STM32WB Series BLE low level driver (LLD)

Introduction

This document describes Bluetooth® Low Energy (refered to as BLE in this document) low level driver (LLD) for the STM32WB Series products , which provides access to the STM32WB Series radio device to send and receive packets in BLE radio format.

BLE LLD is a radio communication layer. It relies on BLE radio hardware, it is a proprietary radio abstraction layer, and not a BLE stack.

BLE LLD provides a light and simple layer for developing proprietary protocols and applications.

Two layers are available:

- LLD with full features
- HAL with simple API

1 General information

This document applies to the STM32WB Series Arm®-based microprocesor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Glossary

Table 1. Glossary

Term	Definition
ACK	Acknowledgment
ActionPacket	Structure used for packet transmission or reception
API	Application programming interface
BLE	Bluetooth® Low Energy
CRC	Cyclic redundancy check
FSM	Finite state machine
HAL	Hardware abstraction layer (upper level of BLE LLD)
IPCC	Inter processor communication controller
ISR	Interrupt service routine
LL	Low level (lower level of BLE LLD)
LLD	Low level driver
MIC	Message integrity check
Rx	Reception
Tx	Transmission
Whitening	Scrambling of the data to avoid patterns which leads to bad radio behavior

2 Architecture

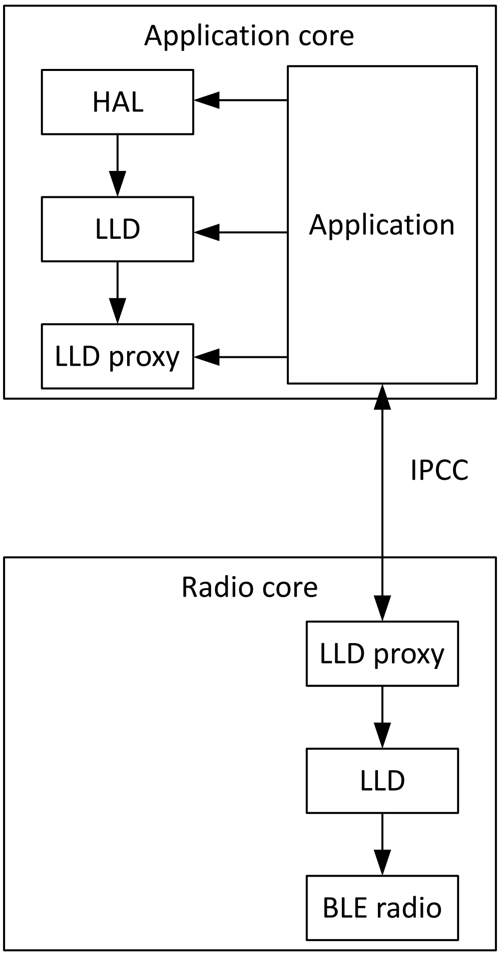


Figure 1. Figure : BLE implementation architecture overview

On application core:

- The application is the user program implementing a custom radio protocol
- HAL is a wrapper based on LLD for simple communication
- LLD is the layer for full features communications
- LLD proxy packs/unpacks data and commands to/from radio core.

On the radio core:

- The LLD proxy packs/unpacks data and commands to/from application core
- The LLD provides the radio abstraction
- The BLE radio is the RF hardware.

2.1 Dual core

The BLE LLD is designed to run on a dual core hardware:

- The application core runs user code
- The radio core runs private code dedicated to radio management

The software communication layer between both cores is called IPCC. This transport layer is decoupled from the BLE LLD.

This architecture brings some important constraints:

- No application code runs on radio core
- It takes a long time to run application code after a radio event.

To help implement fast radio operation sequences despite those constraints, "action packets" must be chained by the radio core. This chaining is configured by the application.

Those constraints impact protocol design and choice between LLD and HAL.

2.2 Action packet

The action packet is the structure used by radio core to control the transmission and reception of radio packets.

When action packets are chained to execute complex radio sequences, the chaining is configured with two fields of the action packet. One for the next action packet to run if operation (Tx or Rx) succeed, and the other for the next action packet to run if an operation fails.

2.2.1 Back-to-back vs wake-up

Delays between action packets (or before the first packet) are configured with two modes: back-to-back or wake-up.

- In back-to-back mode, everything stays powered up, so it offers the lowest delays between action packets. Back-to-back time is a global parameter configured with `BLE_LLD_SetBackToBackTime()`, it cannot be configured separately for each action packet.
- In wake-up mode, the radio goes to sleep, so the delay between action packets cannot be as short as in back-to-back mode. Wake-up time is configured separately for each action packet.

Note: For the first packet of a sequence, since the radio is not doing anything, wake-up mode must be used.

2.3 Radio packet details

Packets include an address that must match the configured address of the recipient.

Each packet is to carry a payload up to a maximum of 255 bytes (less if using encryption).

Packets include a CRC and are checked for error at reception.

3 Usage

3.1 Blocking functions

API functions are synchronous (but a radio transmission/reception may still be running after return).

3.2 Radio proxy configuration

Due to the dual core hardware, user does not have direct access to the radio core, but must control radio through a proxy. Since BLE LLD is independent from the communication layer with the radio core, it must be "wired" by the application, the wiring functions are prefixed `BLE_LLD_PRX_`.

`BLE_LLD_PRX_Init()` must be called first to configure the radio core proxy.
`BLE_LLD_PRX_EventProcessInter()` and `BLE_LLD_PRX_EventProcessTask()` are responsible for radio events processing.

Note: The proxy configuration is required no matter which API (LLD/HAL) is used.

3.3 Radio events

The user can register a callback function when a radio operation is started or configured.

When an event occurs on the radio core (for example transmission success, reception failure, and so on) the BLE LLD proxy on the application core is notified and in turn runs the callback if one was registered for this event.

This mechanism allows the user application to react to radio events.

3.4 HAL interface

The HAL is just a layer on top of the LLD to simplify the communication. The HAL has a simple API with limited features, it is used when no custom packet chaining is required.

3.4.1 HAL configuration

Before any packet exchange, the HAL must be initialized with `HAL_BLE_LLD_Init()` then configured with `HAL_BLE_LLD_Configure()`.

3.4.2 HAL communication

Two sets of functions are available:

- without ACK: the radio transmits or receives just one packet
- with ACK: the radio transmits or receives one packet, then another packet goes in the opposite direction

"With ACK" functions can be used to detect packet loss, thus they can be used to implement a reliable communication channel with the retransmission of lost packets.

3.5 LLD interface

The LLD is the layer that exposes all the features supported by the radio core. Its API is more complex and is used to implement custom packets chaining.

3.5.1 LLD configuration

Before any packet is exchanged, the LLD must be initialized with `BLE_LLD_Init()` then configured with:

- `BLE_LLD_SetChannel()`
- `BLE_LLD_SetTxAttributes()`
- `BLE_LLD_SetTxPower()`
- `BLE_LLD_SetTx_Rx_Phy()`

3.5.1.1 *LLD encryption*

The BLE LLD only provides low level cryptography functions originally intended to support security in the BLE upper layers. Issues like key storage/exchange, authentication, and secure communication setup are not covered in this document and require specific design and review from security experts.

3.5.2 LLD communication

With the LLD API, the user is responsible for the configuration of each action packet.

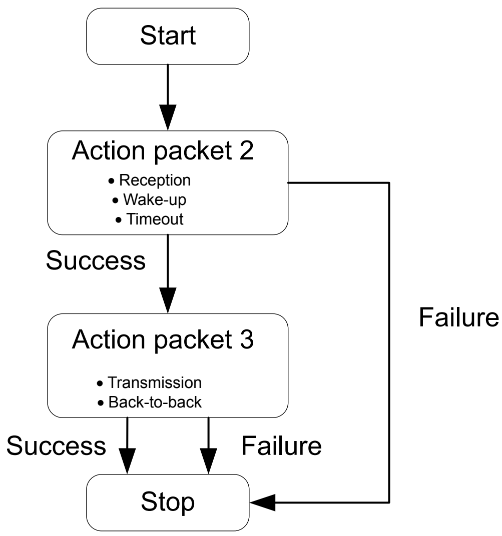
Each action packet must be configured by setting all the required fields for the desired action (some fields are specific to Tx, others to Rx) then calling `BLE_LLD_SetReservedArea()`.

To start the execution of the first action packet, call `BLE_LLD_MakeActionPacketPending()` on that action packet. The radio core then chains action packets based on their configuration and the result of the radio operation. At the end of each action packet, an event is sent to the application if a callback was registered for that action packet.

Action packets chaining can be interrupted with `BLE_LLD_StopActivity()`. This stops the radio, and a new initialization is required before any other operation.

The diagram below shows the action packets configured to implement `HAL_BLE_LLD_ReceivePacketWithAck()`.

Figure 2. **Figure : Action packet configuration overview**



Action packet 2 is run first, it configures the reception of the data packet. If data is properly received (CRC OK), action packet 3 is run next and it configures the transmission of the ACK packet. Then the radio stops. If an action packet fails, the radio stops.

3.6 Tone generation

For test purposes, a tone can be generated with `BLE_LLD_StartTone()`. Use `BLE_LLD_StopTone()` to stop this tone. After a tone, the radio must be reinitialized before any other operation.

4 LLD proxy API

4.1 BLE_LLD_PRX_Init

4.1.1 Description

`BLE_LLD_PRX_Init()` initializes the BLE LLD proxy parameters. This function must be called before any BLE LLD function.

4.1.2 Syntax

```
void BLE_LLD_PRX_Init(param_BLE_LLD_t *parameters,
                      ipBLE_lld_txrxdata_Type *transmitBuffer,
                      ipBLE_lld_txrxdata_Type *transmitBuffer,
                      ipBLE_lld_txrxdata_Type *receiveBuffer,
                      uint8_t (*callbackSend)(BLE_LLD_Code_t bleCmd));
```

Table 2. **BLE_LLD_PRX_Init parameter**

Variable	Description
`parameters`	Parameters for command and response to/from the Cortex®-M0
`transmitBuffer`	Buffer for packet to send
`receiveBuffer`	Buffer for packet received
`bleCmd`	Function to send commands to the Cortex®-M0

4.2 BLE_LLD_PRX_EventProcessInter

4.2.1 Description

`BLE_LLD_PRX_EventProcessInter()` processes any received event from radio core during interruption. It must be called during the interruption that is triggered when a radio event is received from the radio core. It stores the event data for further processing after interruption.

4.2.2 Syntax

```
void BLE_LLD_PRX_EventProcessInter(radioEventType event);
```

Table 3. **BLE_LLD_PRX_EventProcessInter**

Variable	Description
`event`	Radio core event type

4.3 BLE_LLD_PRX_EventProcessTask

4.3.1 Description

`BLE_LLD_PRX_EventProcessTask()` processes received event from radio core after interruption. It must be called after the interruption that is triggered when a radio event is received from the radio core, in a task. It runs the callback which is registered for the received event.

4.3.2 Syntax

```
void BLE_LLD_PRX_EventProcessTask(void);
```

5 HAL API of BLE LLD

5.1 HAL_BLE_LLD_Init

5.1.1 Description

`HAL_BLE_LLD_Init` initializes the radio. Whitening is forced. Before actual use, the radio must be configured with `HAL_BLE_LLD_Configure()`.

5.1.2 Syntax

```
uint8_t HAL_BLE_LLD_Init(uint16_t hsStartupTime, bool lsOscInternal);
```

Table 4. HAL_BLE_LLD_Init parameter

Variable	Description
hsStartupTime	Startup time (system time unit)
lsOscInternal	Use internal RO for the 32 kHz slow speed clock (else external crystal)

5.2 HAL_BLE_LLD_Configure

5.2.1 Description

`HAL_BLE_LLD_Configure()` configures the radio.

5.2.2 Syntax

```
uint8_t HAL_BLE_LLD_Configure(txPower_t txPower,
                              uint8_t channel,
                              bool phy2mbps,
                              uint32_t b2bTimeUs,
                              uint32_t networkId);
```

Table 5. HAL_BLE_LLD_Configure parameter

Variable	Description
txPower	Transmit power for outgoing packets
channel	Radio channel (0 - 39)
phy2mbps	Use 2 Mb/s PHY speed (else 1 Mb/s)
b2bTimeUs	Back to back time (μs), delay between packet and ACK
networkId	Network ID (access address)

5.3 HAL_BLE_LLD_SendPacket

5.3.1 Description

`HAL_BLE_LLD_SendPacket()` sends one packet without listening for an acknowledge.

5.3.2 Syntax

```
uint8_t HAL_BLE_LLD_SendPacket(void *data,
                               uint8_t size,
                               lldCallback_t *callback);
```

Table 6. HAL_BLE_LLD_SendPacket parameters

Variable	Description
data	Data to transmit
size	Size of data to transmit
callback	Function that is called once radio operation is done

5.4 HAL_BLE_LLD_SendPacketWithAck

5.4.1 Description

HAL_BLE_LLD_SendPacketWithAck() sends one packet and listens for an acknowledge.

5.4.2 Syntax

```
uint8_t HAL_BLE_LLD_SendPacketWithAck(void *data,
                                      uint8_t size,
                                      uint32_t receiveTimeout,
                                      lldCallback_t *callback)
```

Table 7. HAL_BLE_LLD_SendPacketWithAck parameters

Variable	Description
data	Data to transmit
size	Size of data to transmit
receiveTimeout	Timeout for ACK reception (µs)
callback	Function that is called once radio operation is done

5.5 HAL_BLE_LLD_ReceivePacket

5.5.1 Description

HAL_BLE_LLD_ReceivePacket() receives one packet without transmitting an acknowledge.

5.5.2 Syntax

```
uint8_t HAL_BLE_LLD_ReceivePacket(uint32_t receiveTimeout,
                                  lldCallback_t *callback);
```

Table 8. HAL_BLE_LLD_ReceivePacket parameters

Variable	Description
receiveTimeout	Timeout for data reception (µs)
callback	Function that is called once radio operation is done

5.6 HAL_BLE_LLD_ReceivePacketWithAck

5.6.1 Description

`HAL_BLE_LLD_ReceivePacketWithAck()` receives one packet and transmits an acknowledge.

5.6.2 Syntax

```
uint8_t HAL_BLE_LLD_ReceivePacketWithAck(void *ack,
                                         uint8_t size,
                                         uint32_t receiveTimeout,
                                         lldCallback_t *callback);
```

Table 9. **HAL_BLE_LLD_ReceivePacketWithAck parameters**

Variable	Description
`ack`	Acknowledge to transmit
`size`	Size of acknowledge to transmit
`receiveTimeout`	Timeout for data reception (μs)
`callback`	Function that is called once radio operation is done

6 LL API of BLE LLD

6.1 Structures

`ActionPacket` parameters are described in Table 10. The `ActionPacket` is composed of input fields used to configure the action and output fields which hold the information on the action once it has been executed. The table below describes the `ActionPacket` parameters.

Table 10. ActionPacket fields description

Field name	Description
`StateMachineNo`	State machine number (0 - 7)
`ActionTag`	Bitfield for configuration of the action packet: TXRX, TIMER_WAKEUP, TIMESTAMP_POSITION
`WakeupTime`	Time before runing the action packet (µs). Applicable only if TIMER_WAKEUP flag is set in ActionTag
`ReceiveWindowLength`	Rx window size in (µs). Applicable only for Rx
`data`	Payload to send. Applicable only for Tx
`dataSize`	Size of payload. Applicable only for Tx
`status`	Interrupt status register from the hardware
`rssi`	RSSI of the received packet. Applicable only for Rx
`nextTrue`	Next action packet to run if success
`nextFalse`	Next action packet to run if failure
`actionPacketNb`	Action packet number (0 - 7)
`callback`	Function to run when action packet has finished. If not used, it must be set to NULL

An `ActionPacket` contains `ActionTag` parameters are described in the table below.

Table 11. ActionTag description

Tag name	Description
`TIMESTAMP_POSITION`	This bit sets the position of the time stamp, whether it is located at the beginning or the end of the packet. (only for receive mode): • 0: End of the packet • 1: Beginning of the packet.
`NS_EN`	This bit activates automatic ACK (only for receive mode): • 0: No ACK • 1: Automatic ACK.
`TIMER_WAKEUP`	The bit determines if the action (Rx or Tx) is going to be executed based on the back-to-back time or based on the wake-up time: • 0: Based on the back-to-back time (default 150 us) • 1: Based on the wake-up time
`TXRX`	This bit determines if the action is an Rx action or a Tx action: • 0: For receive • 1: For transmit.

6.2 BLE_LLD_Init

6.2.1 Description

`BLE_LLD_Init()` initializes the radio.

6.2.2 Syntax

```
void BLE_LLD_Init(uint16_t hsStartupTime,
                  uint8_t lowSpeedOsc,
                  FunctionalState whitening);
```

Table 12. **BLE_LLD_Init parameters**

Variable	Description
hsStartupTime	Startup time (system time unit)
lowSpeedOsc	Source for the 32 kHz slow speed clock: • 1: Internal RO • 0: External crystal
Whitening	ENABLE or DISABLE Whitening for transmission and reception

6.3 BLE_LLD_SetReservedArea

6.3.1 Description

`BLE_LLD_SetReservedArea()` configures an action packet. This function must be called after the relevant ActionPacket fields are set.

6.3.2 Syntax

```
void BLE_LLD_SetReservedArea(ActionPacket *p);
```

Table 13. **BLE_LLD_SetReservedArea parameter**

Variable	Description
p	Action packet to prepare, memory lifetime must extend until response processing

6.4 BLE_LLD_MakeActionPacketPending

6.4.1 Description

`BLE_LLD_MakeActionPacketPending()` starts the radio FSM. This function schedules an action packet to be the first executed by the radio FSM. `BLE_LLD_SetReservedArea()` must be called first to prepare the action packet.

6.4.2 Syntax

```
uint8_t BLE_LLD_MakeActionPacketPending(const ActionPacket *p);
```

Table 14. **BLE_LLD_MakeActionPacketPending parameter**

Variable	Description
p	Action packet to schedule, memory lifetime must extend until response processing

6.5 BLE_LLD_GetStatus

6.5.1 Description

`BLE_LLD_GetStatus()` checks if the radio is busy.

6.5.2 Syntax

```
uint8_t BLE_LLD_GetStatus(void);
```

Table 15. RADIO_GetStatus parameter

Variable	Description
Return value	• `BLUE_IDLE_0`: Radio is not busy. • `BLUE_BUSY_NOWAKEUP_T2`: Radio is busy, but there is no wakeup timer on the schedule but timer2 is running. • `BLUE_BUSY_WAKEUP`: Radio is busy and wakeup timer is on the schedule. • `BLUE_BUSY_TONE`: Radio is in Tone generation mode. • `BLUE_TONE_DESTROY`: Radio Tone has destroyed the BLE: the BLE needs an Init

6.6 BLE_LLD_SetChannel

6.6.1 Description

`BLE_LLD_SetChannel()` sets the radio channel.

6.6.2 Syntax

```
void BLE_LLD_SetChannel(uint8_t StateMachineNo,uint8_t channel);
```

Table 16. BLE_LLD_SetChannel parameters

Variable	Description
`StateMachineNo`	State machine (0 - 7)
`channel`	Radio channel (0 - 39)

6.7 BLE_LLD_SetTxAttributes

6.7.1 Description

`BLE_LLD_SetTxAttributes()` sets the network ID (access address).

6.7.2 Syntax

```
void BLE_LLD_SetTxAttributes(uint8_t StateMachineNo,uint32_t NetworkID);
```

Table 17. BLE_LLD_SetTxAttributes parameters

Variable	Description
`StateMachineNo`	State machine (0 - 7)
`NetworkID`	Network ID

6.8 BLE_LLD_SetBackToBackTime

6.8.1 Description

This routine sets the time between back-to-back radio transmissions.

`BLE_LLD_SetBackToBackTime()` sets the back-to-back time. Back-to-back is a mode where two packets are chained with a short pause between them. The back-to-back time is a global parameter. Back-to-back time must be at least 50 µs.

6.8.2 Syntax

```
void BLE_LLD_SetBackToBackTime(uint32_t backToBackTime);
```

Table 18. BLE_LLD_SetBackToBackTime parameter

Variable	Description
`backToBackTime`	Time between two packets in back-to-back mode (µs)

6.9 BLE_LLD_SetTxPower

6.9.1 Description

`BLE_LLD_SetTxPower()` sets the transmit power level.

6.9.2 Syntax

```
void BLE_LLD_SetTxPower(txPower_t powerLevel);
```

Table 19. BLE_LLD_SetTxPower parameter

Variable	Description
`powerLevel`	Transmit power level (0 - 31)
Return value	No return

Table 20. Power level values

Value	Out power (dBm)	Value	Out power (dBm)	Value	Out power (dBM)	Value	Out power (dBM)
0x1F	+6	0x17	-0.5	0x0F	-5.9	0x07	-14.1
0x1E	+5	0x16	-0.85	0x0E	-6.9	0x06	-15.25
0x1C	+3	0x14	-1.8	0x0C	-8.85	0x04	-17.6
0x1B	+2	0x13	-2.45	0x0B	-9.9	0x03	-18.85
0x1A	+1	0x12	-3.15	0x0A	-10.9	0x02	-19.85
0x19	0	0x11	-4	0x09	-12.05	0x01	-20.85
0x18	-0.15	0x10	-4.95	0x08	-13.15	0x00	-40

6.10 BLE_LLD_SetTx_Rx_Phy

6.10.1 Description

`BLE_LLD_SetTx_Rx_Phy()` sets the bitrate for transmission and reception.

6.10.2 Syntax

```
void BLE_LLD_SetTx_Rx_Phy(uint8_t StateMachineNo,uint8_t txPhy,uint8_t rxPhy);
```

Table 21. BLE_LLD_SetTx_Rx_Phy parameters

Variable	Description
StateMachineNo	State machine (0 - 7)
txPhy	Speed for transmission: TX_PHY_1MBPS / TX_PHY_2MBPS
rxPhy	Speed for reception: RX_PHY_1MBPS / RX_PHY_2MBPS
Return value	No return

6.11 BLE_LLD_StopActivity

6.11.1 Description

BLE_LLD_StopActivity() stops the radio FSM. After a call to this function ISR will not be triggered, unless MakeActionPacketPending() is called again. This function returns when the radio is ready to be initialized.

6.11.2 Syntax

```
uint8_t BLE_LLD_StopActivity(void);
```

Table 22. BLE_LLD_StopActivity parameters

Variable	Description
Return value	Always returns TRUE

6.12 BLE_LLD_SetEncryptionCount

6.12.1 Description

BLE_LLD_SetEncryptionCount() sets the 40 bits receive and transmit packet count, used in encryption. Both set the 39-bit count + 1 bit MSB as defined in the Bluetooth Low Energy specifications for encryption nonce calculation.

6.12.2 Syntax

```
void BLE_LLD_SetEncryptionCount(uint8_t StateMachineNo,
                                const uint8_t (*countTx)[5],
                                const uint8_t (*countRx)[5]);
```

Table 23. BLE_LLD_SetEncryptionCount parameters

Variable	Description
StateMachineNo	State machine (0 - 7)
countTx	40-bit transmit packet count
countRx	40-bit receive packet count
Return value	No return

6.13 BLE_LLD_SetEncryptionAttributes

6.13.1 Description

BLE_LLD_SetEncryptionAttributes() sets the encryption initialization vector and the encryption key.

6.13.2 Syntax

```
void BLE_LLD_SetEncryptionAttributes(uint8_t StateMachineNo,
                                     const uint8_t (*encIv)[8],
                                     const uint8_t (*encKey)[16]);
```

Table 24. BLE_LLD_SetEncryptionAttributes parameters

Variable	Description
StateMachineNo	State machine (0 - 7)
encIv	8-byte encryption initialization vector
encKey	16-byte encryption key
Return value	No return

6.14 BLE_LLD_SetEncryptFlags

6.14.1 Description

BLE_LLD_SetEncryptFlags() enables or disables encryption.

6.14.2 Syntax

```
void BLE_LLD_SetEncryptFlags(uint8_t StateMachineNo,FunctionalState EncryptFlag);
```

Table 25. BLE_LLD_SetEncryptFlags parameters

Variable	Description
StateMachineNo	State machine (0 - 7)
EncryptFlag	Encryption state: • 0: Disable • 1: Enable
Return value	No return

6.15 BLE_LLD_StartTone

6.15.1 Description

`BLE_LLD_StartTone()` starts a tone transmission. Use `BLE_LLD_StopTone()` to stop tone.

6.15.2 Syntax

```
void BLE_LLD_StartTone(uint8_t rfChannel,uint8_t powerLevel);
```

Table 26. BLE_LLD_StartTone parameters

Variable	Description
rfChannel	Radio channel (0 - 39)
powerLevel	Output power level (0 - 31)
Return value	No return

Table 27. Output power level values

Value	Out power (dBm)	Value	Out power (dBm)	Value	Out power (dBM)	Value	Out power (dBM)
0x1F	+6	0x17	-0.5	0xF	-5.9	0x7	-14.1
0x1E	+5	0x16	-0.85	0xE	-6.9	0x6	-15.25
0x1C	+3	0x14	-1.8	0xC	-8.85	0x4	-17.6
0x1B	+2	0x13	-2.45	0xB	-9.9	0x3	-18.85
0x1A	+1	0x12	-3.15	0xA	-10.9	0x2	-19.85
0x19	0	0x11	-4	0x9	-12.05	0x1	-20.85
0x18	-0.15	0x10	-4.95	0x8	-13.15	0x0	-40

6.16 BLE_LLD_StopTone

6.16.1 Description

`BLE_LLD_StopTone()` stops tone transmission. After calling this function the radio must be re-initialized.

6.16.2 Syntax

```
void BLE_LLD_StopTone(void);
```

Table 28. BLE_LLD_StopTone parameters

Variable	Description
Return value	No return

Revision history

Table 29. **Document revision history**

Date	Version	Changes
21-Oct-2021	1	Initial release.

Contents

1 General information 2
1.1 Glossary 2
2 Architecture 3
2.1 Dual core 4
2.2 Action packet 4
2.2.1 Back-to-back vs wake-up 4
2.3 Radio packet details 4
3 Usage 5
3.1 Blocking functions 5
3.2 Radio proxy configuration 5
3.3 Radio events 5
3.4 HAL interface 5
3.4.1 HAL configuration 5
3.4.2 HAL communication 5
3.5 LLD interface 5
3.5.1 LLD configuration 5
3.5.2 LLD communication 6
3.6 Tone generation 6
4 LLD proxy API 7
4.1 BLE_LLD_PRX_Init 7
4.1.1 Description 7
4.1.2 Syntax 7
4.2 BLE_LLD_PRX_EventProcessInter 7
4.2.1 Description 7
4.2.2 Syntax 7
4.3 BLE_LLD_PRX_EventProcessTask 7
4.3.1 Description 7
4.3.2 Syntax 7
5 HAL API of BLE LLD 8
5.1 HAL_BLE_LLD_Init 8
5.1.1 Description 8
5.1.2 Syntax 8
5.2 HAL_BLE_LLD_Configure 8
5.2.1 Description 8
5.2.2 Syntax 8

5.3 HAL_BLE_LLD_SendPacket 8
5.3.1 Description 8
5.3.2 Syntax. 8
5.4 HAL_BLE_LLD_SendPacketWithAck 9
5.4.1 Description 9
5.4.2 Syntax. 9
5.5 HAL_BLE_LLD_ReceivePacket 9
5.5.1 Description 9
5.5.2 Syntax. 9
5.6 HAL_BLE_LLD_ReceivePacketWithAck 10
5.6.1 Description 10
5.6.2 Syntax. 10
6 LL API of BLE LLD 11
6.1 Structures 11
6.2 BLE_LLD_Init. 12
6.2.1 Description 12
6.2.2 Syntax. 12
6.3 BLE_LLD_SetReservedArea 12
6.3.1 Description 12
6.3.2 Syntax. 12
6.4 BLE_LLD_MakeActionPacketPending 12
6.4.1 Description 12
6.4.2 Syntax. 12
6.5 BLE_LLD_GetStatus 13
6.5.1 Description 13
6.5.2 Syntax. 13
6.6 BLE_LLD_SetChannel 13
6.6.1 Description 13
6.6.2 Syntax. 13
6.7 BLE_LLD_SetTxAttributes 13
6.7.1 Description 13
6.7.2 Syntax. 13
6.8 BLE_LLD_SetBackToBackTime 14
6.8.1 Description 14
6.8.2 Syntax. 14
6.9 BLE_LLD_SetTxPower. 14
6.9.1 Description 14

6.9.2 Syntax 14
6.10 BLE_LLD_SetTx_Rx_Phy 14
6.10.1 Description 14
6.10.2 Syntax 14
6.11 BLE_LLD_StopActivity 15
6.11.1 Description 15
6.11.2 Syntax 15
6.12 BLE_LLD_SetEncryptionCount 15
6.12.1 Description 15
6.12.2 Syntax 15
6.13 BLE_LLD_SetEncryptionAttributes 15
6.13.1 Description 15
6.13.2 Syntax 16
6.14 BLE_LLD_SetEncryptFlags 16
6.14.1 Description 16
6.14.2 Syntax 16
6.15 BLE_LLD_StartTone 16
6.15.1 Description 16
6.15.2 Syntax 16
6.16 BLE_LLD_StopTone 17
6.16.1 Description 17
6.16.2 Syntax 17
Revision history 18

List of figures

Figure 1. Figure : BLE implementation architecture overview 3
Figure 2. Figure : Action packet configuration overview 6

List of tables

Table 1. Glossary . . . 2
Table 2. BLE_LLD_PRX_Init parameter . . . 7
Table 3. BLE_LLD_PRX_EventProcessInter . . . 7
Table 4. HAL_BLE_LLD_Init parameter. . . . 8
Table 5. HAL_BLE_LLD_Configure parameter . . . 8
Table 6. HAL_BLE_LLD_SendPacket parameters . . . 9
Table 7. HAL_BLE_LLD_SendPacketWithAck parameters . . . 9
Table 8. HAL_BLE_LLD_ReceivePacket parameters . . . 9
Table 9. HAL_BLE_LLD_ReceivePacketWithAck parameters . . . 10
Table 10. ActionPacket fields description . . . 11
Table 11. ActionTag description . . . 11
Table 12. BLE_LLD_Init parameters . . . 12
Table 13. BLE_LLD_SetReservedArea parameter . . . 12
Table 14. BLE_LLD_MakeActionPacketPending parameter . . . 12
Table 15. RADIO_GetStatus parameter . . . 13
Table 16. BLE_LLD_SetChannel parameters. . . . 13
Table 17. BLE_LLD_SetTxAttributes parameters . . . 13
Table 18. BLE_LLD_SetBackToBackTime parameter . . . 14
Table 19. BLE_LLD_SetTxPower parameter . . . 14
Table 20. Power level values . . . 14
Table 21. BLE_LLD_SetTx_Rx_Phy parameters . . . 15
Table 22. BLE_LLD_StopActivity parameters. . . . 15
Table 23. BLE_LLD_SetEncryptionCount parameters . . . 15
Table 24. BLE_LLD_SetEncryptionAttributes parameters . . . 16
Table 25. BLE_LLD_SetEncryptFlags parameters . . . 16
Table 26. BLE_LLD_StartTone parameters . . . 16
Table 27. Output power level values . . . 17
Table 28. BLE_LLD_StopTone parameters . . . 17
Table 29. Document revision history . . . 18