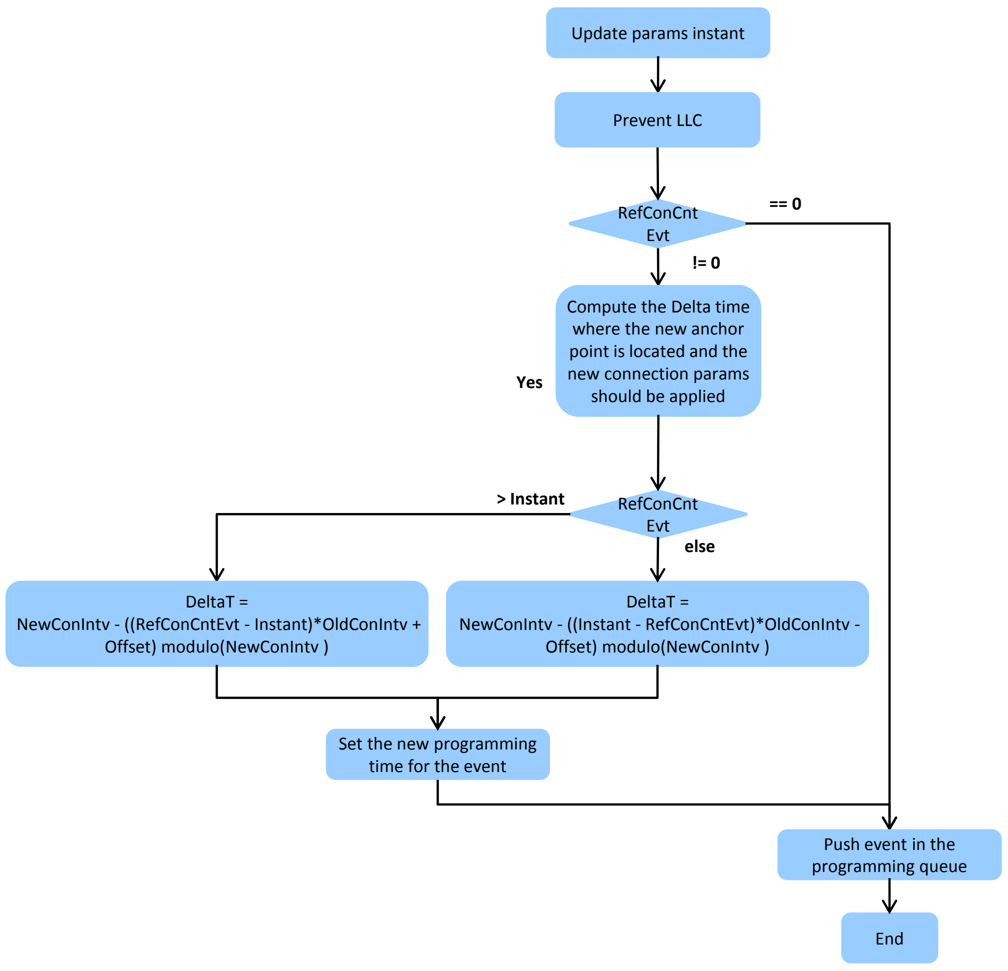

Update params instant
Prevent LLC
RefConCnt Evt
== 0
!= 0
Compute the Delta time where the new anchor point is located and the new connection params should be applied
Yes
> Instant
RefConCnt Evt
else
DeltaT = NewConIntv - ((RefConCntEvt - Instant)*OldConIntv + Offset) modulo(NewConIntv)
DeltaT = NewConIntv - ((Instant - RefConCntEvt)*OldConIntv - Offset) modulo(NewConIntv)
Set the new programming time for the event
Push event in the programming queue
End

Figure 8-67 – Master/Slave LLD programs event with the new parameters

The PDU are described in document [1] Volume 6 Part B Chapter 2.4.2.14, 2.4.2.16 and 2.4.2.17.
The Command are described document [1] Volume 2 Part E Chapter 7.8.31, 7.8.32 and 7.8.18.

2. Automatic request of a parameter update.

The Link Layer can decide to move an anchor point to maintain its link quality:

- Avoid anchor point overlapping.
- Avoid bandwidth drastic reduction.

This procedure is started by the LLM on connection request demand.

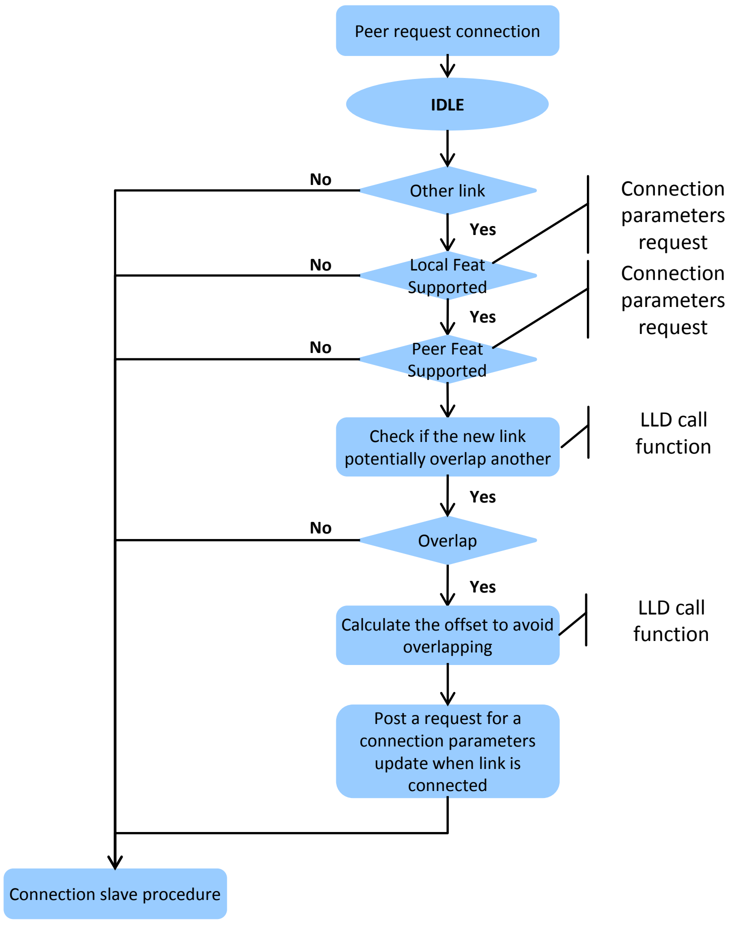

Peer request connection
IDLE
Other link
No
Yes
Local Feat Supported
No
Yes
Peer Feat Supported
No
Connection parameters request
Connection parameters request
Check if the new link potentially overlap another
LLD call function
Yes
Overlap
No
Yes
Calculate the offset to avoid overlapping
LLD call function
Post a request for a connection parameters update when link is connected
Connection slave procedure

Figure 8-68 – Slave LLM posts an update parameters procedure

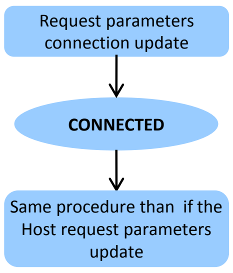

Request parameters connection update
CONNECTED
Same procedure than if the Host request parameters update

Figure 8-69 – Slave LLC starts update parameters request procedure

The decision of the offset value depends on the available bandwidth: if the algorithm finds a bandwidth full, it decides to share fairly the bandwidth between the connection (e.g: 2 links→BW = 50%, 3 links→BW = 33%, …). It means that a connection parameters request procedure is started (if feature supported) on all the links to match with the bandwidth requirements.

When a link is disconnected a connection parameters request procedure is started (if feature supported) on all the remaining links to reassign the bandwidth.

8.6.9 LE ping MSC

The LE Ping procedure, when supported, can be used by the Link Layer to verify the presence of the remote Link Layer. (See [1] Volume 6 Part B Chapter 5.1.8).
The figure below shows how to start the procedure:

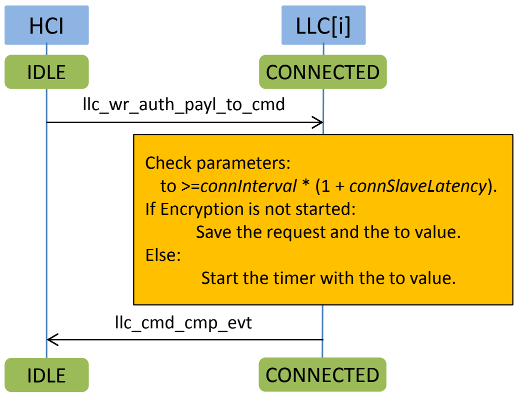

HCI
LLC[i]
IDLE
CONNECTED
llc_wr_auth_payl_to_cmd
Check parameters:
to >=connInterval * (1 + connSlaveLatency).
If Encryption is not started:
Save the request and the to value.
Else:
Start the timer with the to value.
llc_cmd_cmp_evt
IDLE
CONNECTED

Figure 8-70 – Master/Slave Start Ping procedure MSC

If the link is encrypted then the timer is started with a value smaller than the "real TO", called "Nearly TO".
This "Nearly TO" allows to send the control PDU (PING_REQ) and to receive the response (PING_RSP) before the expiration of the "real TO".
The "real TO" is equal to the TO set by the command minus the "Nearly TO" value. It is started when the "Nearly TO" expires.

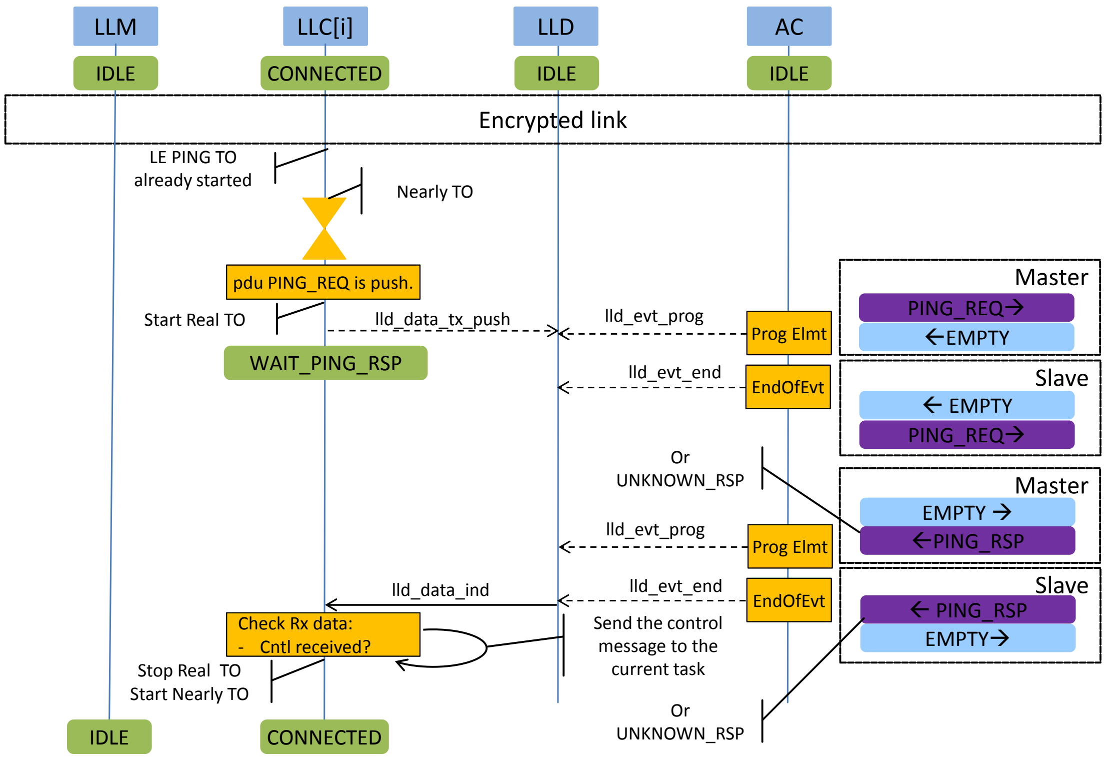


Figure 8-71 – Master/Slave Ping Nearly TO expiration MSC

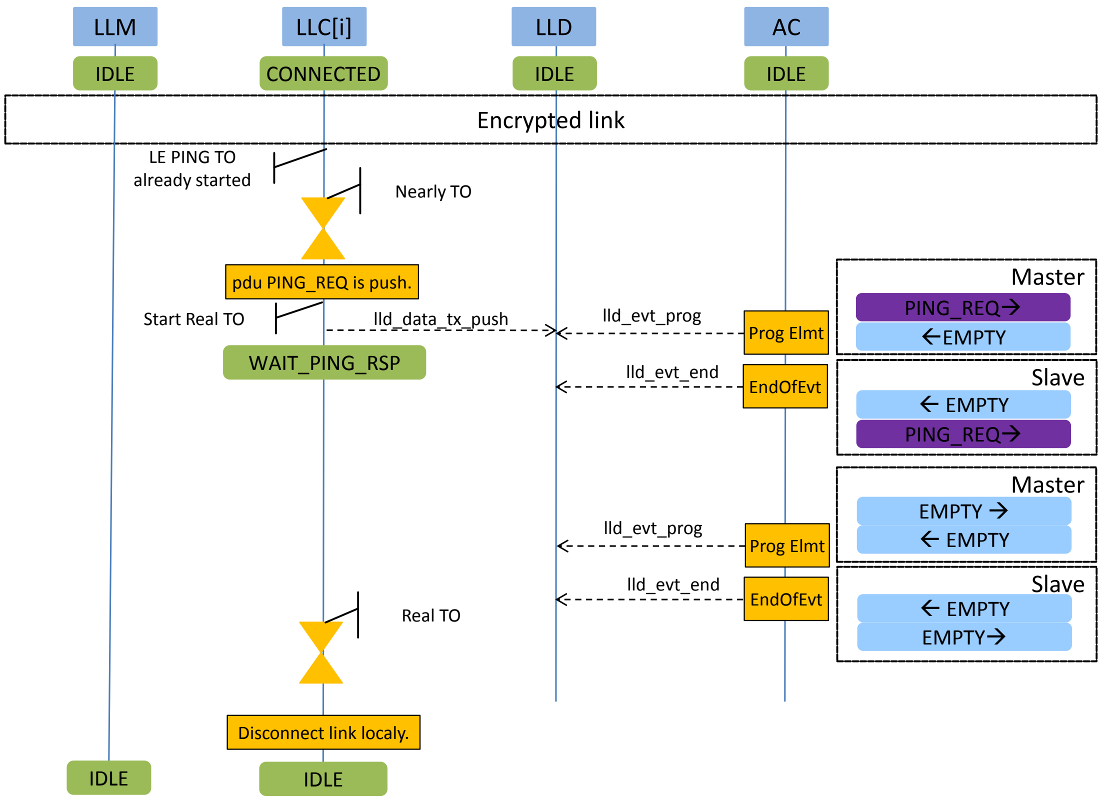


Figure 8-72 – Master/Slave Ping Real TO expiration MSC

8.6.10 Disconnection MSC

A disconnection procedure can be requested by the host or automatically launched if a MIC error is detected, or no synchronization is detected before the LSTO expiration. (See [1] Volume 6 Part B Chapter 5.1.6).

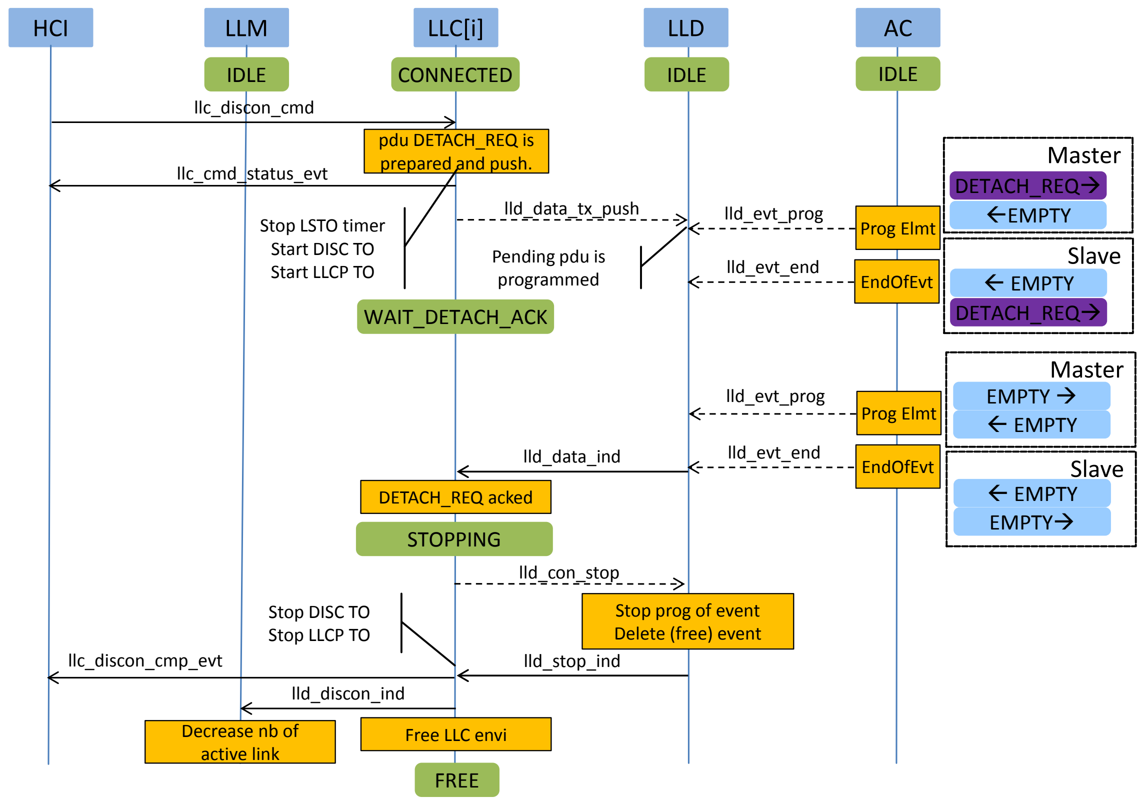


Figure 8-73 – Master/Slave disconnect MSC

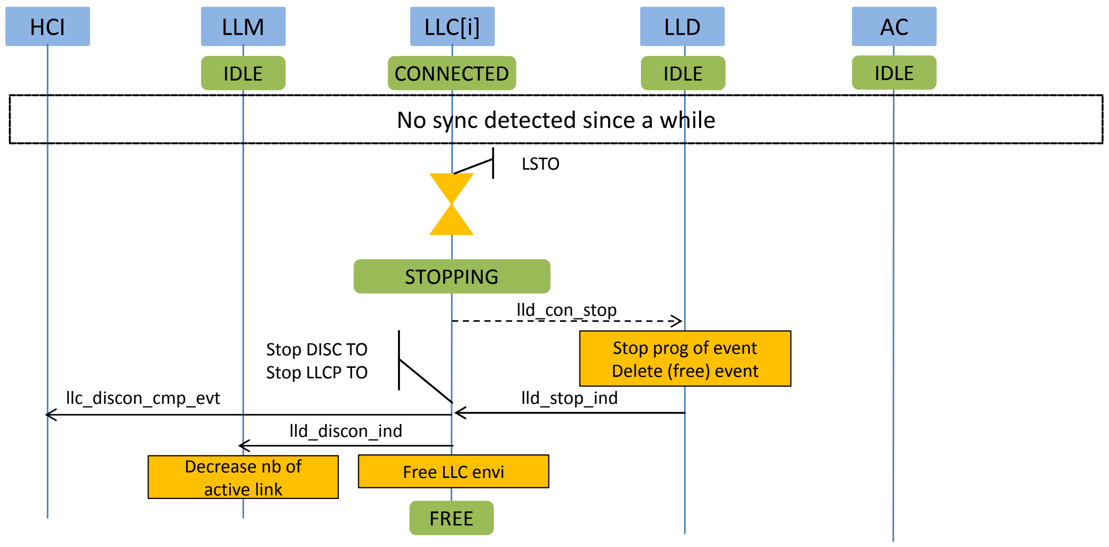


Figure 8-74 – LSTO expiration MSC

8.6.11 Data Length Extension

Data length procedure is used automatically by the controller to negotiate transmit and received packet size and time for a new connection or on host demand to change transmit packet time and size.

8.6.11.1 Environment

The link layer environment is split in two parts:

- General part

  This part uses the LLM environment and contains all the information use for a new connection and the maximum values supported by the controller.

Parameters	Description
connInitialMaxTxOctets	The value of connection maximum transmit octets that the controller will use for a new connection
connInitialMaxTxTime	The value of connection maximum transmit time that the controller will use for a new connection
supportedMaxTxOctets	The maximum value of connection maximum transmit octets that the controller supports
supportedMaxTxTime	The maximum value of connection maximum transmit time that the controller supports
supportedMaxRxOctets	The maximum value of connection maximum receive octets that the controller supports
supportedMaxRxTime	The maximum value of connection maximum receive time that the controller supports

Table 8-4 – DLE LLM environment: fields' description

- Link dedicated part

  This part use the LLC environment and contains all the information exchanged during the procedure and the most important the effective values used by the link.

Parameters	Description
connMaxTxOctets	The maximum number of octets in the Payload field that the local device will send to the remote device
connMaxRxOctets	The maximum number of octets in the Payload field that the local device is able to receive from the remote device
connMaxTxTime	The maximum number of microseconds that the local device will take to transmit a PDU to the remote device
connMaxRxTime	The maximum number of microseconds that the local device can take to receive a PDU from the remote device
connRemoteMaxTxOctets	The maximum number of octets in the Payload field that the remote device will send to the local device
connRemoteMaxRxOctets	The maximum number of octets in the Payload field that the remote device is able to receive from the local device
connRemoteMaxTxTime	The maximum number of microseconds that the remote device will take to transmit a PDU to the local device
connRemoteMaxRxTime	The maximum number of microseconds that the remote device can take to receive a PDU from the local device
connEffectiveMaxTxOctets	The lesser of connMaxTxOctets and connRemoteMaxRxOctets
connEffectiveMaxRxOctets	The lesser of connMaxRxOctets and connRemoteMaxTxOctets
connEffectiveMaxTxTime	The lesser of connMaxTxTime and connRemoteMaxRxTime
connEffectiveMaxRxTime	The lesser of connMaxRxTime and connRemoteMaxTxTime

Table 8-5 – DLE LLC environment: fields' description

Link layer supported values:

Parameters	Value

MinOctets	27 (octets)
MaxOctets	251 (octets)
MinTime	328 (µsec)
MaxTime	2120 (µsec)

Table 8-6 – DLE range values

8.6.11.2 Procedures

8.6.11.2.1 Read/Write suggested default value

The Host can specify its preferred values for the controller's maximum transmission number of payload octets and maximum packet transmission time to be used for new connections over HCI commands.

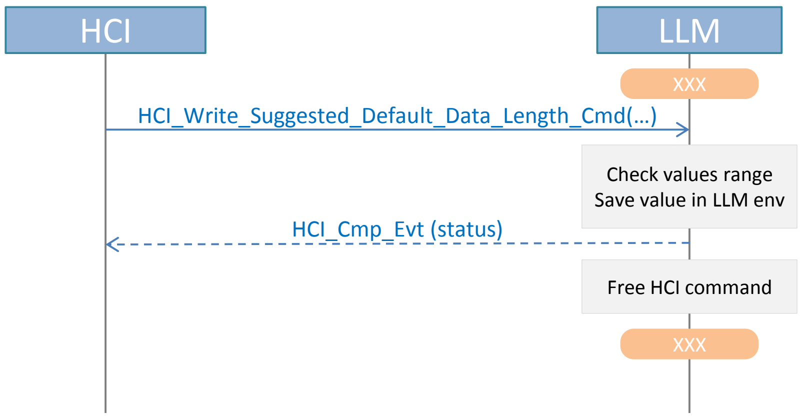


Figure 8-75 – DLE Write suggested value for new connection

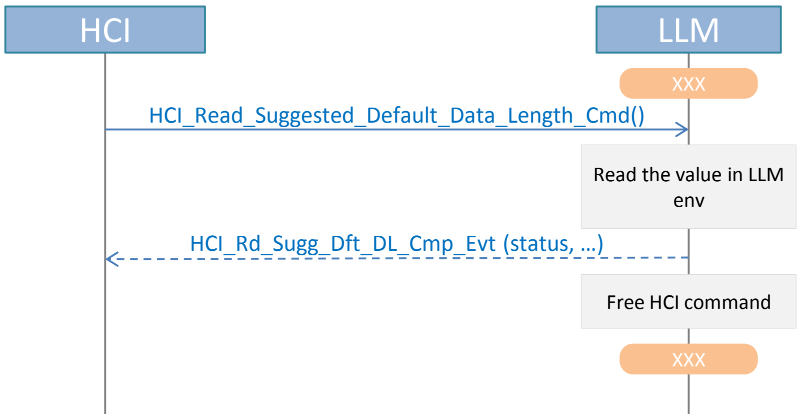

HCI
LLM
XXX
HCI_Read_Suggested_Default_Data_Length_Cmd()
Read the value in LLM env
HCI_Rd_Sugg_Dft_DL_Cmp_Evt (status, …)
Free HCI command
XXX

Figure 8-76 – DLE read suggested value for new connection

The Host to read the controller's maximum supported payload octets and packet duration times for transmission and reception over a HCI command.

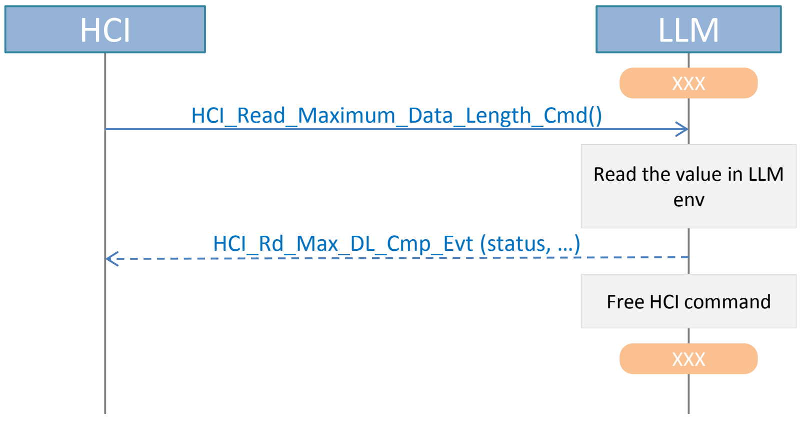

HCI
LLM
XXX
HCI_Read_Maximum_Data_Length_Cmd()
Read the value in LLM env
HCI_Rd_Max_DL_Cmp_Evt (status, …)
Free HCI command
XXX

Figure 8-77 – DLE read maximum value supported by the controller

8.6.11.2.2 Automatically started (link creation)

The link layer starts immediately the procedure after connection to use the maximum RX and TX packet size and time supported by the controller (251 octets and 2120 µs).

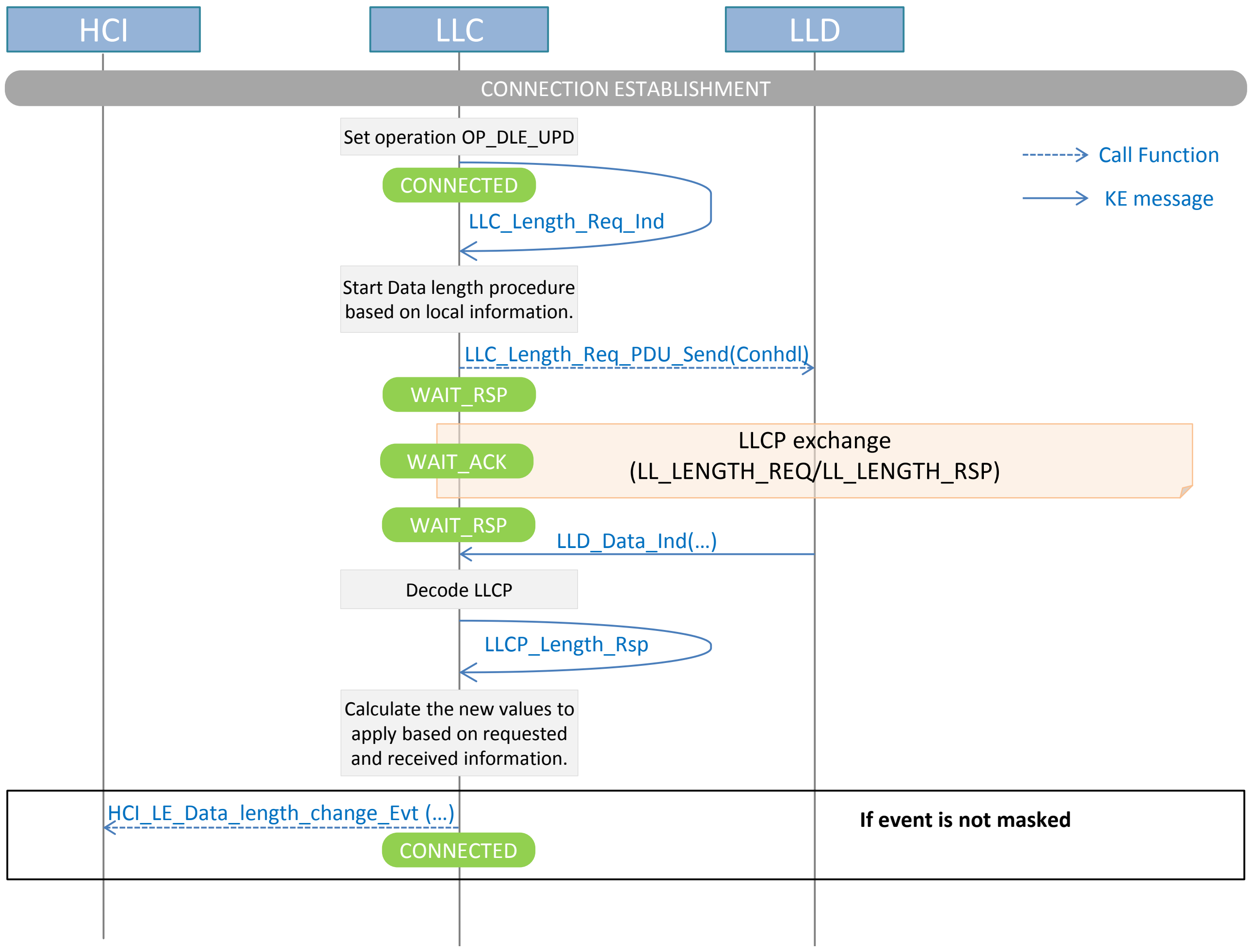

HCI
LLC
LLD
CONNECTION ESTABLISHMENT
Set operation OP_DLE_UPD
CONNECTED
LLC_Length_Req_Ind
Call Function
KE message
Start Data length procedure based on local information.
LLC_Length_Req_PDU_Send(Conhdl)
WAIT_RSP
LLCP exchange (LL_LENGTH_REQ/LL_LENGTH_RSP)
WAIT_ACK
WAIT_RSP
LLD_Data_Ind(…)
Decode LLCP
LLCP_Length_Rsp
Calculate the new values to apply based on requested and received information.
HCI_LE_Data_length_change_Evt (…)
CONNECTED
If event is not masked

Figure 8-78 – DLE automatically started by controller

If the peer negotiates the TX/RX size and time, the effective values are computed accordingly to the peer response (following the rules defined in the Table 8-5).
If a value has changed an event is sent, on both sides, to prevent the host of this change (except if the event is masked).

8.6.11.2.3 Host request

Host can suggest maximum transmission packet size and maximum packet transmission time.

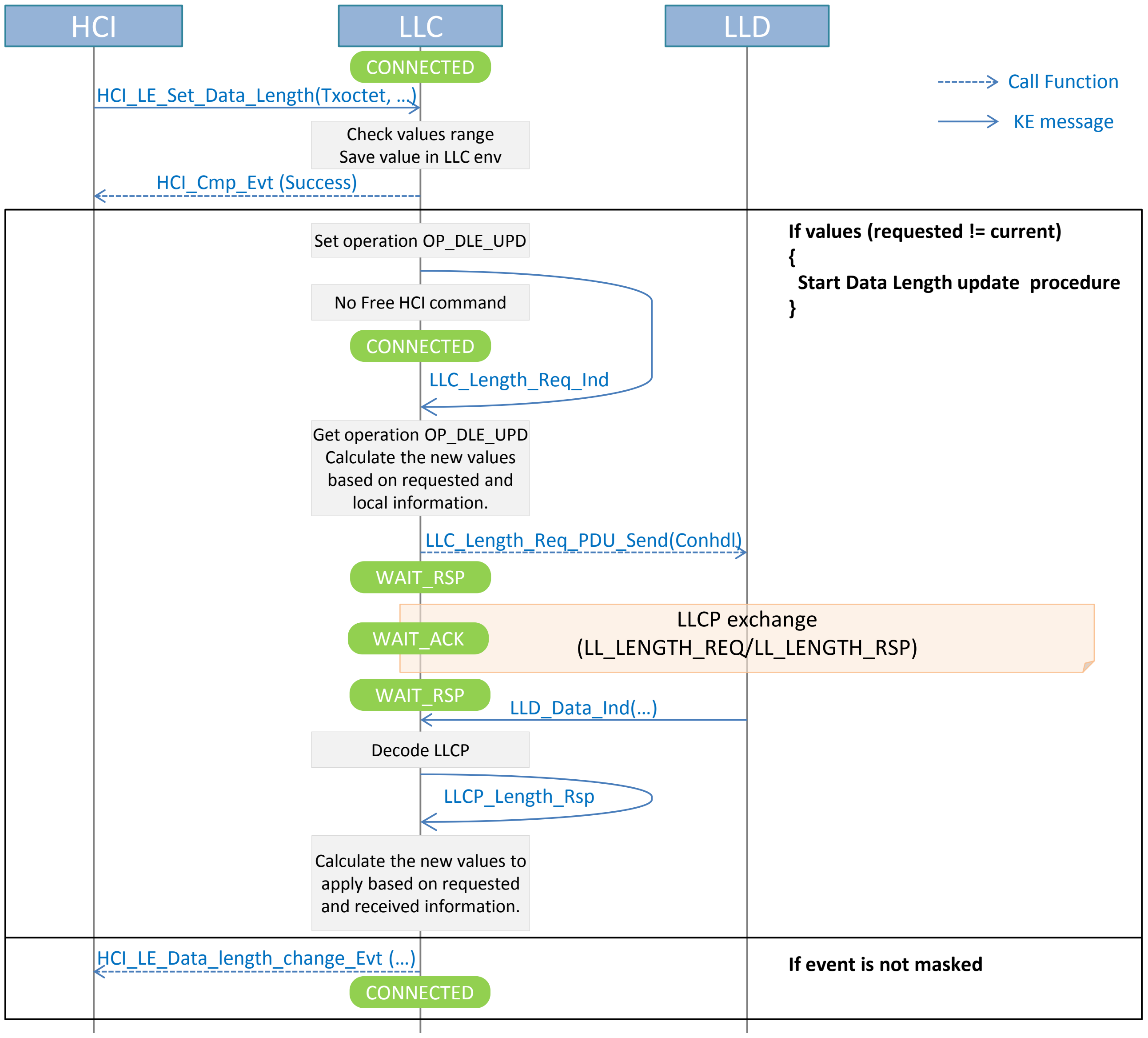

HCI
LLC
LLD
CONNECTED
HCI_LE_Set_Data_Length(Txoctet, ...)
Call Function
KE message
Check values range
Save value in LLC env
HCI_Cmp_Evt (Success)
Set operation OP_DLE_UPD
If values (requested != current)
{
Start Data Length update procedure
}
No Free HCI command
CONNECTED
LLC_Length_Req_Ind
Get operation OP_DLE_UPD
Calculate the new values based on requested and local information.
LLC_Length_Req_PDU_Send(Conhdl)
WAIT_RSP
LLCP exchange
(LL_LENGTH_REQ/LL_LENGTH_RSP)
WAIT_ACK
WAIT_RSP
LLD_Data_Ind(...)
Decode LLCP
LLCP_Length_Rsp
Calculate the new values to apply based on requested and received information.
HCI_LE_Data_length_change_Evt (...)
If event is not masked
CONNECTED

Figure 8-79 – DLE started by host

8.6.11.2.4 Responder side

Anyone requests the DLE procedure the responder reacts by the same way; it gets the requested values checks if the range is respected then merge them with the local information before sending the response to the peer.

If a value has changed an event is sent, on both sides, to prevent the host of this change (except if the event is masked).

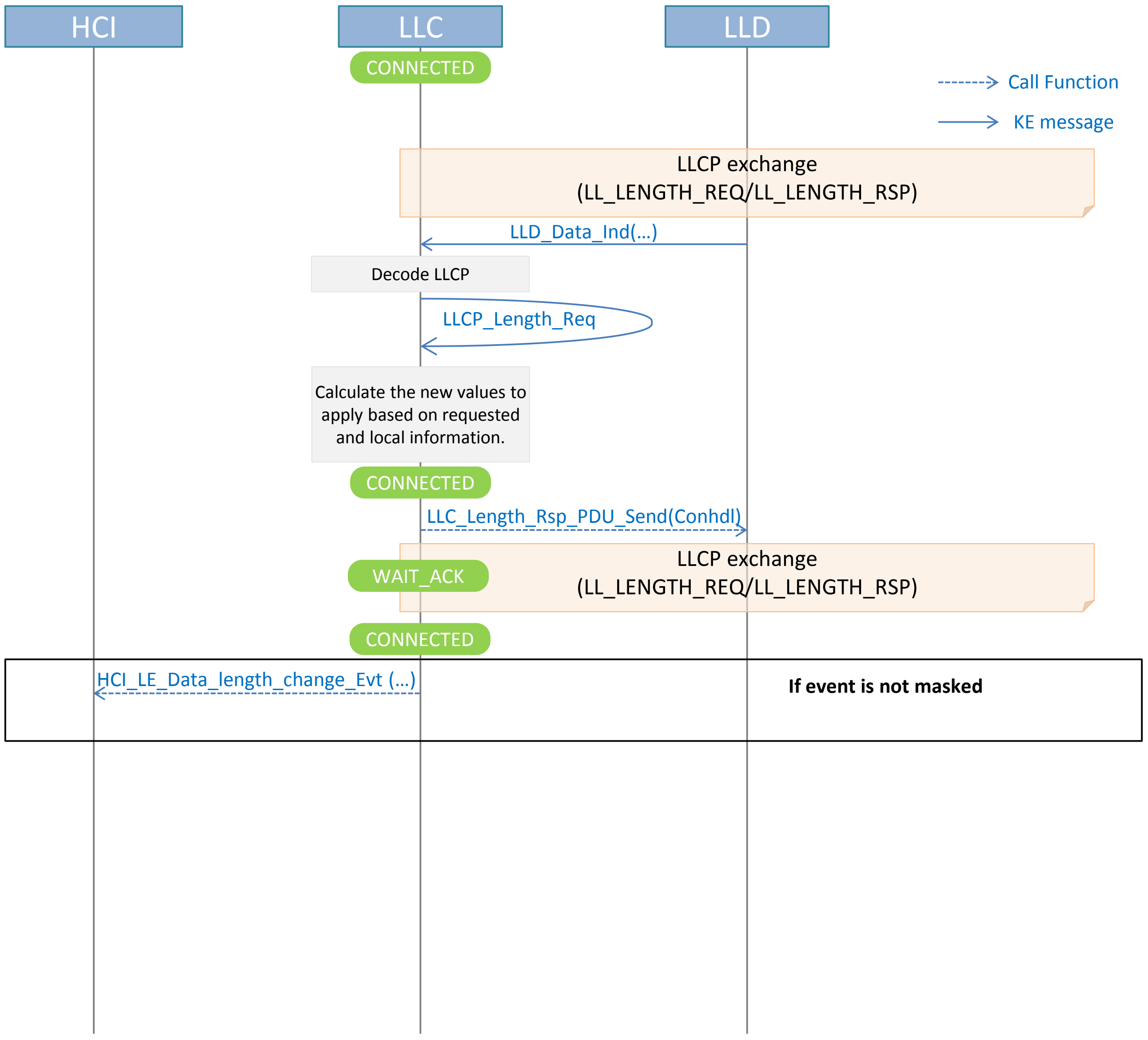

HCI
LLC
LLD
CONNECTED
Call Function
KE message
LLCP exchange (LL_LENGTH_REQ/LL_LENGTH_RSP)
LLD_Data_Ind(...)
Decode LLCP
LLCP_Length_Req
Calculate the new values to apply based on requested and local information.
CONNECTED
LLC_Length_Rsp_PDU_Send(Conhdl)
WAIT_ACK
LLCP exchange (LL_LENGTH_REQ/LL_LENGTH_RSP)
CONNECTED
HCI_LE_Data_length_change_Evt (...)
If event is not masked

Figure 8-80 – DLE responder side

All new data packet will be sent with the new values and all pending packet(s) with the previous values.
The collision doesn't exist, upon receiving an LL_LENGTH_REQ PDU, the Link Layer shall respond with an LL_LENGTH_RSP PDU.

8.7 Testing state

When the host requests to enter in test mode, the device goes in dedicated states:

- TX mode state, test packets shall be transmitted from the device under test with a packet interval of 625µs.
- RX mode state, the nominal packet interval of the test packets transmitted from the tester is 625µs. The tester packet interval can be extended up to 12.5ms upon change of the dirty transmitter parameter settings and during verification of the device under test PER reporting functionality.

The SW is in charge to set:

- The pattern

- The channel
- The length
- Allocate a Tx or Rx descriptor
- Set the CS accordingly to the mode
- Program as soon as possible the event

There is no end of event (except when the test mode is stopped by an abort request), the HW schedules automatically the transmission or reception every 625us (see [2]).

When the test mode has ended, an event is sent back to host. The parameter returned is the number of packet received (set to NULL for Tx test mode).

9 Deep Sleep mode

This chapter describes the algorithm in place to switch from the active clock to the low power clock.

More details are available in [2] and [5].

9.1 Entering in deep sleep mode algorithm

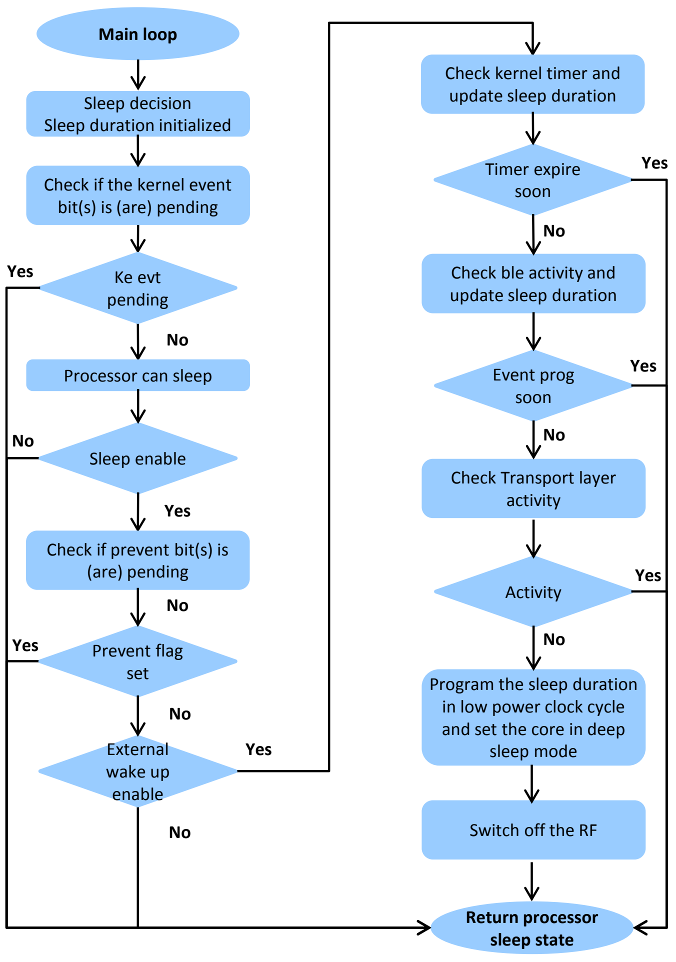

Main loop
Sleep decision
Sleep duration initialized
Check if the kernel event bit(s) is (are) pending
Ke evt pending
Yes
No
Processor can sleep
Sleep enable
No
Yes
Check if prevent bit(s) is (are) pending
No
Prevent flag set
Yes
No
External wake up enable
Yes
No
Check kernel timer and update sleep duration
Timer expire soon
Yes
No
Check ble activity and update sleep duration
Event prog soon
Yes
No
Check Transport layer activity
Activity
Yes
No
Program the sleep duration in low power clock cycle and set the core in deep sleep mode
Switch off the RF
Return processor sleep state

Figure 9-1 - Entering Deep Sleep algorithm

9.2 Exiting from deep sleep mode algorithm

The wake up is performed by an interruption (wake up or external). The stack performs the clock correction on wake-up, and then waits for the future event to be scheduled, or the execution of external uart tasks.

As the algorithm to enter in sleep mode is located in the main loop, the check is continuous.

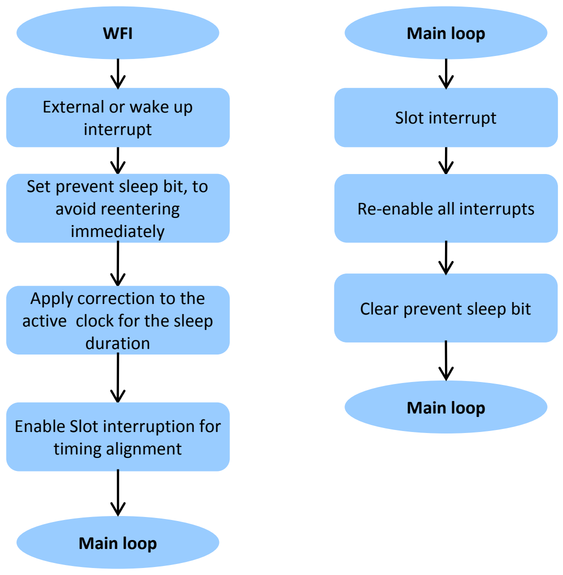

WFI
External or wake up interrupt
Set prevent sleep bit, to avoid reentering immediately
Apply correction to the active clock for the sleep duration
Enable Slot interruption for timing alignment
Main loop
Main loop
Slot interrupt
Re-enable all interrupts
Clear prevent sleep bit
Main loop

Figure 9-2 - Exiting Deep Sleep algorithm

References

[1]	**Title**	Specification of the Bluetooth System		
	Reference	Core_v4.1		
	Version	V4.1	**Date**	December 2013
	Source	Bluetooth SIG		

[2]	**Title**	RW BLE Core Functional Specification		
	Reference	RW-BLE-CORE-FS		
	Version	7.0.16	**Date**	March 2014
	Source	RivieraWaves		

[3]	**Title**	RW Host Controller Interface Functional Specification		
	Reference	RW-HCI-SW-FS		
	Version		**Date**	
	Source	RivieraWaves		

[4]	**Title**	RW Kernel Functional Specification		
	Reference	RW-KERNEL-SW-FS		
	Version	1.1	**Date**	Mai 2012
	Source	RivieraWaves		

[5]	**Title**	RW Deep Sleep Functional Specification		
	Reference	RW-DEEPSLEEP-SW-FS		
	Version	1.0	**Date**	February 2012
	Source	RivieraWaves		

[6]	**Title**	RW Event Arbiter Functional Specification		
	Reference	RW-EA-SW-FS		
	Version	1.0	**Date**	
	Source	RivieraWaves		