Application note

Getting started with Dynamic-concurrent mode BLE / Zigbee® on STM32WB Series microcontrollers

Introduction

This document gives an overview of the Dynamic-concurrent mode Bluetooth® Low Energy (BLE) / Zigbee® on STM32WB Series microcontrollers.

The STM32WB Series microcontrollers support Bluetooth® 5.0 and IEEE 802.15.4 wireless standards.

Some use cases require a Dynamic-concurrent mode, with the ability to control the Zigbee® or Thread® network through a Bluetooth® Low Energy device. The dynamic device must be able to handle both protocols at any one time, using a radio time sharing scheme.

1 General information

This document applies to the STM32WB Series dual-core Arm®-based Series 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**

Acronym	Defintion
API	Application programming interface
APS	Application support sub-layer
BDB	Base device behavior
BLE	Bluetooth® Low Energy
CI	Connection interval
GAP	Generic application profile
GATT	Generic attribute profile
HAL	Hardware abstraction layer
IAS	Intruder alarm system
IoT	Internet of things
IPCC	Inter-processor communication controller
MAC	Media access control
NVM	Non volatile memory
PAN	Personal area network
RTSM	Radio Time Sharing Manager
SED	Sleepy end device
ZCL	Zigbee® cluster library
ZDO	Zigbee® device object

1.2 Reference documents

Table 2. Reference documents

Reference	Document name
[1]	Building Wireless Application with STM32WB microcontroller series (AN5289)
[2]	Zigbee® persistent data management non-volatile memory for STM32WB Series (AN5492)
[3]	Creating Manufacture Specific Zigbee® Clusters for STM32WB (AN5491)
[4]	Using Zigbee® Cluster Templates for STM32WB (AN5498)
[5]	ZSDK API for Zigbee® on STM32WB (AN5500)
[6]	STM32WB Zigbee® Getting Started (AN5506)
[7]	STM32CubeWB Nucleo demonstration firmware (UM2551)
[8]	STM32WB Bluetooth® Low Energy wireless Interface (AN5270)
[9]	stm32wb-ble-stack-programming-guidelines-stmicroelectronics.pdf (PM071)

2 Dynamic mode introduction

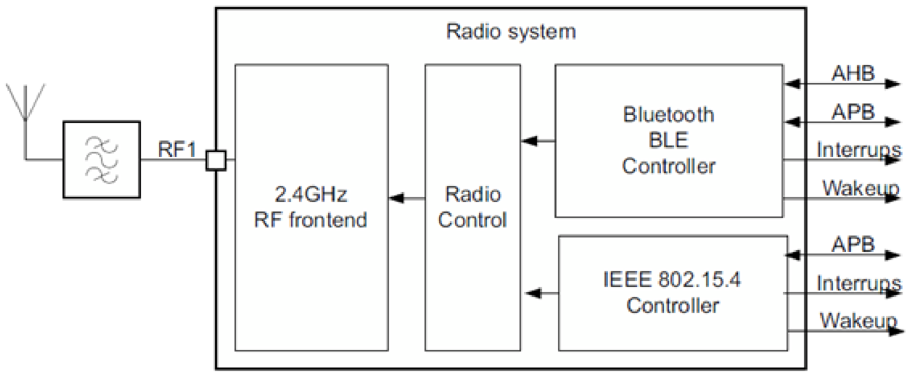
STM32WB Series microcontrollers are dual-core, multi-protocol wireless microcontrollers based on an Arm® Cortex®-M4 core running at 64 MHz (application processor) and an Arm® Cortex®-M0+ core at 32 MHz (network processor),

This microcontrollers supports BLE network, multiple profiles, and has flexibility to integrate proprietary BLE stacks.

The generic IEEE 802.15.4 MAC layer ensures that the STM32WB Series is able to run proprietary protocols or stacks including Zigbee® and Thread® low-power mesh networking protocols, giving designers even more options for connecting devices to the internet of things (IoT).

To support these features, in the internal architecture, each protocol (BLE or 802.15.4) shares the same radio peripheral as illustrated in the figure below. The radio peripheral is dynamically configured in either 802.15.4 mode or BLE mode at any one time.

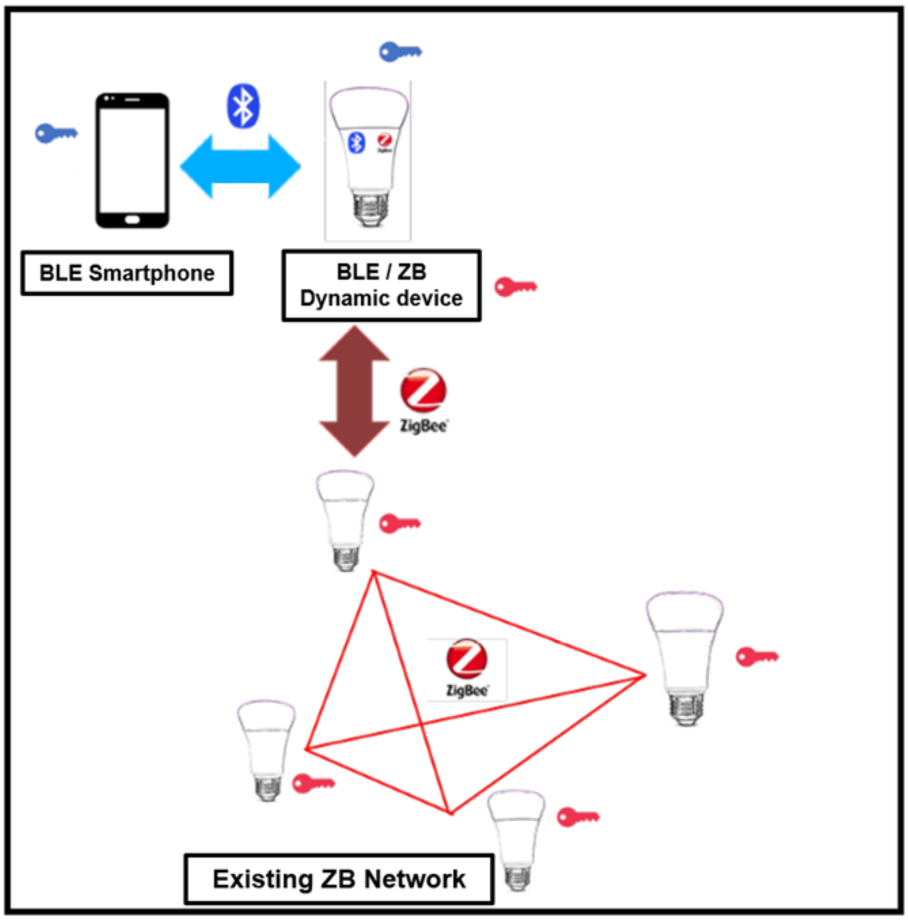
Figure 1. Radio peripheral



2.1 Dynamic mode use case

Some use cases require a Dynamic mode, with the ability to control their Zigbee® or Thread® network through a BLE device and thus require a dynamic device able to handle both protocols at any one time, using a radio time sharing scheme.

Figure 2. **Dynamic mode use case**



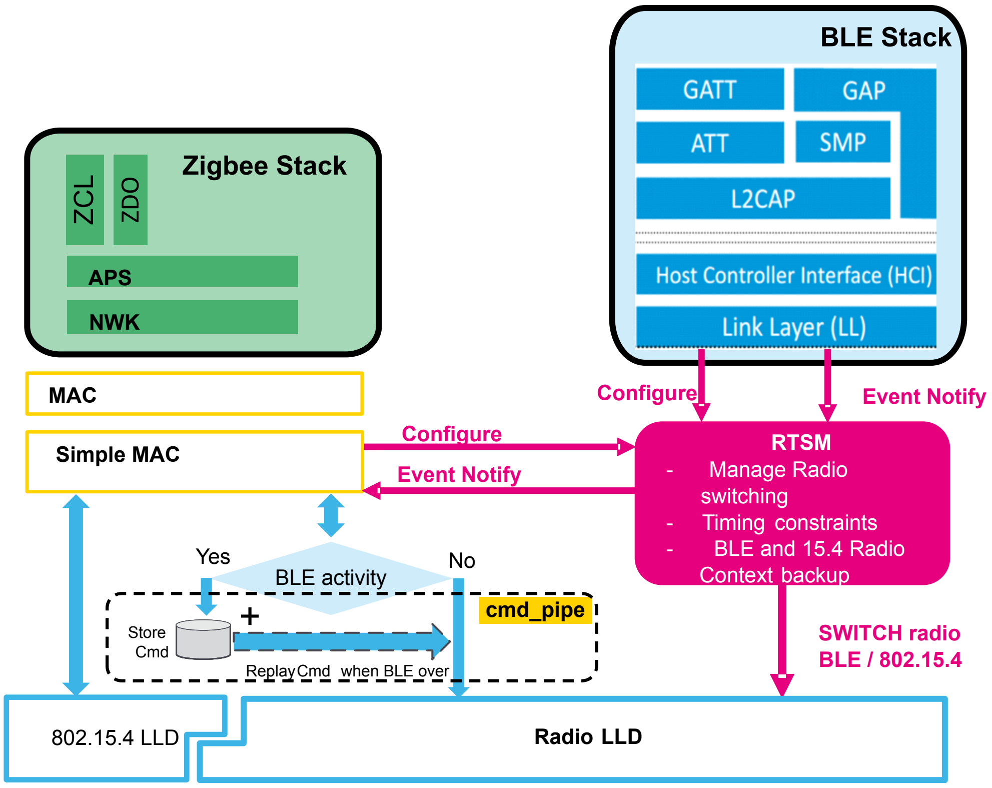
3 RTSM architecture

3.1 RTSM block diagram

The RTSM implementation enables radio switching between BLE and 802.15.4, thus BLE and Zigbee® stacks are able to run in parallel on the same radio. The implementation is illustrated in the diagram below.

The Zigbee® and BLE stacks are described in Section 4 .

Figure 3. RTSM block diagram



3.2 RTSM description

The RTSM is implemented on the Cortex®-M0+ to enable radio switching between BLE and 802.15.4, thus allowing to run the BLE and Zigbee® stacks in parallel using the same radio.

- On the Zigbee® side, the RTSM is integrated at SimpleMAC level, the layer which interfaces between MAC layer and the 802.15.4 / radio LLD.
- On the BLE side, the RTSM is integrated between the link layer and radio LLD.

3.2.1 RTSM features

The RTSM implements the Dynamic mode key features:

- Manages radio switching between BLE and 802.15.4 modes
- Manages the tight timing constraints required by a BLE connection, specifically the connection interval, in order to keep connection alive.
- Backs up and restores the radio context associated with BLE and 802.15.4
- Gives full priority to BLE over 802.15.4 in case BLE needs high bandwidth.

3.2.2 RTSM scheduling

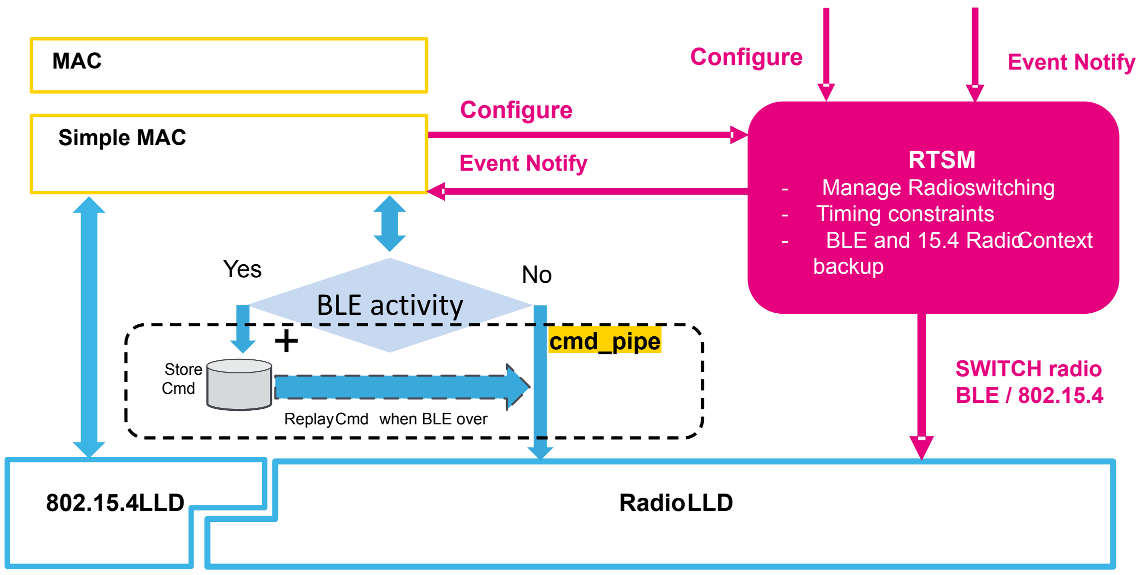
The RTSM behavior is scheduled by the BLE connection:

- BLE programs an RTSM interrupt at each connection interval (1 ms before)
  – This interrupt is managed by a “CompC SfTimer” interrupt
  – When getting this interrupt, RTSM saves the current 802.15.4 radio context and SPI registers, then switches to a BLE radio context
  – BLE is now owner of the radio
- At the end of the BLE connection event, BLE notifies to RTSM the end of connection event and programs the time of next connection interval
  – The RTSM programs accordingly the next CompC interrupt
  – The RTSM switches back radio and SPI registers to 802.15.4 context
  – The 802.15.4 is now owner of the radio until the next RTSM interrupt
- The RTSM also manages the “SfTimer” wraparound which occurs every 3 minutes

3.2.3 Cmd_pipe module

Together with the RTSM module, a new module called cmd_pipe (see figure below) has been implemented to manage the MAC to radio interface depending on the radio state:

Figure 4. **Cmd_pipe module**



- MAC commands are either sent directly to the 802.15.4 controller (through LLD) or sent to the radio peripheral. "`cmd_pipe`" only deals with “MAC to radio” commands.
- If the 802.15.4 (Zigbee® mode) has ownership of the radio, all “MAC to radio” commands are issued directly to the radio.
- If the BLE has ownership of the radio, all “MAC to radio” commands are temporarily stored in the `cmd_pipe` buffer, waiting for the radio to be allocated once again to 802.15.4. Once the ownership of the radio is given to the 802.15.4, the commands are safely sent to the radio. Other MAC commands which do not deal directly with the radio function are normally executed.
- Most of these commands are configuration commands, usually used during startup of the Zigbee® stack (`SetChannel`, `SetPower`).

- The commands available at runtime are defined in the table below.

Table 3. Zigbee®runtime commands

Command	Execution details
Transmit	Executed when the radio is available to the 802.15.4
Sleep	Executed when the radio is available to the 802.15.4
Wake	Executed when the radio is available to the 802.15.4
Energy detection	This command returns an error code if radio not available

- Some commands may return an error code, while others do not:
  - Non-void commands return an ERROR code if radio is not available.
  - In the specific case of transmit commands, no error is returned, Tx is executed when the radio is available again.
- The current size of the `cmd_pipe` allows for 30 pending commands.
  - During the debug phase, a maximum depth of eight pending commands has been seen in the `cmd-pipe` , mainly during Zigbee® initialization and join procedure.
  - In case of a `cmd_pipe` overflow, the whole `cmd_pipe` is flushed and restarted, and all pending commands are lost.

3.2.4 BLE & Zigbee® stack behavior in Dynamic mode

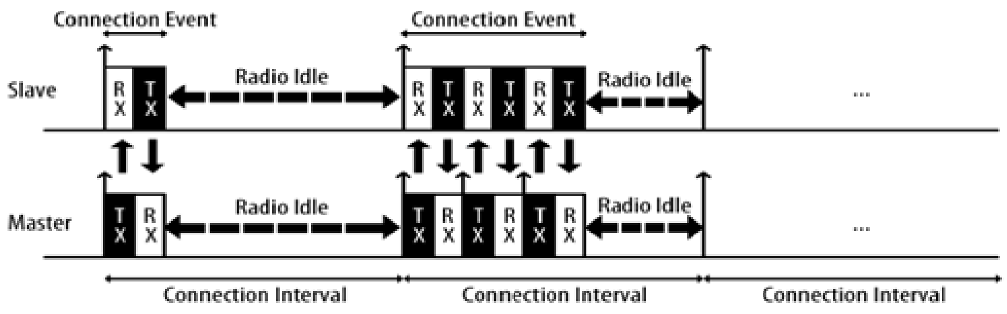
The BLE and Zigbee® stacks are not aware of the radio switching mechanism, and behave as though the radio was fully dedicated to their own stack.

The BLE has tighter time constraints than Zigbee®. Therefore the BLE must be given a higher priority on the radio access than 802.15.4 based protocols.

BLE stack

On the BLE side, the connection event must be precisely scheduled exactly at the expected connection interval (CI). The RTSM takes care of this precise timing. Both the connection interval and connection event are defined in the following list and illustrated in the figure below:

Figure 5. BLE connection events and connection intervals



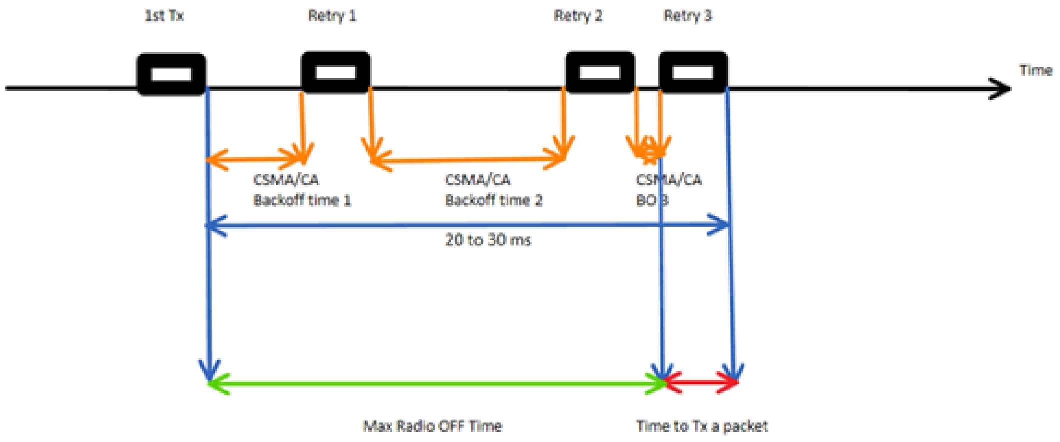
- Connection interval: time between two connection events (from 7,5 ms to 4 s).
- Connection events: consecutive Rx/Tx switch between master and slave in a connection interval.
- The “Radio Idle” slots defined above is assigned by RTSM to 802.15.4, allowing Zigbee® operation.

Zigbee® stack

On Zigbee® side, Tx and Rx occur at any time depending on the local and remote Zigbee® device needs.

- For Zigbee®/MAC Tx requests:
  - If the radio is granted to the 802.15.4, it transmit as usual.
  - If the radio is granted to BLE, all radio commands (including Tx) are stored in the `cmd_pipe`.
  - When the radio is back to 802.15.4, all pending commands (including Tx) are sent.
- For Zigbee®/MAC Rx events:
  - All Rx events are managed under Rx interrupt. Such an interrupt is disabled while radio is in BLE mode. Thanks to the CSMA/CA retry mechanism (up to 3 retries), these Rx events are rescheduled few milliseconds later.
  - Internal investigations have shown that the max time without radio (assigned to BLE) must be less than 16 ms to prevent packet loss.

Figure 6. Zigbee® Rx/Tx behavior



4 Dynamic NVM feature

4.1 Dynamic NVM overview general overview

The BLE/Zigbee® Dynamic mode also provides NVM features. This mode is able to save and restore the Zigbee® state of a dynamic device in Flash. The NVM features support all primary Flash operations (read/write/erase) while the BLE is running.

To maintain the integrity of the BLE behavior when a Flash operation occurs (for example when there is a write or an erase operation), the Cortex®-M4 is able to perform two commands on the Cortex®-M0 (hosting the BLE/ Zigbee® stack & RTSM). Those commands are directly handled by the RTSM. This is due to the fact that the Flash is blocked, and no fetch instruction is possible and is defined in the following list:

- Request the amount of time (µs) available before the next BLE event. If in BLE mode, time is returned as ‘0’.
- Being notified when the next 802.15.4 event is to take place.

4.2 BLE/Zigbee® Dynamic mode Flash operation

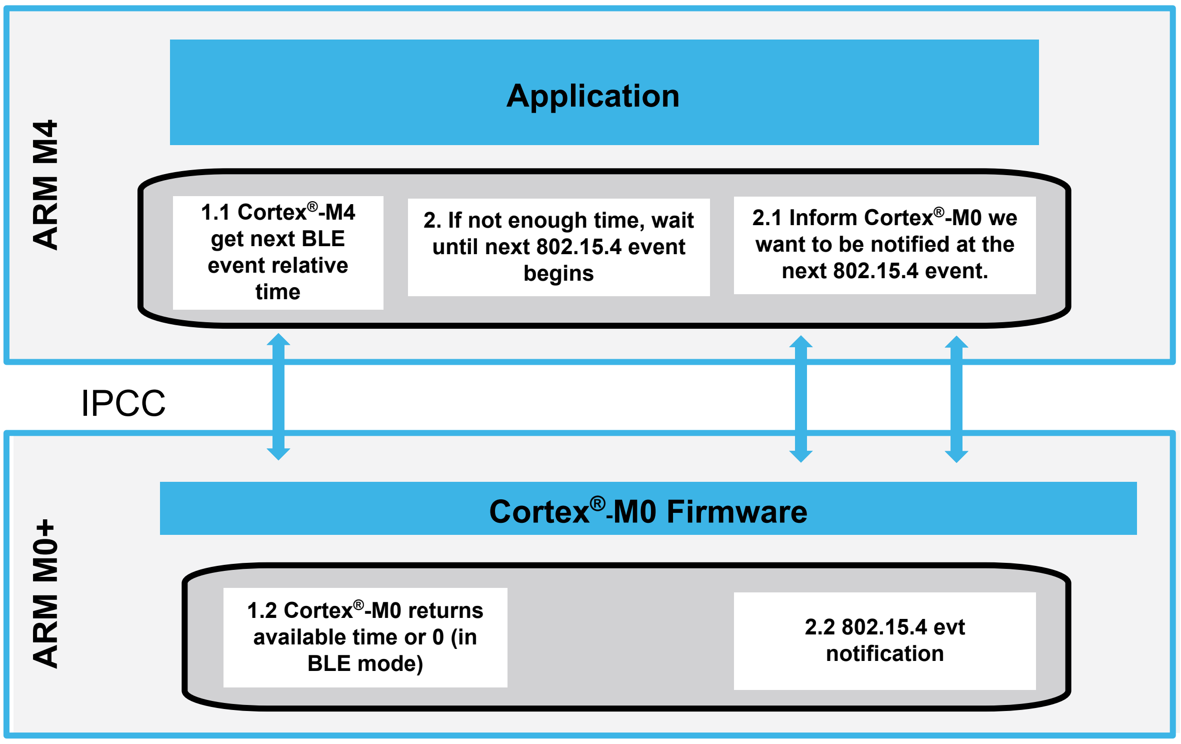
The following list outlines the behavior of the Flash operation:

- A Flash write lasts 5 µs
- A Flash erase operation lasts 20 ms for a whole page.

The Cortex®-M4 software organizes the Flash operation sequences based on these IPCC commands. A Dynamic mode Flash driver is available with the BLE/Zigbee® dynamic NVM application (see Section 8.4.4).

For an illustration of the Dynamic Flash operation, see the figure below.

Figure 7. Flash operation process illustration


ARM M4
Application
1.1 Cortex®-M4 get next BLE event relative time
2. If not enough time, wait until next 802.15.4 event begins
2.1 Inform Cortex®-M0 we want to be notified at the next 802.15.4 event.
IPCC
ARM M0+
Cortex®-M0 Firmware
1.2 Cortex®-M0 returns available time or 0 (in BLE mode)
2.2 802.15.4 evt notification

5 Zigbee® architecture

5.1 Zigbee® overview

Zigbee® is an IEEE 802.15.4 based communication protocol used to create wireless personal area networks (WPAN). The aim is to provide a simple networking layer and standard application profiles that are used to create interoperable solutions, with low-power and low-bandwidth constraint.

It concerns, among other things:

- Home automation
- Industrial control systems
- Building automation, HVAC control
- Medical data collection & monitoring
- Wireless sensor networks.

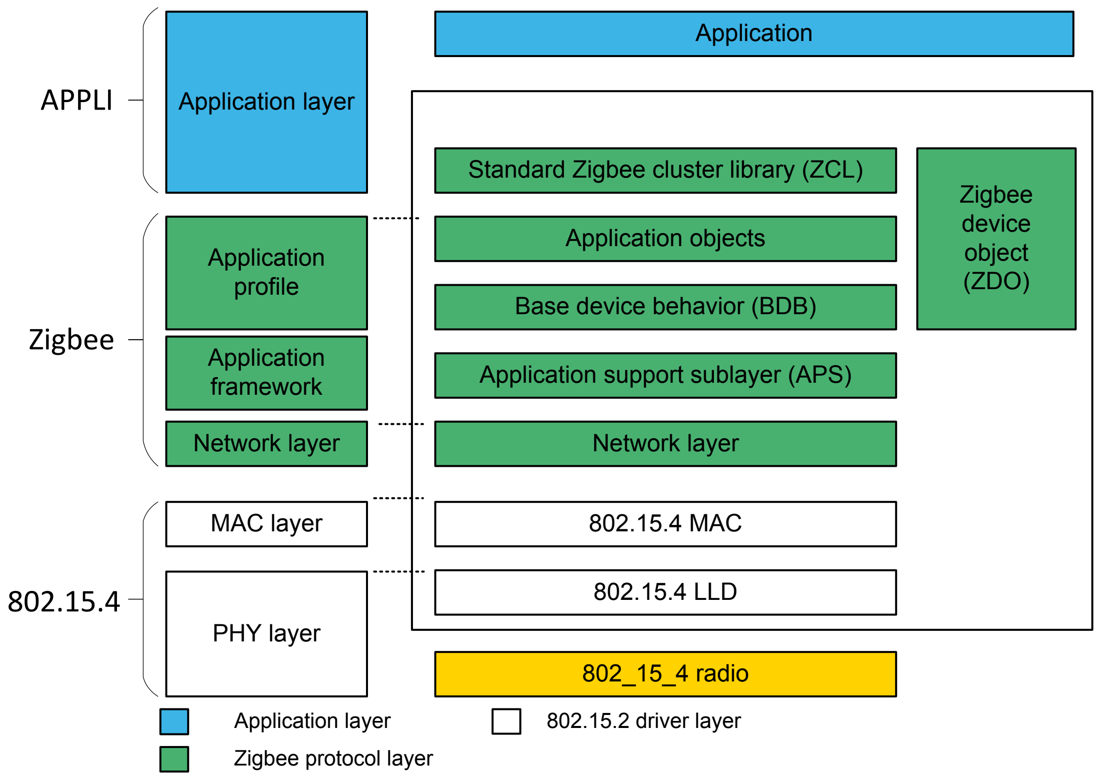
The throughput is 250 kbps at a frequency of 2.4 GHz with a typical range of 10-20 meters.

5.2 Zigbee® stack layers

As described above, Zigbee® is built on top of the IEEE 802.15.4 standard interface. Zigbee® provides routing and multi-hop functions to the packet-based radio protocol. It is built on top of two layers specified by 802.15.4: the physical (PHY) and MAC layers.

The figure below describes the main components of a Zigbee® stack (green blocks), and the way it interacts with IEEE 802.15.4 and general application layer.

Figure 8. Zigbee® stack description



6 BLE architecture

6.1 BLE overview

The STM32WB Series BLE architecture separates the BLE profiles and application; running the application on the Cortex®-M4, with the BLE stack residing in the Cortex®-M0+.

The BLE stack handles the link layer, the generic attribute profile (GATT) and generic access profile (GAP) layers. The link layer directly interfaces with the physical 2.4 GHz radio.

The theoretical BLE 5.0 mode throughput value of 1 or 2 Mbps is reached based on the chosen configuration and a typical range of 10 meters.

6.2 BLE stack layers

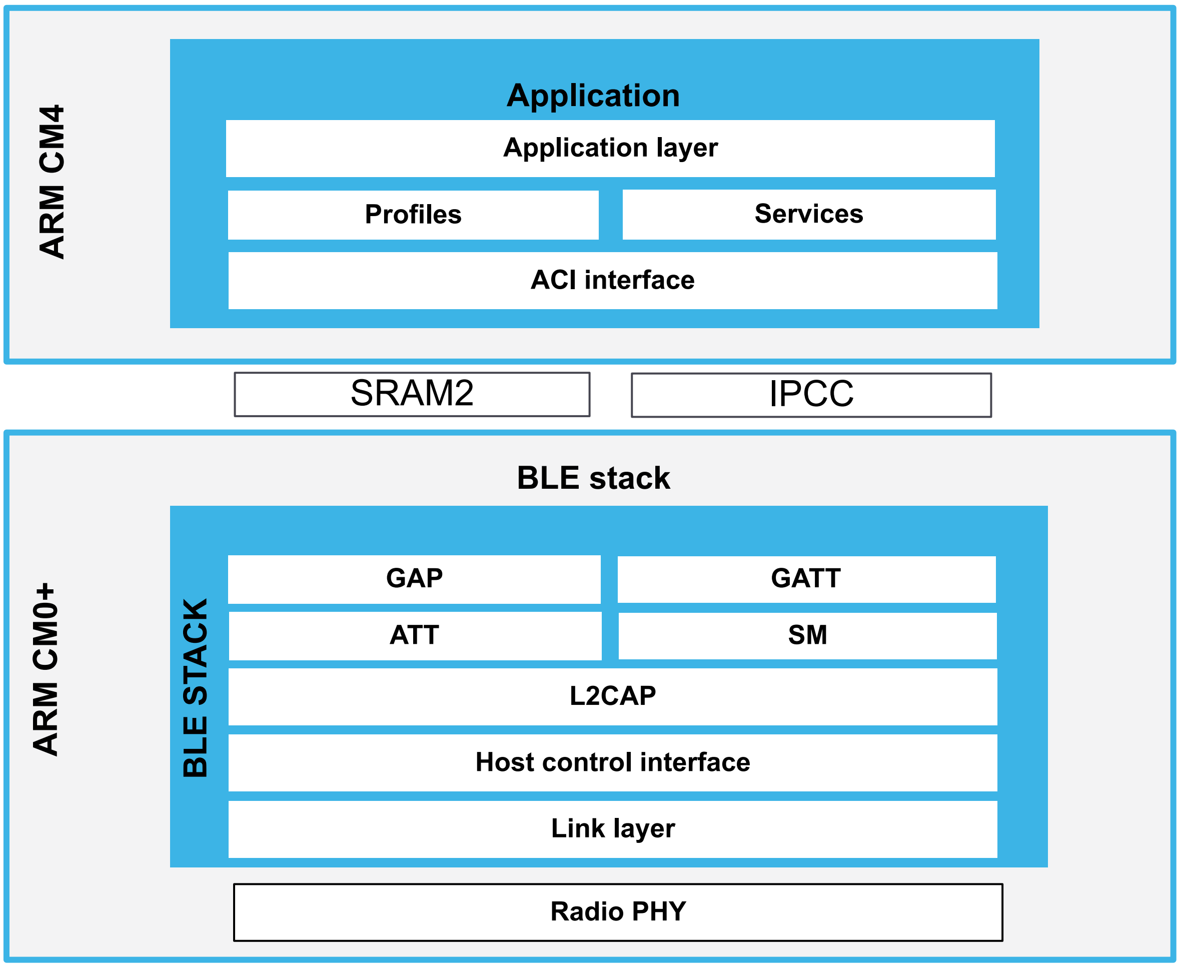
The application is managed by the Cortex®-M4 core as follows:

- Collects & computes the data to be transferred over BLE.
- To transfer data, using the BLE stack services and capabilities.

The BLE stack is managed by the Cortex®-M0+ core as follows:

- The communication with the application layer takes place over the GATT profile
- Implements the LE controller & LE host through HCI protocol
- The "Link Layer" manages all radio PHY layer interaction

Figure 9. **BLE stack description**



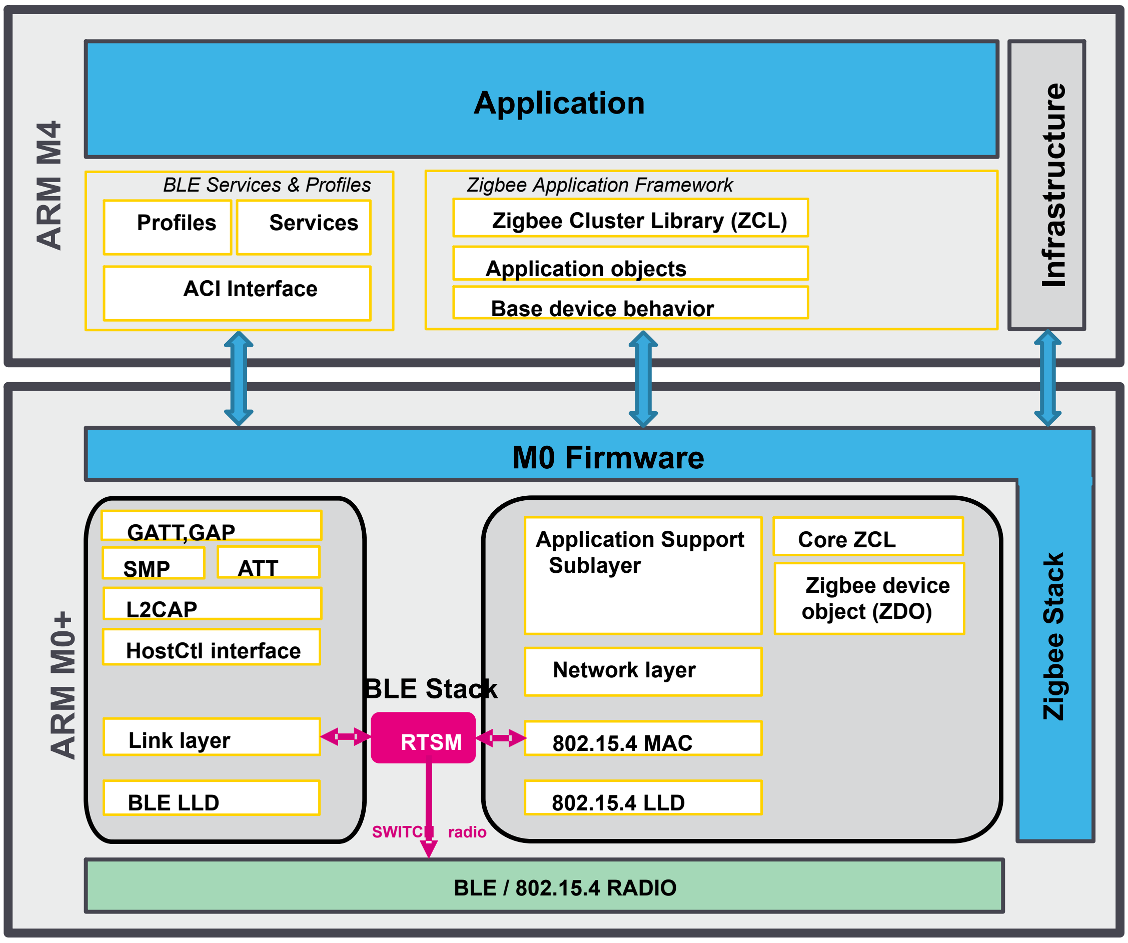
7 BLE/ Zigbee® Dynamic mode on STM32WB Series

7.1 Architecture overview

The figure below gives an overview of the overall architecture. It shows in particular, the split between the Cortex®-M4 and the Cortex®-M0 cores. All the code running on the Cortex®-M0 is delivered as a binary library (refer to Section 7.2 for more details).

The customer has only access to the Cortex®-M4 core and sees the firmware running on the -Cortex®-M0 core as a black box. All the intercommunication between the Cortex®-M4 and Cortex®-M0 is hidden by the framework. Dedicated IPCC channels are allocated for Zigbee® and BLE.

Figure 10. BLE/Zigbee® Dynamic mode architecture



- The Zigbee® stack runs on top of the 802.15.4 MAC layer which itself uses services provided by the 802.15.4 LLD (low level driver) in charge of controlling the radio.
- The BLE stack runs on top of the BLE LLD which controls the radio.
- The RTSM interfaces with both stacks and manages the switch between both radios.

7.2 Dynamic firmware supported

In Dynamic mode, both BLE and Zigbee® stacks run in parallel, enabling the simultaneous use of BLE and Zigbee® applications.

The Dynamic mode firmware includes both BLE and Zigbee® stacks:

- The BLE stack is BLE 5.0 certified.
- Two flavors of the Zigbee® stacks are supported on the STM32WB Series device, FFD (full feature device) and RFD (reduced feature device). These stacks are Zigbee® PRO 2017 (revision 22) certified and are detailed in the table below.

Table 4. **Stack firmware association**

Stacks supported	Firmware associated
Zigbee® FFD + Bluetooth® Low Energy 5.0	`stm32wb5x_BLE_ZigBee_FFD_dynamic_fw.bin`
Zigbee® RFD + Bluetooth® Low Energy 5.0	`stm32wb5x_BLE_ZigBee_RFD_dynamic_fw.bin`

- An FFD is able to take any role in the network which are:
  - a router
  - a coordinator
  - an end device.
- An RFD only supports the end device role. An RFD has a smaller footprint compared to an FFD. When building an application acting as a ‘sleepy end device’, in order to reach optimal low power consumption, the application must be built using the Zigbee® RFD stack.

These binaries are used for Dynamic concurrent mode applications. Examples of such applications are provided in:
`Projects\P-NUCLEO-WB55.Nucleo\Applications\BLE_ZigBee` directory.

Important:
Before running any BLE/Zigbee® application on STM32WB Series, check the proper firmware is downloaded on the Cortex®-M0. If it is not the case, use STM32CubeProgrammer (STM32CubeProg) to load the appropriate binary.
All available BLE/Zigbee® binaries are located under: `/Projects/STM32WB_Copro_Wireless_Binaries/STM32WB5x`.
Refer to `/Projects/STM32WB_Copro_Wireless_Binaries/STM32WB5x/Release_Notes.html` for the detailed procedure on how to change the wireless coprocessor binary.

7.3 Zigbee® clusters supported

The Zigbee® ecosystem available on STM32WB Series supports Zigbee® 3.0.

Zigbee® 3.0 clusters are ZCL 7 compliant.

As a matter of fact, it includes base device behavior (BDB), Zigbee® Green Power and several specific ZCL clusters as listed in the table below:

Table 5. **Supported Zigbee® clusters**

Nb	Cluster ID	Cluster name
1	0x0000	Basic cluster
2	0x0001	Power configuration cluster
3	0x0003	Identify cluster
4	0x0004	Groups cluster
5	0x0005	Scenes cluster
6	0x0006	On/Off cluster
7	0x0008	Level control

Nb	Cluster ID	Cluster name
8	0x000a	Time cluster
9	0x0019	Over-The-Air upgrade clustere
10	0x0020	Poll control cluster
11	0x0021	Green power proxy
12	0x0102	Window Covering cluster
13	0x0202	Fan control cluster
14	0x0204	Thermostat user interface cluster
15	0x0300	Color control cluster
16	0x0301	Ballast configuration cluster
17	0x0400	Illuminance measurement cluster
18	0x0402	Temperature measurement cluster
19	0x0406	Occupancy sensing cluster
20	0x0502	IAS WD cluster
21	0x0b05	Diagnostics cluster
22	0x1000	Touchlink cluster
23	0x0002	Device temperature configuration cluster
24	0x0007	On/Off switch configuration cluster
25	0x0009	Alarms cluster
26	0x000b	RSSI location cluster
27	0x0015	Commissioning cluster
28	0x001a	Power profile cluster
29	0x0024	Nearest gateway cluster
30	0x0101	Door lock cluster
31	0x0200	Pump configuration and control cluster
32	0x0201	Thermostat cluster
33	0x0203	Dehumidification Control cluster
34	0x0401	Illuminance level sensing cluster
35	0x0403	Pressure measurement cluster
36	0x0405	Relative humidity measurement
37	0x0500	IAS Zone cluster
38	0x0501	IAS ACE cluster
39	0x0700	Price cluster
40	0x0701	Demand response and load control cluster
41	0x0702	Metering cluster
42	0x0703	Messaging cluster
43	0x0704	Smart energy tunneling (complex metering)
44	0x0800	Key establishment
45	0x0904	Voice over Zigbee® cluster
46	0x0b01	Meter identification cluster
47	0x0b04	Electrical measurement cluster

- All these 47 clusters are available through the STM32_WPAN middleware. This middleware is common to BLE and Thread®. For specific needs, a customer may create a ‘proprietary’ cluster if needed. Refer to [3] for more details.
- The APIs relative to these clusters are available in the following directory: `\Middlewares\ST\STM32_WPAN\Zigbee\stack\include`.
- By default, all clusters are delivered as a single library. Nevertheless, it is possible to have access to the source code on request.

8 STM32WB Series dynamic application design

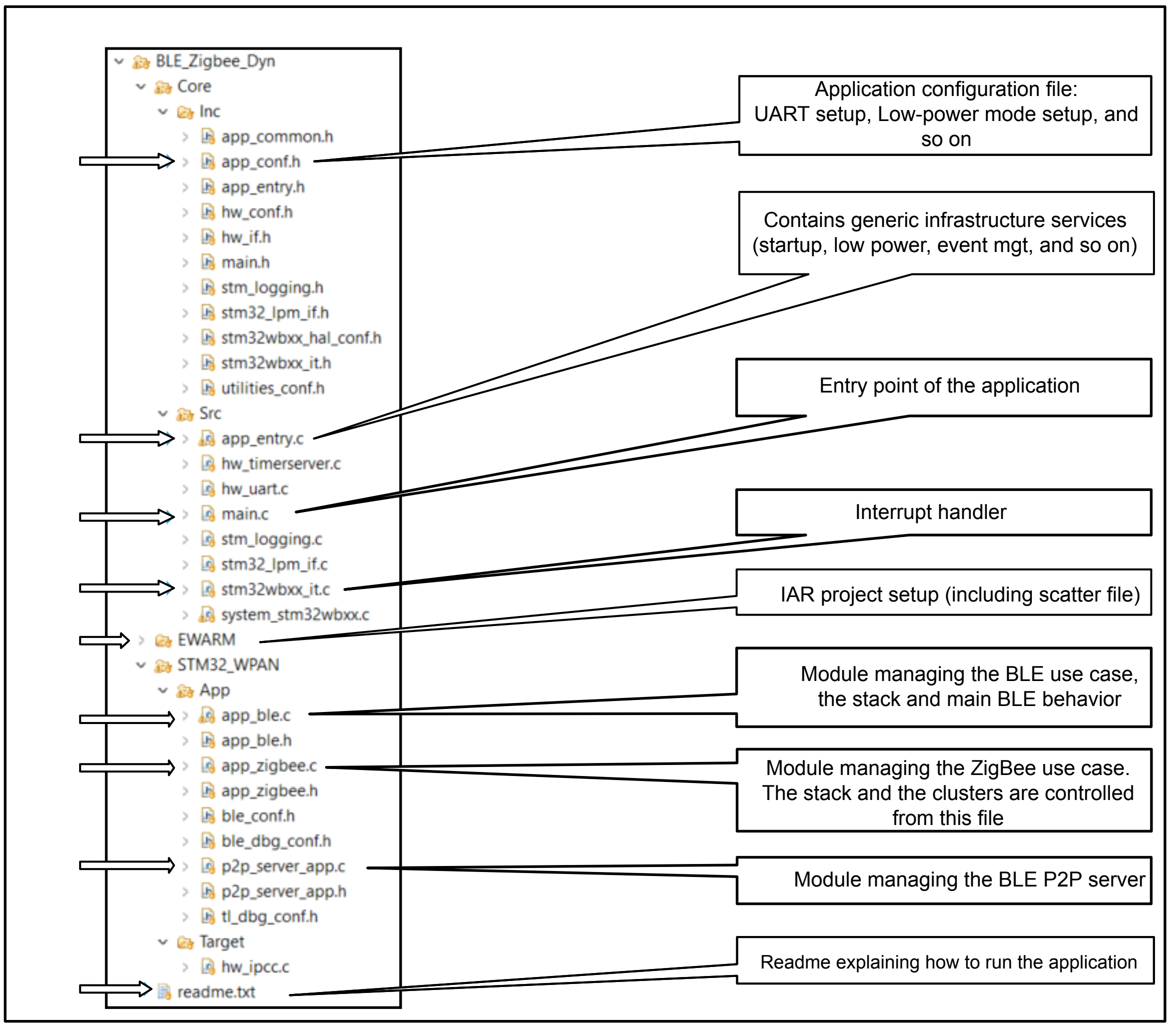
8.1 BLE/Zigbee® dynamic application framework

All projects are built using the same framework. The main App features are defined under: `Projects\Board_X\Applications\BLE_ZigBee\BLE_ZigBee_Dyn\STM32_WPAN\App`.

- the Zigbee® use case is defined and implemented in the `app_zigbee.c` file.
- the BLE use case is defined and implemented in the `app_ble.c` file.
- the BLE P2P sever is defined and implemented in the `p2p_server_app.c` file.

All the other files present in the application projects are mainly used for the global infrastructure management (Interrupt management, IPCC wrapper, system startup and configuration, and so on)

Figure 11. BLE/Zigbee® dynamic application


BLE_Zigbee_Dyn
Core
Inc
app_common.h
app_conf.h
app_entry.h
hw_conf.h
hw_if.h
main.h
stm_logging.h
stm32_lpm_if.h
stm32wbxx_hal_conf.h
stm32wbxx_it.h
utilities_conf.h
Src
app_entry.c
hw_timerserver.c
hw_uart.c
main.c
stm_logging.c
stm32_lpm_if.c
stm32wbxx_it.c
system_stm32wbxx.c
EWARM
STM32_WPAN
App
app_ble.c
app_ble.h
app_zigbee.c
app_zigbee.h
ble_conf.h
ble_dbg_conf.h
p2p_server_app.c
p2p_server_app.h
tl_dbg_conf.h
Target
hw_ipcc.c
readme.txt
Application configuration file: UART setup, Low-power mode setup, and so on
Contains generic infrastructure services (startup, low power, event mgt, and so on)
Entry point of the application
Interrupt handler
IAR project setup (including scatter file)
Module managing the BLE use case, the stack and main BLE behavior
Module managing the ZigBee use case. The stack and the clusters are controlled from this file
Module managing the BLE P2P server
Readme explaining how to run the application

8.2 Zigbee® application framework

For more details about the Zigbee® application framework and architecture, refer to [6].

8.3 BLE application architecture

For more details about the BLE application architecture, refer to [1].

8.4 Dynamic applications available

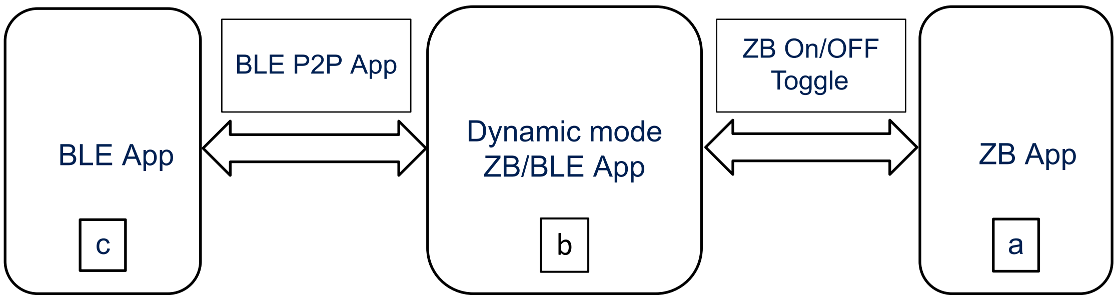
The following four dynamic applications have been implemented:

- BLE/Zigbee dynamic application
- BLE/Zigbee dynamic SED application
- BLE/Zigbee dynamic BLE throughput application
- BLE/Zigbee dynamic NVM application

8.4.1 BLE/Zigbee® dynamic application

This application illustrates the simultaneous BLE and Zigbee® connections on the same device, with a BLE P2P app execution and a Zigbee® toggle On/Off application running simultaneously on the same device.

Figure 12. BLE/Zigbee® dynamic application illustration



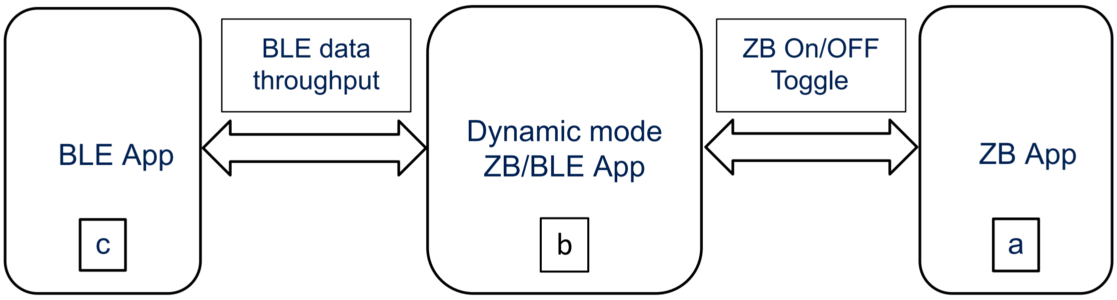
8.4.2 BLE/Zigbee® dynamic SED application

Same as above, but optimized for SED devices (sleepy end devices) and Low-power mode.

8.4.3 BLE/Zigbee® dynamic BLE throughput application

The BLE throughput application has been ported to the dynamic environment, to check the BLE/Zigbee® coexistence when using a high BLE bandwidth.

Figure 13. BLE/Zigbee® dynamic BLE throughput application illustration



8.4.4 BLE/Zigbee® dynamic NVM application

This application illustrates the simultaneous BLE and Zigbee® connections on the same device, with a BLE P2P application execution and a Zigbee® toggle On/Off application running simultaneously on the same device.

Persistent data is used on Zigbee® to save/restore the Zigbee® state while BLE is running.

This is illustrated in Figure 12.

8.5 BLE/Zigbee® dynamic application

8.5.1 Firmware & software requirements

In order to run the BLE/Zigbee® dynamic application, the following binaries and/or software applications must be installed as listed in the table below:

Table 6. **Device firmware specification**

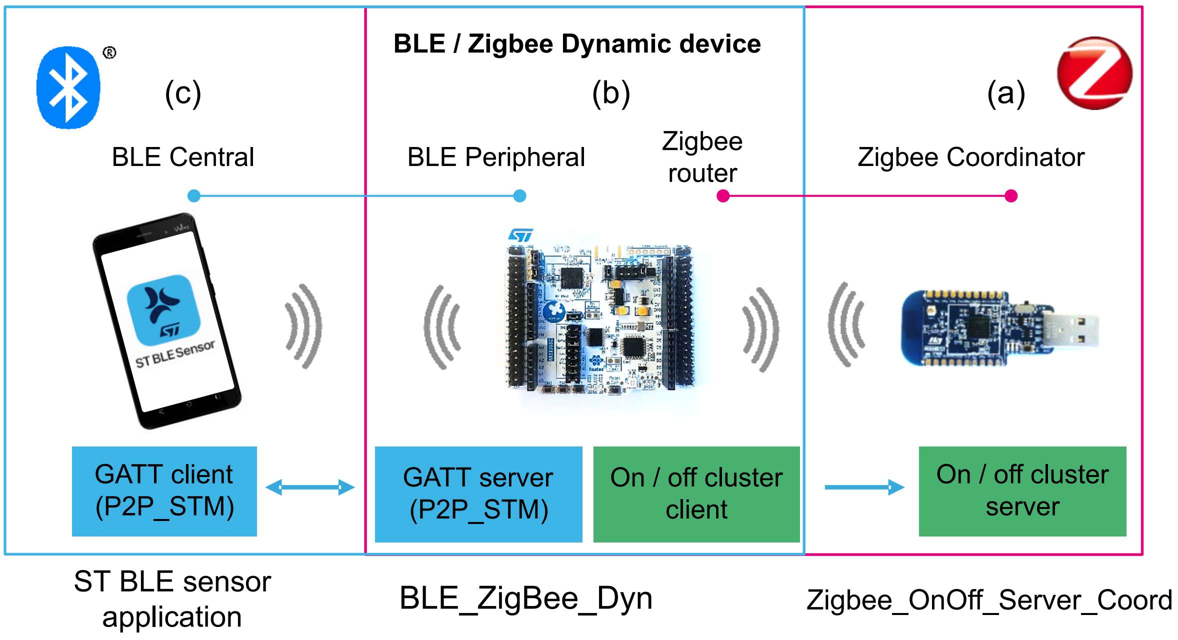
Role	ID	Device	Cortex®-M0 firmware / Cortex®-M4 application
Zigbee® coordinator	(a)	Nucleo board	Cortex®-M0 `stm32wb5x_ZigBee_FFD_fw.bin` Cortex®-M4 `ZigBee_OnOff_Coord.bin`
Dynamic device	(b)	Nucleo board	Cortex®-M0 `stm32wb5x_BLE_ZigBee_FFD_dynamic_fw.bin` Cortex®-M4 ble_zigbee_Dyn.bin
BLE device	(c)	Smartphone (Android™/iOS™)	"ST BLE Sensor" phone application, available on App Store and Google Play
		Nucleo board	Cortex®-M0 `stm32wb55xx_ble_full_host_stack_cut2.1.bin` Cortex®-M4 `BLE_p2pClient.out`

8.5.2 Dynamic application description

The main components of the BLE/Zigbee® dynamic application illustrated in the figure below are the following:

- A Zigbee® coordinator (a) running the On/Off cluster server.
- A BLE/Zigbee® Dynamic device (b) configured as a Zigbee® router running the On/Off cluster client, and a BLE peripheral running a P2P server app.
- A BLE device (c) running the P2P client app 7.

Figure 14. Dynamic application overview



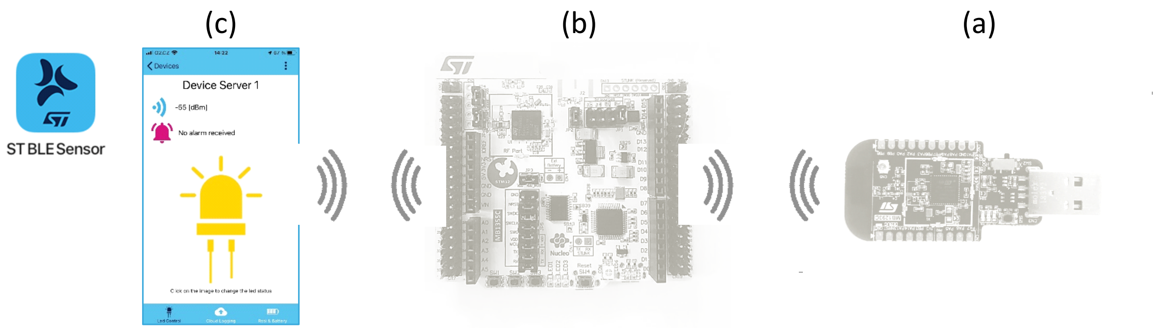
8.5.3 Running dynamic application

This Demo illustrates the simultaneous BLE and Zigbee connections, with the red led toggling on Dynamic device (b), reflecting BLE activity, and the RED LED toggling on the Zigbee® coordinator (a), reflecting Zigbee® activity.

The Dynamic device (b) starts with both BLE and Zigbee® modes activated.

The BLE example implements point-to-point communication using P2P component.

Figure 15. Dynamic application behavior



1. The blue LED indicates that the network is formed (Zigbee® coordinator (a)) and that the network is joined (Dynamic device (b)).
2. The On/Off cluster client (Dynamic device (b)) sends the toggle command to the On/Off cluster server (Zigbee® coordinator (a)) once every sec, the red LED toggles on Zigbee® coordinator (a).
3. Then connect the Dynamic device (b) to the BLE via the "ST BLE Sensor" app (c) and sends a BLE toggle commands through the app, reflected by the red LED state on Dynamic device (b) at the same time.

Dynamic application detailed behavior

1. The Dynamic device (b) (P2P server) starts BLE advertising (green LED toggling) and simultaneously starts to join the Zigbee® network.

2. The node then attaches the existing Zigbee® network (Zigbee® coordinator (a) running `ZigBee_OnOff_Coord.bin` application).The Dynamic device (b) blue LED turns on when the Zigbee® join is successful.
3. When the Zigbee® connection is established, the Zigbee® router (b) sends a Zigbee® ON_OFF toggle every second to the Zigbee® coordinator (a).The red LED toggles on the Zigbee® coordinator (a).
4. The BLE connection starts as follows depending on the device that is connecting. In this case, the BLE device is either a smartphone or a third Nucleo board:
   a. In the case of a smartphone, the "ST BLE Sensor" (smartphone application) scans and connects to the P2P server by selecting the "P2PZBSRV1" entry. The process is as follows:
      i. Once connected, the application starts to search the P2P services and characteristics.
      ii. LED button service, LED characteristic and button characteristic are discovered at this stage (this service is specific to the STMicroelectronics application).
      iii. Pressing the LED button on the application turns the red LED of the Dynamic device (b) on or off .
   b. When using a third Nucleo board, run the BLE_p2pClient.out application, and start the scan by pressing the SW1 button:
      - The blue LED lights up on the BLE device (c).
      - After the scan is complete, the BLE device (c) connects to the P2P server on Dynamic device (b) dynamic node and sends a BLE toggle when pressing SW1.
      - As a result, the red LED toggles on the Dynamic device (b), at each SW1 button press on the BLE device (c).

On BLE disconnection, the Dynamic device (b) restarts advertising by displaying a flashing green LED.

8.5.4 Dynamic Zigbee® routing issue and workaround

During the testing phase, it has been seen that in some cases, the Zigbee® side of the dynamic device may be stalled for some time (1 to 5 minutes), then recovers normal execution.

This section presents an analysis of the issue and describes a workaround.

Description of the Zigbee® routing issue

It has been seen that in some cases, the Zigbee® side of the dynamic device may stalled, and the Zigbee® stops toggling for a period (1 to 5 minutes), then recovers normal execution again.

This issue occurs at any time depending on Zigbee® and BLE timings.

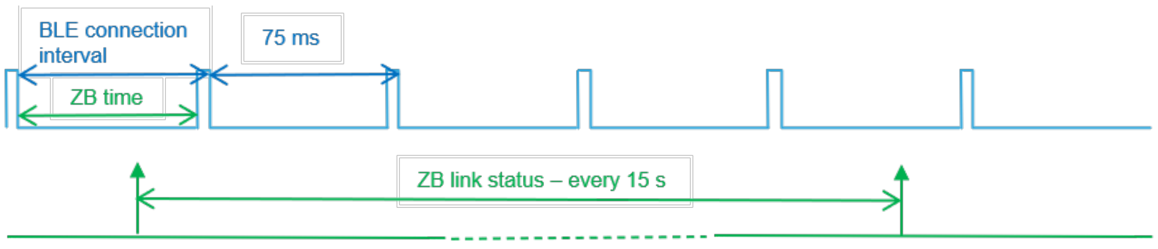
Analysis of Zigbee® routing issue

The issue is due to a timing alignment between the Zigbee® and BLE. and occurs when the Zigbee® link status, sent every 15 s, is received during BLE time. The following figures show what happens during a successful and a failed use case.

- Successful use case:

  In this case the link status occurs during Zigbee time and everything works fine. This is illustrated in Figure 16.

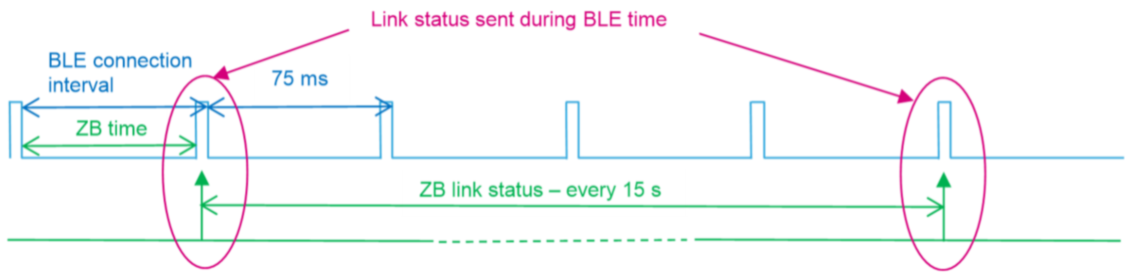
Figure 16. Zigbee® successful dynamic routing



- Failure use case:

  In this case the link status occurs during BLE time and is not seen by the router in Dynamic mode. As there is no retry on the “link status” message, it is definitely lost. This is illustrated in Figure 17.

Figure 17. Zigbee® dynamic routing failure



The Zigbee® link status is sent by the Zigbee® coordinator every 15 seconds, to notify of the Zigbee® link is correct (15 seconds is the default duration defined by Zigbee® spec).

It may happen that in some cases this notification is not seen by the BLE/Zigbee® dynamic device because it is in BLE mode, and is unable to see any 802.15.4 traffic.

As this period (15 s) is an exact multiple of the BLE connection interval (75 ms), and there is no retry on the “link status” message, it is lost. After 4 Link status losses in a row, the Zigbee® connection is seen as invalid, and enters in a lock state.

The Zigbee® lock state may remain for 3 or 5 minutes because the Zigbee® and BLE timings remain aligned, this is the time required for both clocks to slowly drift, and finally allow the Zigbee® link status to be visible in the Zigbee® window. This allows the Zigbee® to recover.

Such an issue may occur more frequently when using BLE applications requiring high bandwidths. In this case, the BLE window is increased while the Zigbee® bandwidth is reduced.

When developing dynamic BLE/Zigbee® applications, special care must be taken in selecting and adjusting BLE parameters (connection interval and CE_{max}) allowing a good performance for both Zigbee® and BLE applications.

Workaround for Zigbee® routing issue

A workaround is implemented, in which the dynamic BLE/Zigbee® device requests a change of its connection interval and is outlined below:

- The dynamic device monitors the AGE of the Zigbee® connection every 15s.
- If AGE equals 4 (meaning 4 link status are lost in a row), the BLE/Zigbee® device requests an update of the BLE connection interval (CI).
- The remote BLE device (master) accepts and updates the connection interval.
- The new CI causes the clocks to drift slowly, and finally reach a point where the “Link status” is received in the Zigbee® window.
- Zigbee® link then recovers normal execution.

As the dynamic BLE/Zigbee® device is a slave BLE device, it is up to the BLE master to accept or reject this change. Testing done with Android™ and iPhone® devices show that both devices accept the request for a connection update:

- Android™ devices usually accept the exact connection update value requested.
- iPhone® devices usually accept the connection update after adjusting the requested value.

After implementation of this workaround, the side effects of this issue are greatly reduced, the Zigbee® connection is no longer blocked for any length of time, and usually recovers within 15 seconds, when receiving the next “Link status”.

8.6 BLE/Zigbee® dynamic SED Application

8.6.1 Firmware & software requirements

To run the BLE/Zigbee® dynamic SED application, the following binaries and/or apps must to be installed :

Table 7. **BLE/Zigbee® dynamic SED application**

Role	Id	Device	M0 Firmware / M4 Application
Zigbee® coordinator	(a)	Nucleo board	Cortex®-M0 `stm32wb5x_ZigBee_FFD_fw.bin` Cortex®-M4 `ZigBee_OnOff_Coord.bin`
Dynamic device	(b)	Nucleo board	Cortex®-M0 `stm32wb5x_BLE_ZigBee_RFD_dynamic_fw.bin` Cortex®-M4 ble_zigbee_SED_Dyn.bin
BLE device	(c)	Smartphone (Android™/iOS™)	"ST BLE Sensor" Phone Application, available on App Store and Google Play
		Nucleo board	Cortex®-M0 `stm32wb55xx_ble_full_host_stack_cut2.1.bin` Cortex®-M4 `BLE_p2pClient.out`

8.6.2 Dynamic SED application description

The behavior of the BLE/Zigbee® dynamic SED application is basically the same as the BLE/Zigbee® dynamic application described in Section 8.5.2 .

The main components of the BLE/Zigbee® dynamic SED application are the following:

- A Zigbee® coordinator (a) runs the On/Off cluster server.
- A BLE/Zigbee® Dynamic device (b) configured as a Zigbee® router runs the On/Off cluster client, and a BLE peripheral running a P2P server app.
- A BLE device (c) runs the P2P client app.

The Firmware differences are listed below (only for dynamic device):

- Cortex®-M0 Firmware built in RFD (reduced function device) instead of FFD.
- Cortex®-M4 Firmware built in SED configuration (sleepy end device).

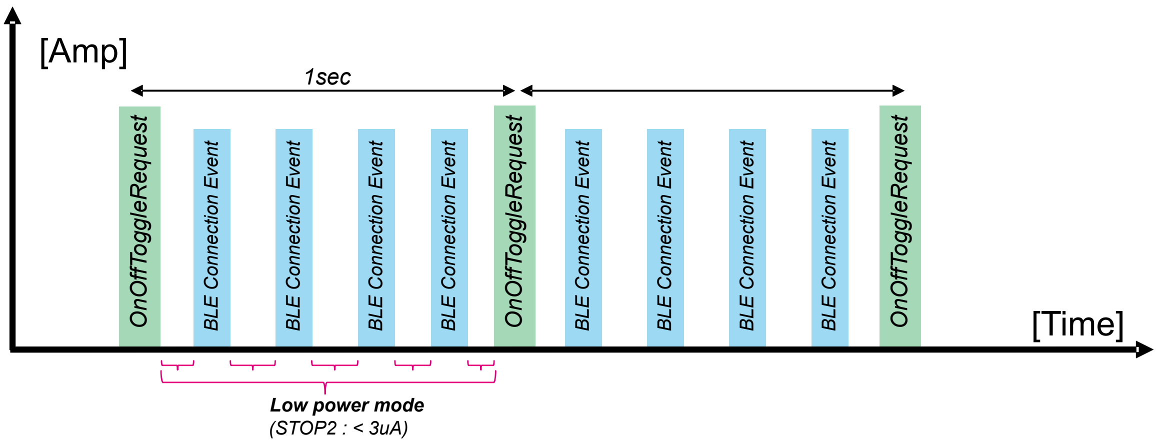
The functional differences are listed below:

- Low- power mode activated
- LEDs and push buttons are disabled
- Debugging mode and debug traces are disabled.

To have the lowest possible power consumption on the SED side, this application is by default compiled with the flag `CFG_FULL_LOW_POWER` set to 1 (in `app_conf.h` file). In this configuration, LEDs and buttons are no longer available, and debug access to the Cortex®-M4 core is also disabled.

In this configuration and using the power shield, it is possible to check that when there is no BLE, nor Zigbee® activity, the SED is able to reach STOP2 Low-power mode with the power consumption dropping as low as 3 µA.

Figure 18. **Dynamic SED application low power**

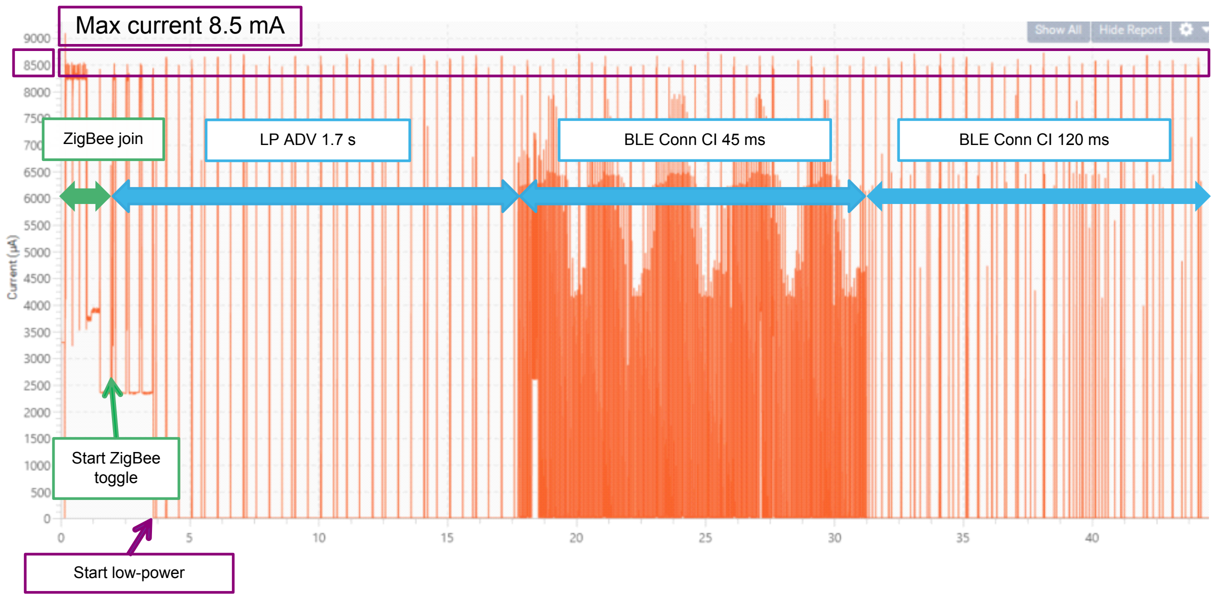


8.6.3 Dynamic SED power figures

Overall power profile

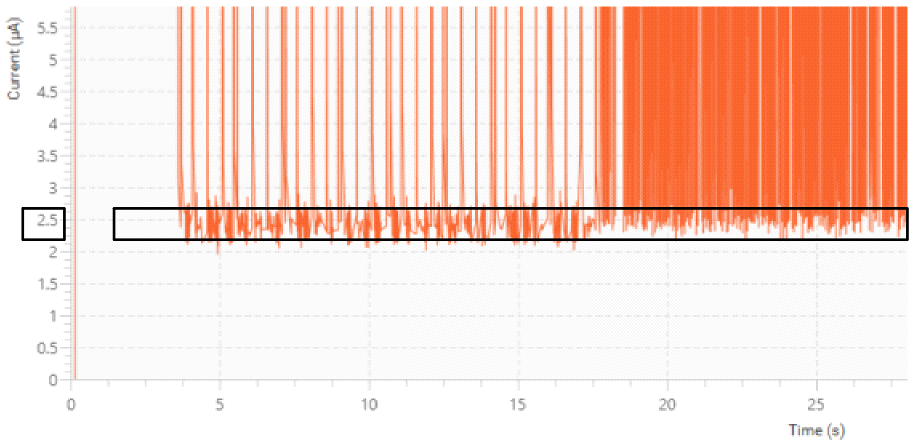
Figure 19 shows the overall power profile during the different phases from startup to BLE connected state.

Figure 19. **Dynamic SED overall power profile**



1. Immediately after startup, the dynamic device starts the "ZB JOIN" process.
2. Simultaneously, the dynamic device starts BLE advertising every 1.7 second.
3. Once Zigbee® join is done, the dynamic device starts sending ZB Toggle OnOff every sec.
4. At this time, the Low-power mode is enabled (around timestamp 5 s).
5. After around 17 secs, the BLE connection is established (connection interval 50 ms).
6. Nine seconds later, a BLE connection update sets the new connection interval to 120 ms.
7. As illustrated Figure 19, the maximum current is 8.5 mA with SMPS enabled.
8. As illustrated in Figure 20 below the low-power edge is at exactly 2,5 µA with SMPS enabled.

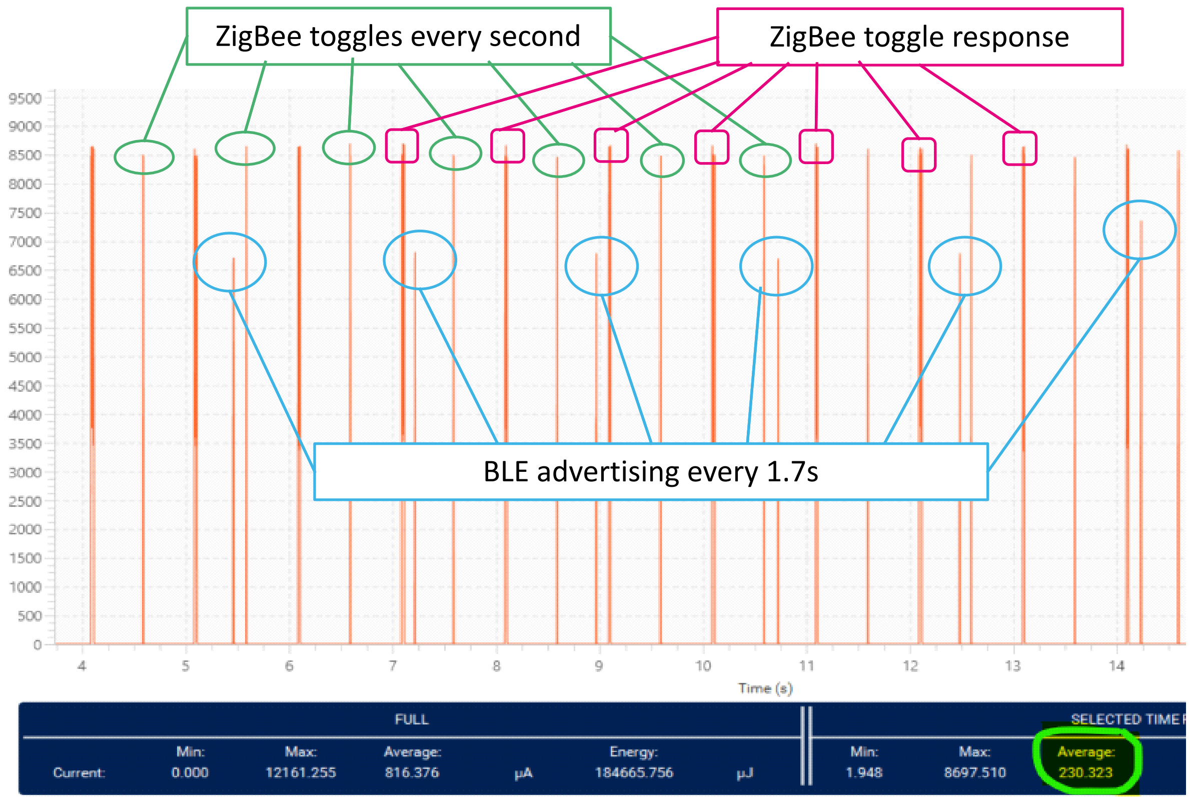
Figure 20. **Low-power edge current with SMPS enabled**



BLE advertising power profile

Figure 21 shows the power profile during the BLE advertising phase. The average power consumption is 460 µA without SMPS, 230 µA with SMPS enabled.

Figure 21. **BLE advertising power profile**



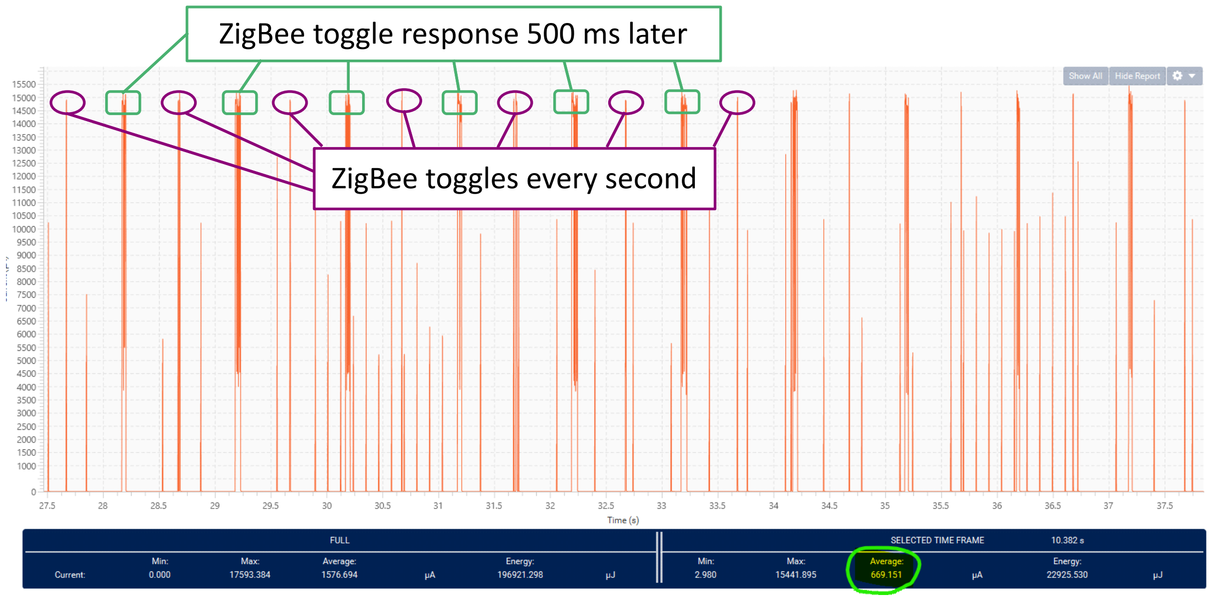
FULL | SELECTED TIME

	Min:	Max:	Average:		Energy:		Min:	Max:	Average:
Current:	0.000	12161.255	816.376	µA	184665.756	µJ	1.948	8697.510	230.323

BLE connected power profile

Figure 22 shows the power profile during the BLE connected state.

Figure 22. **BLE connected power profile**


ZigBee toggle response 500 ms later
ZigBee toggles every second
Show All
Hide Report
Time (s)

	FULL						SELECTED TIME FRAME			10.382 s	
	Min:	Max:	Average:		Energy:		Min:	Max:	Average:		Energy:	
Current:	0.000	17593.384	1576.694	µA	196921.298	µJ	2.980	15441.895	669.151	µA	22925.530	µJ

When BLE is connected with connection intervals of 120 ms, the average power consumption is 660 µA without SMPS, 280 µA with SMPS enabled.

Looking in more detail, the Zigbee® toggle events and the BLE connection events are clearly visible in Figure 23 and Figure 24, with CI to set 120 ms and 45 ms.

Figure 23. **Zigbee® toggle & BLE connection events (CI = 120)**

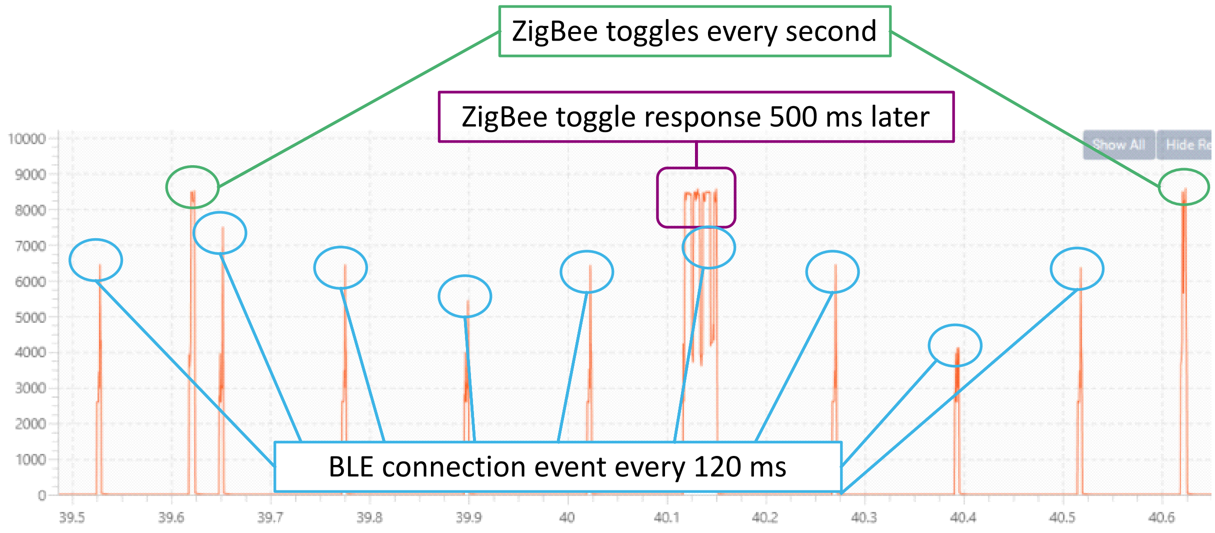
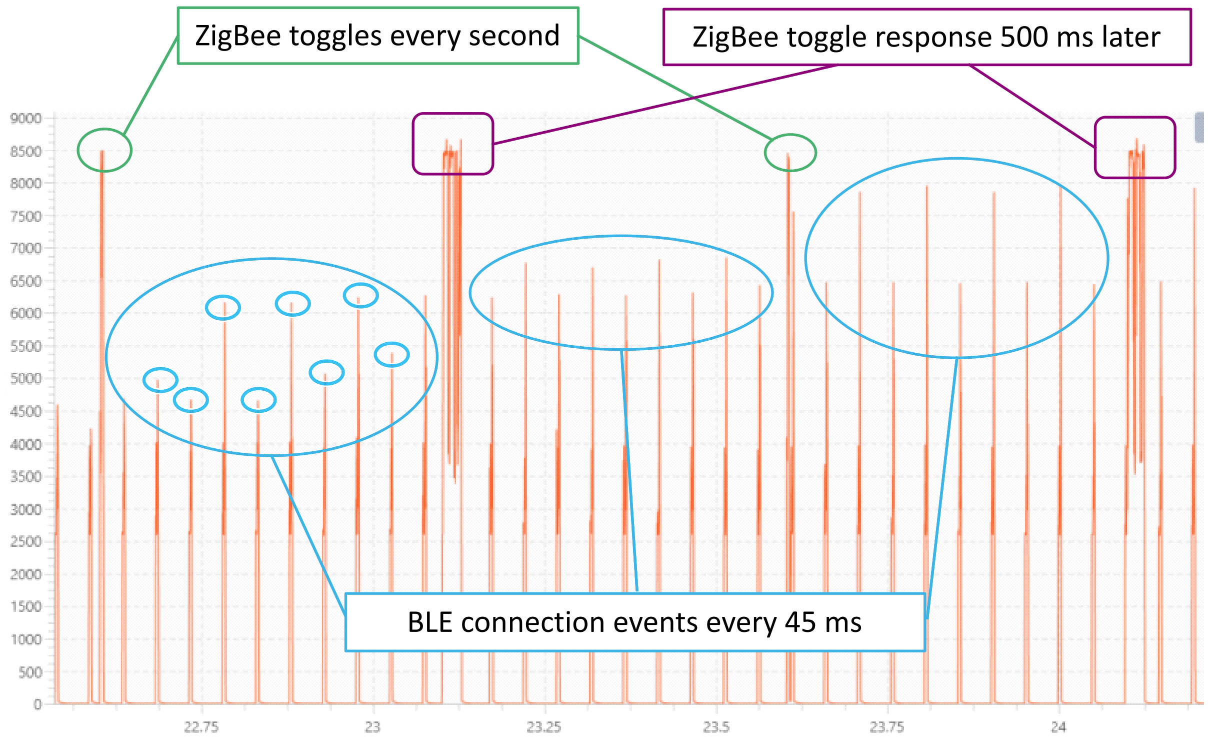


Figure 24. **Zigbee® toggle & BLE connection events (CI = 45)**



Dynamic SED power measurements summary

Table 8 gives a summary of the power consumption in different BLE states, with three different SMPS configurations (SMPS disabled, SMPS enabled – 1V4, SMPS enabled – 1V7).

The Zigbee® activity is always the same, sending a Zigbee® toggle every second, and receiving the toggle response 500 ms later.

This demonstrates that enabling the SMPS gives approximately 50% energy saving.

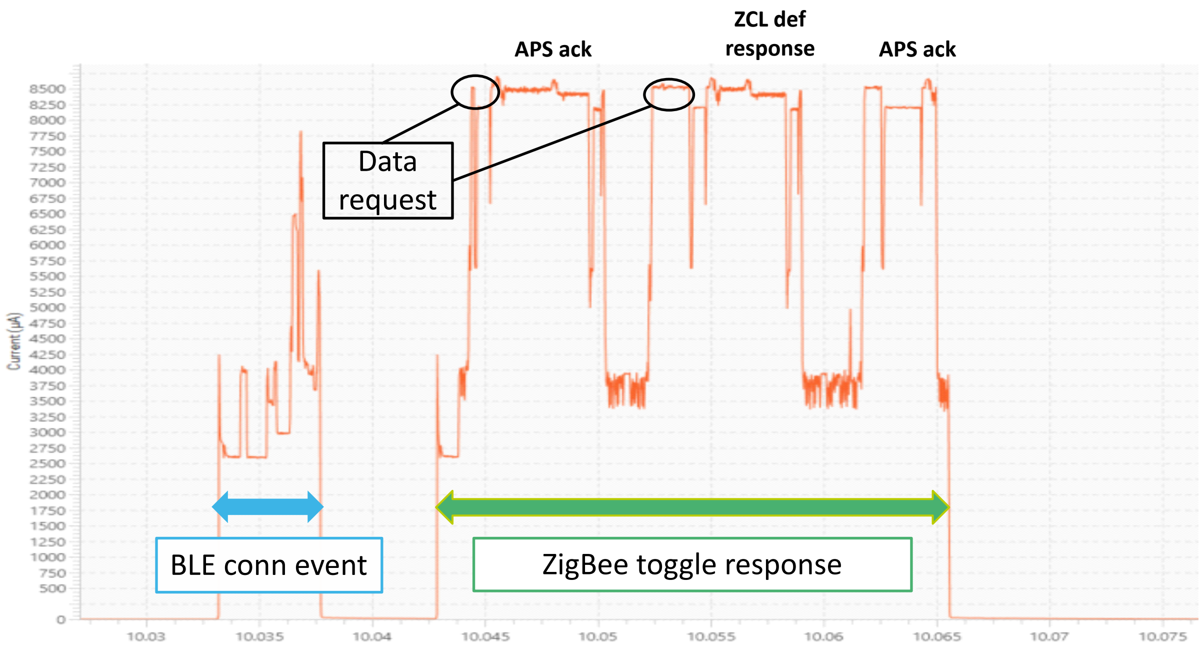
Table 8. **Dynamic SED power measurement**

State	SMPS disabled	SMPS enabled – 1V4	SMPS enabled – 1V7
Advertising 1.7 s	460 µA	230 µA	273 µA
Connected CI=120	660 µA	280 µA	321 µA
Connected CI=45	1.14 mA	579 µA	703 µA
Connected CI=15	NA	1.54 mA	1.72 mA
Max power	15 mA	8.5 mA	10.2 mA
Min power	3 µA	2.5 µA	2.5 µA

BLE/Zigbee® power profile details

Figure 25 shows the details of a BLE connection event and a Zigbee® toggle response.

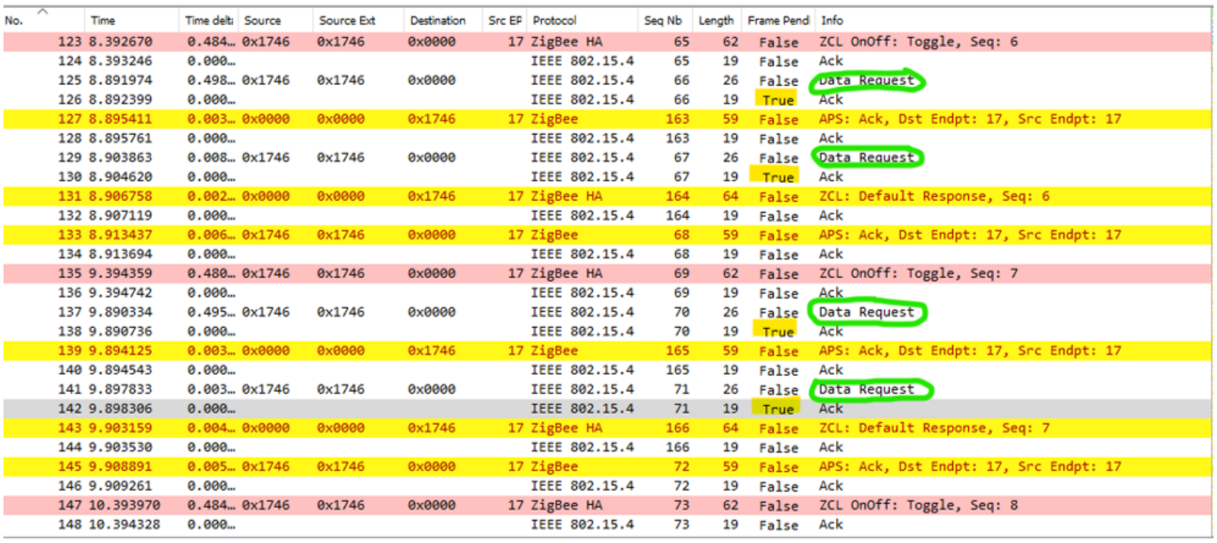
Figure 25. BLE connection events & Zigbee® toggle details



And below is the corresponding Zigbee® trace showing the different steps of the ZB toggle response together with the toggle trace in Figure 26:

- Data request
- APS Ack
- ZCL default response
- APS Ack.

Figure 26. Zigbee® toggle trace



No.	Time	Time delt:	Source	Source Ext	Destination	Src EP	Protocol	Seq Nb	Length	Frame Pend	Info
123	8.392670	0.484…	0x1746	0x1746	0x0000	17	ZigBee HA	65	62	False	ZCL OnOff: Toggle, Seq: 6
124	8.393246	0.000…					IEEE 802.15.4	65	19	False	Ack
125	8.891974	0.498…	0x1746	0x1746	0x0000		IEEE 802.15.4	66	26	False	Data Request
126	8.892399	0.000…					IEEE 802.15.4	66	19	True	Ack
127	8.895411	0.003…	0x0000	0x0000	0x1746	17	ZigBee	163	59	False	APS: Ack, Dst Endpt: 17, Src Endpt: 17
128	8.895761	0.000…					IEEE 802.15.4	163	19	False	Ack
129	8.903863	0.008…	0x1746	0x1746	0x0000		IEEE 802.15.4	67	26	False	Data Request
130	8.904620	0.000…					IEEE 802.15.4	67	19	True	Ack
131	8.906758	0.002…	0x0000	0x0000	0x1746	17	ZigBee HA	164	64	False	ZCL: Default Response, Seq: 6
132	8.907119	0.000…					IEEE 802.15.4	164	19	False	Ack
133	8.913437	0.006…	0x1746	0x1746	0x0000	17	ZigBee	68	59	False	APS: Ack, Dst Endpt: 17, Src Endpt: 17
134	8.913694	0.000…					IEEE 802.15.4	68	19	False	Ack
135	9.394359	0.480…	0x1746	0x1746	0x0000	17	ZigBee HA	69	62	False	ZCL OnOff: Toggle, Seq: 7
136	9.394742	0.000…					IEEE 802.15.4	69	19	False	Ack
137	9.890334	0.495…	0x1746	0x1746	0x0000		IEEE 802.15.4	70	26	False	Data Request
138	9.890736	0.000…					IEEE 802.15.4	70	19	True	Ack
139	9.894125	0.003…	0x0000	0x0000	0x1746	17	ZigBee	165	59	False	APS: Ack, Dst Endpt: 17, Src Endpt: 17
140	9.894543	0.000…					IEEE 802.15.4	165	19	False	Ack
141	9.897833	0.003…	0x1746	0x1746	0x0000		IEEE 802.15.4	71	26	False	Data Request
142	9.898306	0.000…					IEEE 802.15.4	71	19	True	Ack
143	9.903159	0.004…	0x0000	0x0000	0x1746	17	ZigBee HA	166	64	False	ZCL: Default Response, Seq: 7
144	9.903530	0.000…					IEEE 802.15.4	166	19	False	Ack
145	9.908891	0.005…	0x1746	0x1746	0x0000	17	ZigBee	72	59	False	APS: Ack, Dst Endpt: 17, Src Endpt: 17
146	9.909261	0.000…					IEEE 802.15.4	72	19	False	Ack
147	10.393970	0.484…	0x1746	0x1746	0x0000	17	ZigBee HA	73	62	False	ZCL OnOff: Toggle, Seq: 8
148	10.394328	0.000…					IEEE 802.15.4	73	19	False	Ack

8.7 BLE/Zigbee® dynamic BLE throughput application

The BLE throughput application has been ported to the dynamic environment, to check the BLE/Zigbee® coexistence when using a high BLE bandwidth.

8.7.1 Firmware & software requirements

To run the BLE/Zigbee® dynamic data throughput application, the following binaries and/or software apps must be installed as described in the table below:

Table 9. **Firmware & software requirements**

Role	Id	Device	Cortex®-M0 Firmware / Cortex®-M4 Application
Zigbee® coordinator	(a)	Nucleo board	Cortex®-M0 `stm32wb5x_ZigBee_FFD_fw.bin` Cortex®-M4 `ZigBee_OnOff_Coord.bin`
Dynamic device	(b)	Nucleo board	Cortex®-M0 `stm32wb5x_BLE_ZigBee_FFD_dynamic_fw.bin` Cortex®-M4 `BLE_DataThroughput.out`
BLE device	(c)	Smartphone (Android™/iOS™)	"STM32WB Toolbox" phone application, using the data throughput test, available on App Store and Google Play
		Nucleo board	Cortex®-M0 `stm32wb55xx_ble_full_host_stack_cut2.1.bin` Cortex®-M4 `BLE_DataThroughput.out`

8.7.2 Dynamic BLE throughput application overview

The BLE throughput application has been ported to the dynamic environment, in two different configurations:

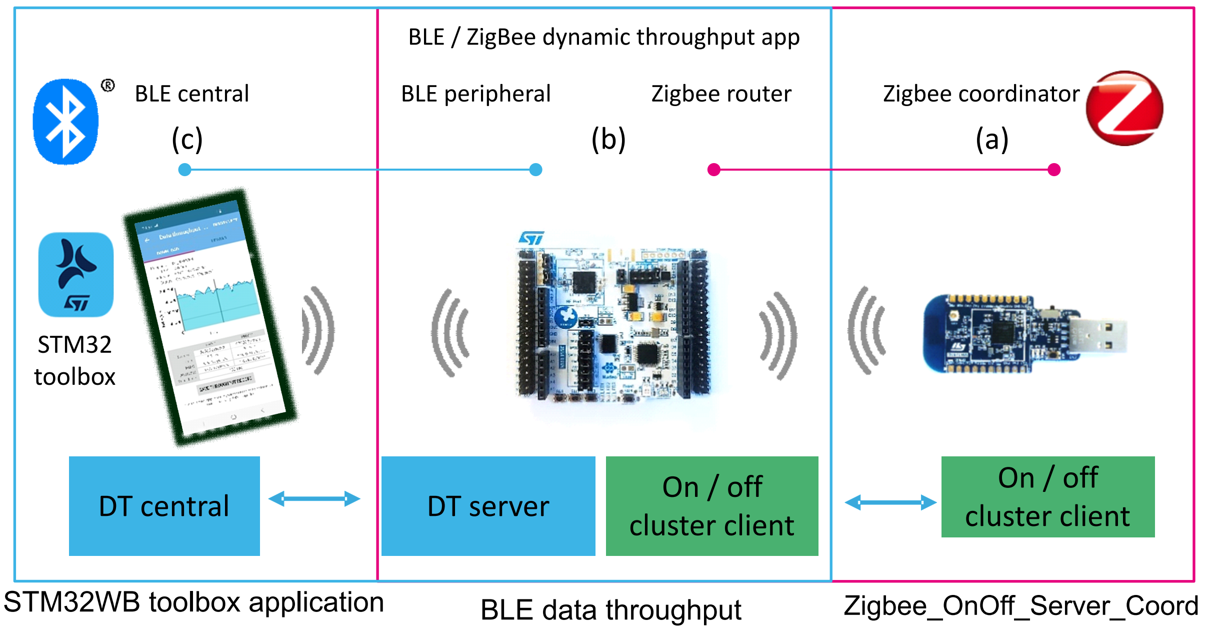
- Configuration 1: two Nucleo boards and the BLE throughput app on mobile phone
  – Zigbee® coordinator (a) and Dynamic device (b) on the Nucleo boards
  – BLE device (c) on smartphone
- Configuration 2: three Nucleo boards with the BLE throughput app on one of the Nucleo boards as illustrated in Figure 12
  – Zigbee® coordinator (a) on board (a)
  – Dynamic device (b) on board (b)
  – BLE device (c) on board (c).

8.7.3 Dynamic BLE throughput application description

The main components of the dynamic BLE throughput application are the following:

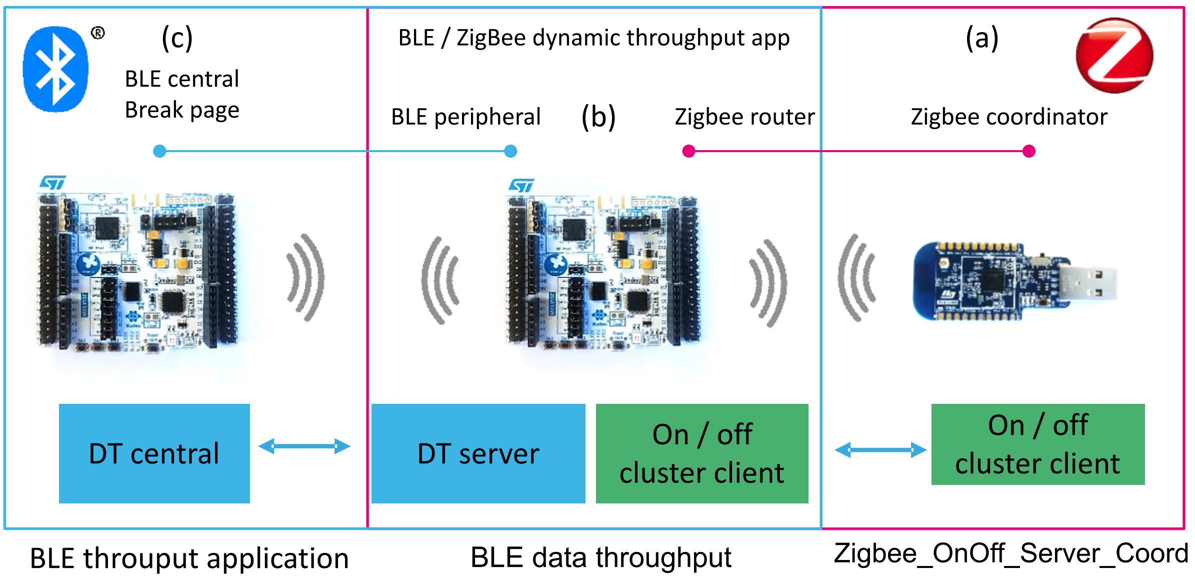
- A Dynamic device (b) running both a Zigbee® router with the On/Off cluster client, and a BLE peripheral running the BLE throughput application.
- A BLE device (c) running the BLE throughput application (smartphone or Nucleo board).
- A Zigbee® coordinator (a) running the On/Off cluster.

Figure 27. Dynamic BLE throughput application with BLE smartphone



The BLE device (c) can alternatively be a smartphone illustrated in Figure 27 or a Nucleo board illustrated in Figure 28 running BLE throughput application. This last configuration gives more flexibility to tune the BLE connection parameters.

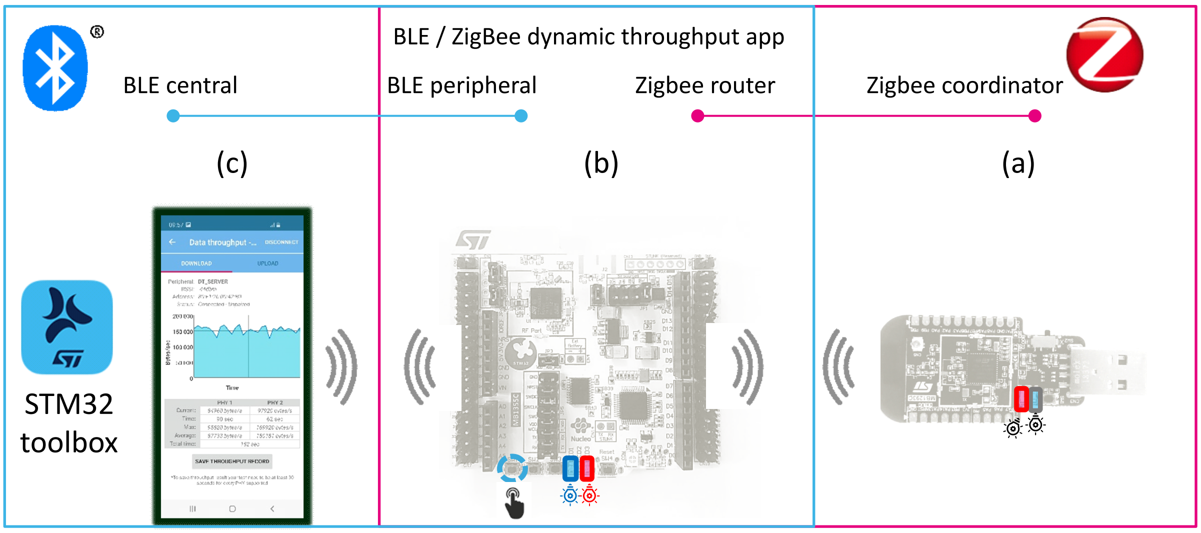
Figure 28. **Dynamic BLE throughput application with BLE on Nucleo board**



8.7.4 Running dynamic BLE throughput application

The dynamic BLE throughput application enables BLE throughput to be measured while running a Zigbee® router with the Zigbee® coordinator client, to check the BLE/Zigbee® coexistence when using a high BLE bandwidth. The application is illustrated in Figure 29.

Figure 29. **Dynamic BLE throughput application behavior**



The Dynamic device (b) starts with both BLE and Zigbee® modes activated. The app implements both the BLE throughput app, and the Zigbee® toggle Zigbee® coordinator.

Blue LED indicates that the network is setup on Zigbee® coordinator (a), and that the network is joined on the dynamic Zigbee® router (b).

On/Off cluster (Dynamic device (b)) sends a toggle command to the On/Off cluster server (Zigbee® coordinator (a)) every 500 ms, a red LED toggles on the Zigbee® coordinator (a).

The Dynamic device (b) advertises as "DT_SERVER" (green LED blinks)

To get BLE throughput measurements using a Smartphone (c), connect it via STM32WB Toolbox App (c).

If using a Nucleo board (c) running a BLE throughput application (DT central), it starts a scan and connect to the "DT_SERVER" Dynamic device (b).

The BLE throughput measurements are done using the 1M or 2M PHY. This setting is selected from the SW2 on the Dynamic device (b).

BLE throughput application detailed behavior

1. The Dynamic device (b) (DT server) starts the BLE advertising (green LED toggling) and simultaneously starts to join the Zigbee® network.
2. The node attaches to the existing Zigbee® network (Zigbee® coordinator (a) running `ZigBee_OnOff_Coord.bin` application).

   The Dynamic device (b) blue LED turns on when the Zigbee® join is successful.
3. When the Zigbee® connection is established, the Zigbee® router (b) sends a ZB ON_OFF toggle to the Zigbee® coordinator (a) every 500 ms.

   The red LED toggles on the Zigbee® coordinator (a).
4. The BLE connection starts as follows, and depends on whether the BLE device is a smartphone or a third Nucleo board:
   – With a smartphone, use the STM32WB toolbox app (select “Data throughput” in the list) to scan and connect to the DT server by selecting "DT_SERVER" entry:
      a. On the next screen, select the “Downlink” or “Uplink” test.
      b. To run the “Downlink” test, connect, and start the transfers by pressing the SW1 button on the Dynamic device (b). The results are displayed on the smartphone. The physical link is set by default to 1M:
         • The test is stopped by pressing again the SW1 button.
         • To change PHY to 2M, press the SW2 button. Restart and stop the test with 2M using SW1 button again.
      c. To run the “Uplink” test, connect, and start the transfers by pressing the SW1 button on the smartphone app. The transfer results are displayed on the smartphone. The physical link is set by default to 1M.
         • The test is stopped by pressing the SW1 button on the app.
         • To change PHY to 2M, press the SW2 button on the BLE/Zigbee® device (b). Restart and stop the test with 2M using the SW1 button again.

   – With a Nucleo board (running a BLE throughput application) (c) , the BLE scan starts automatically, then connects to the "DT_SERVER" Dynamic device (b).
      ◦ To start the “Downlink” test, press the SW1 button on the Dynamic device (b), the results are displayed on the BLE device (c) serial console. The physical link is set by default to 1M.
      ◦ The test is stopped by pressing on the SW1 button again.
      ◦ To change the PHY to 2M, press the SW2 button. Restart and stop the test with 2M using the SW1 button again.
      ◦ Pressing SW3 button on Dynamic device (b) changes the connection interval on the BLE device (c) with some predefined values (currently 23, 43, 63, 83, 105, 124, 153, 209, 305, 405 ms) and restart the Throughput test.
5. The red LED keeps toggling on the Zigbee® coordinator (a) during and after the transfer.

On BLE disconnection, the Dynamic device (b) restarts advertising (green LED flashing).

8.7.5 Test results

Test Config 1 with Samsung Galaxy S10e

Table 10 gives the results in the above configuration.

Table 10. Samsung Galaxy S10e test results

Config	Download 1M	Download 2M	Upload 1M	Upload 2M
Connection interval	48.75 ms		11.25 ms	
BLE only	90.8 kB/s	161 kB/s	42.5 kB/s	75.6 kB/s
Dynamic mode	87.7 kB/s	150.2 kB/s(1)	41.7 kB/s	75.6 kB/s(1)
ZB Toggle	Yes, ZB toggle works, but pretty unstable and slow, with Fails			

1. *A Zigbee® fatal error is sometimes observed during BLE 2M transfer after a period of time, and BLE keeps working.*

The download results are 3 to 10% lower than in a BLE configuration only. The upload results are similar, but the Zigbee® activity is slow and unstable during the tests, with toggle errors because BLE takes up most of the bandwidth, and BLE parameters cannot be tuned.

The Zigbee® activity recovers after the test ends.

Test configuration 2 with BLE app on Nucleo board

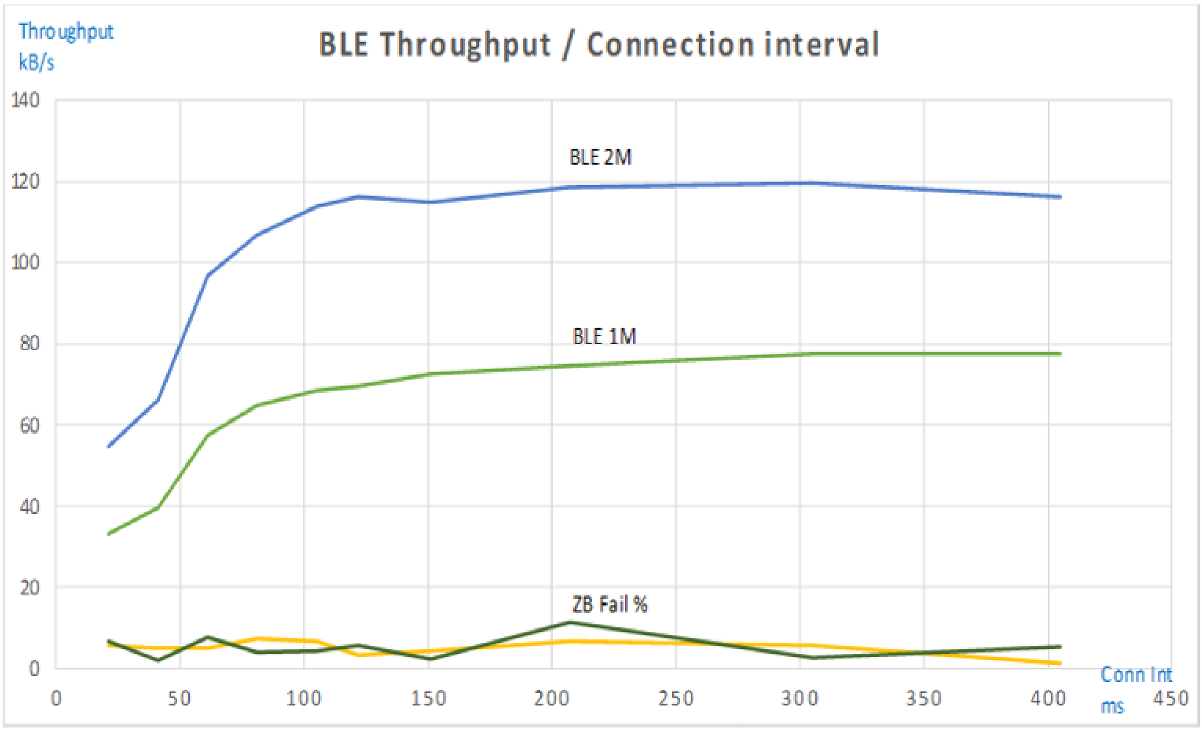
The BLE parameters are all configurable on the Nucleo board. The tests are run with different connection configurations and CE_{max} parameters.

The results in Table 11 and Figure 30 are obtained with a Zigbee® toggle frequency of 500 ms.

Table 11. Dynamic BLE throughput results

Cfg	Conn Interval	CE_{max}	Dnl 1M kB/s	Toggle ZB	Fail ZB	Fail %	Dnl 2M kB/s	Toggle ZB	Fail ZB	Fail %
Default	400	625	78.1	unstable and very slow	13/159	8.2%	123.8	Unstable and slow	15/242	6.2%
2	405	385	77.4	Very stable	5/410	1.2%	116.3	stable	19/363	5.4%
3	305	285	77.5	stable	21/385	5.5%	119.6	stable	11/448	2.5%
4	207	188	74.4	stable	27/401	6.7%	118.7	stable	41/360	11.3%
5	151	132	72.5	stable	17/396	4.3%	114.9	stable	11/498	2.2%
6	122	103	69.4	stable	14/444	3.1%	116.2	stable	21/374	5.6%
7	105	85	68.6	stable	26/400	6.5%	113.8	stable	20/452	4.4%
8	81	62	64.8	stable	30/411	7.3%	106.9	stable	17/445	3.8%
9	61	42	57.4	stable	22/448	4.9%	96.6	stable	31/405	7.6%
10	41	22	39.4	stable	20/508	4.9%	66.1	Very stable	10/525	1.9%
11	21	12	33.3	stable	23/420	5.5%	54.6	Pretty stable	25/371	6.7%

Figure 30. Dynamic BLE throughput graph



8.7.6 Notes and comments

The maximum CE value is set at 20 ms below connection interval value to allow Zigbee® to have at least 20 ms available during each connection interval.

For instance, with a connection interval of 105 ms, the CE_{max} value is set to 85 ms, which gives 85 ms for BLE and 20 ms for Zigbee®.

Due to a specific route management issues with the Zigbee® protocol (described in Section 8.5.4), the connection interval values which are a direct sub-multiple of 15 ms must be avoided to prevent a deadlock in Zigbee® mechanism.

Therefore, the following values are used: 21, 41, 61, 81, and so on instead of 20, 40, 60, 80, and so on as connection intervals, the latter prevented a stable Zigbee® behavior.

Connection interval values have been empirically adjusted to the provide the best value for a stable Zigbee® behavior, for example 122, 151, 207 and so on.

When requesting a new connection interval from the dynamic device, it is up to the BLE master to accept it, so the final value of CI may not be exactly the one requested:

- The list of possible values requested are: 23, 43, 63, 83, 105, 124, 153, 209, 305, 405 ms, and the real valued obtained are: 21, 41, 61, 81, 105, 122, 151, 207, 305, 405 ms.
- These final values are always a multiple of 1.25 ms (e.g. 81.25 instead of 81).

The Zigbee® fail rate is always under 12%. However, some packets are lost, the Zigbee® toggle is sometimes stalled for 2 to 5 seconds, then recovers normal execution toggling every 500 ms.

8.8 BLE/Zigbee® dynamic NVM application

8.8.1 Firmware and software requirements

To run the BLE/Zigbee® dynamic NVM application, the following binaries and/or software apps must be installed:

Table 12. **Zigbee® dynamic NVM application firmware**

Role	ID	Device	M0 firmware / M4 application
Zigbee® coordinator	(a)	Nucleo board	Cortex®-M0 `stm32wb5x_ZigBee_FFD_fw.bin` Cortex®-M4 `ZigBee_OnOff_Coord.bin`
Dynamic device	(b)	Nucleo board	Cortex®-M0 `stm32wb5x_BLE_ZigBee_FFD_dynamic_fw.binM4 ble_zigbee_Dyn_NVM.bin`
BLE device	(c)	Smartphone (Android™/iOS™)	"ST BLE Sensor" phone application, available on App Store and Google Play
		Nucleo board	Cortex®-M0 `stm32wb55xx_ble_full_host_stack_cut2.1.bin` Cortex®-M4 `BLE_p2pClient.out`

8.8.2 Dynamic NVM application description

The main components of the BLE/Zigbee® dynamic NVM application are the following:

- A Zigbee® coordinator (a) running the On/Off cluster server.
- A Dynamic device (b) configured as a Zigbee® router running the On/Off cluster client, and a BLE peripheral running a P2P server app.
- A BLE device (c) running the P2P client app.

This is illustrated in Figure 14.

8.8.3 Running dynamic NVM application

This demonstration illustrates the simultaneous BLE and Zigbee® connections, with the red LED toggling on the Dynamic device (b), reflecting BLE activity, and the red LED toggling on the Zigbee® coordinator (a), reflecting Zigbee® activity. NVM persistent data is used to save/restore the Zigbee® device state.

The Dynamic device (b) starts with both BLE and Zigbee® modes activated.

The BLE example implements point-to-point communication using the P2P component.

This is illustrated in Figure 15.

The blue LED on the Zigbee® coordinator indicates the network is formed and the network is joined with the Dynamic device (b).

The On/Off cluster client (Dynamic device (b)) sends a toggle command to the On/off cluster server (Dynamic device (a)) every 1 sec, the red LED toggles on Zigbee(R) coordinator (a).

Then connect Dynamic device (b) to the BLE via "ST BLE Sensor" app (c) and send the BLE toggle commands through the app, shown by the red LED state on Dynamic device (b) at the same time.

8.9 Dynamic NVM application detailed behavior

1. The Dynamic device (b) (P2P server) starts nLE advertising (green LED toggling) and simultaneously starts to join the Zigbee® Network.
2. With the Zigbee® NVM feature, the Dynamic device (b) tries to start from persistence data, which enables a restart from the previous configuration. This leads to two choices:
   – For persistent data read from NVM are valid: the router takes back is role in the network. See point 4.
   – For non persistent data found or corrupted data, a fresh start is performed.

   See point 3. below.

3. The node attaches to the existing Zigbee® network (Zigbee® coordinator (a) running `ZigBee_OnOff_Server_Coord` application).
   a. When the Zigbee® successfully joins the network, the Dynamic device (b) blue LED turns ON Zigbee® is successfully joined.
4. When the Zigbee® connection is established, the Dynamic device (b) sends a `ZB ON_OFF` toggle every second to the Zigbee® coordinator (a):
   a. The red LED toggles on the Zigbee® coordinator (a).
5. The BLE connection initiation depends on the BLE device that is used, smartphone or a third Nucleo board:
   a. Using a smartphone, then the "ST BLE Sensor" app (smart Phone application) scans and connects to the P2P Server by selecting the "P2PZBSRV1" entry.
      i. Once connected, the application starts to search the P2P services and characteristics.
      ii. LED button service, LED characteristic and button characteristics are discovered at this stage (this service is STMicroelectronics specific).
      iii. Pressing the LED button on the app turns the red LED on or off on the Dynamic device (b).
   b. Using a Nucleo board (running a BLE_p2pClient application (c)), pressing the SW1 button starts the scanning process:
      i. The blue LED lights up on the Bluetooth® Low Energy device (c).
      ii. After the scan is complete, the Bluetooth® Low Energy device (c) connects to the P2P server on Dynamic device (b) dynamic node, and sends a BLE toggle when pressing SW1.
      iii. As a result, the red LED toggles on the Dynamic device (b), at each SW1 button press on BLE device (c).
6. Zigbee® persistence data is automatically updated in NVM when needed.
7. When the user power cycle of the Dynamic device (b), the persistent data is read back and the stack configuration is restored.
8. Push button SW2 is used to delete Zigbee® NVM (fresh start is done on next start up).

On BLE disconnection, the Dynamic device (b) restarts advertising with the green LED flashing.

9 BLE/Zigbee® static concurrent mode

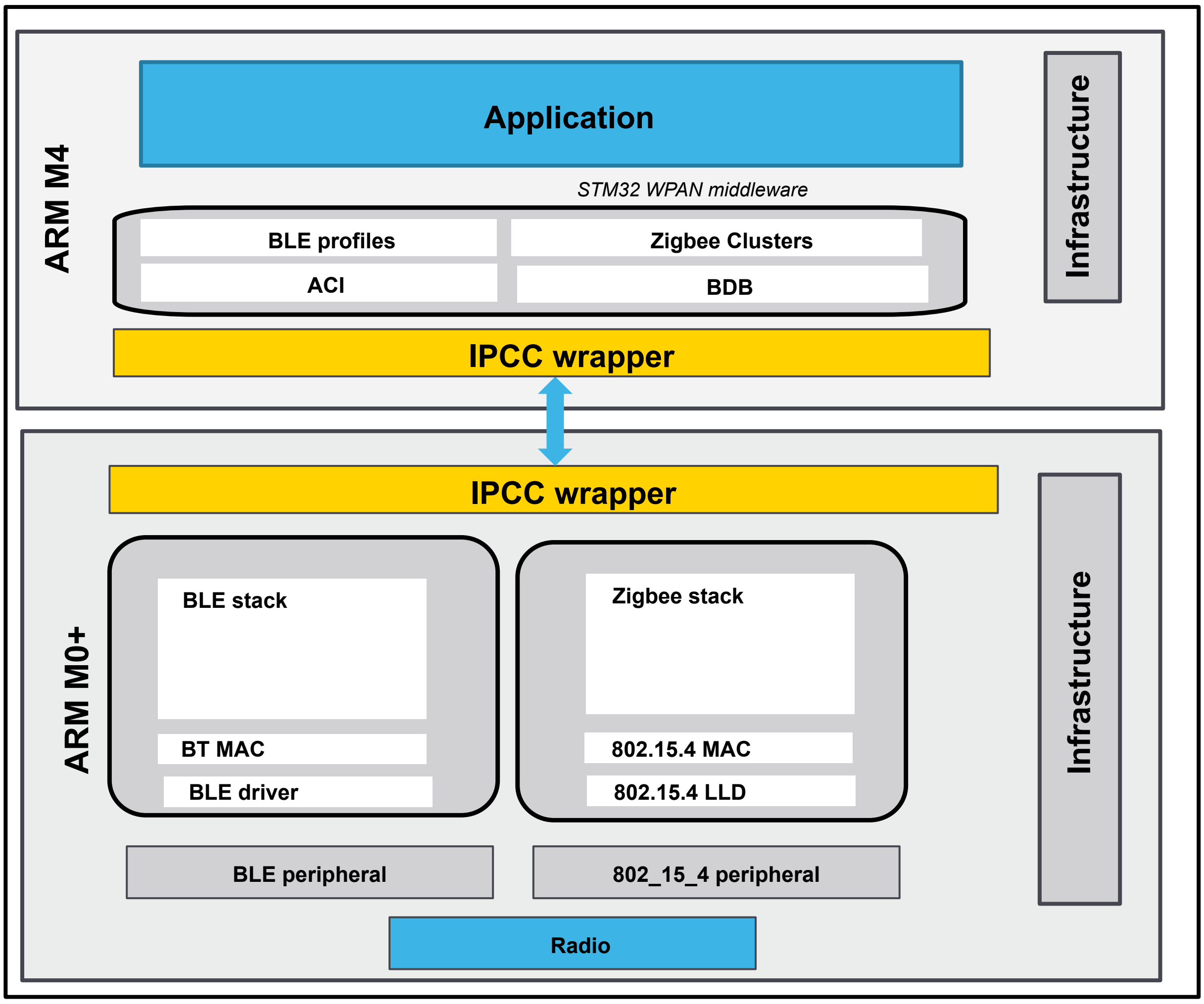
9.1 Static mode overview

An example of static concurrent mode (BLE/Zigbee®) is provided in the STM32WB firmware package. This application is located under:

`Projects\P-NUCLEO-WB55.Nucleo\Applications\BLE_ZigBee` directory.

The static mode is a subset of the dynamic mode allowing it to switch from BLE to Zigbee® and vice-versae, upon user request. This device connects through BLE to a smartphone running the "ST BLE Sensor" application and once the BLE activity is stopped, it joins a Zigbee® network. Then, once the Zigbee® application is completely stopped, it is possible to connect to BLE again. This is illustrated in the figure below.

Figure 31. Static concurrent mode on STM32WB Series



9.2 Supported static firmware

Static mode

The Static mode is a subset of the Dynamic mode which allows the switch from BLE to Zigbee® and vice-verse, upon user request. When the BLE protocol is running, the Zigbee® stack is not running. When the BLE is stopped, the system switches back to Zigbee®. In this case, the Zigbee® stack is fully re-initialized.

The static mode firmware includes both BLE and Zigbee® stacks:

- The BLE stack is Bluetooth® Low Energy 5.0 certified.
- Two flavors of the Zigbee® stack are supported on the STM32WB Series device, FFD (full feature device) and RFD (reduced feature device). These stacks are Zigbee® PRO 2017 (revision 22) certified.

Table 13. **static stacks**

Stacks supported	Firmware associated
Zigbee® FFD + Bluetooth® Low Energy 5.0	`stm32wb5x_BLE_ZigBee_FFD_static_fw.bin`
Zigbee® RFD + Bluetooth® Low Energy 5.0	`stm32wb5x_BLE_ZigBee_RFD_static_fw.bin`

- An FFD accepts any role in the network which is either:
  - a router
  - a coordinator
  - an end device.
- An RFD only supports the end device role. An RFD has a smaller footprint compared to an FFD. When building an application acting as a 'sleepy end device', in order to reach optimal low power consumption, it is mandatory to build this application using the Zigbee® RFD stack.

These binaries are used for static concurrent mode applications. Example of such applications are provided as follows:
`Projects\P-NUCLEO-WB55.Nucleo\Applications\BLE_ZigBee` directory.

Important:
Before running any BLE/Zigbee® application on the STM32WB Series, ensure the proper firmware is downloaded on the Cortex®-M0. If it is not the case, use the STM32CubeProgrammer to load the appropriate binary.

All available BLE/Zigbee® binaries are located under:

```
/Projects/STM32WB_Copro_Wireless_Binaries/STM32WB5x
```

Refer to:

```
/Projects/STM32WB_Copro_Wireless_Binaries/STM32WB5x/Release_Notes.html for the detailed
procedure on how to change the wireless coprocessor binary
```

9.3 Static mode vs Dynamic mode

In this section, the difference between Static and Dynamic modes is discussed in the following points:

- Dynamic mode offers a greater number of functions over the Static mode.
- Static mode offers a more optimized memory usage and more effective memory management. For example, when Zigbee® is stopped the Cortex®-M0 memory (but allocated on Cortex®-M4 side) is completely freed when we switch back to BLE. This does not apply to the Cortex®-M4 memory.
- In Static mode, there is a gain of 7 kbytes versus Dynamic mode.
- Throughput in Static mode is improved by around 5 to 10 % over the Dynamic mode.
- The overall system robustness is improved due to the fact that the communication protocols are only used once at a time with less potential conflicts.
- Static mode implementation may be a specific customer requirement. The product may be dedicated to only one protocol and may not need the other at all.
- In stress tests which is represented by sending Zigbee® toggles at max rate, the static mode performance is slightly better using than the Dynamic mode. It provides an improvement of around 5% since more bandwidth allocated to 802.15.4 radio. But for the standard "use" case, this may have no impact.

The design of the system depends greatly on the product specification and the required mode.

Revision history

Table 14. **Document revision history**

Date	Version	Changes
22-Jun-2021	1	Initial release.

Contents

List of figures

List of tables