3 Things to Know Before Choosing Your Bluetooth Mesh Hardware Platform

Author: Kai Ren

Version: 1.0.1

Revision Date: 9 December 2020

Bluetooth®

Revision History

Version	Date	Author	Changes
1.0	August 2018	Kai Ren	Initial Version
1.0.1	9 November 2020	Martin Woolley	Language Changes

Table of Contents

1.0 Introduction

It has been several months since the Bluetooth Special Interest Group (SIG) released the Bluetooth mesh specifications. In that time, we have participated in a number of exhibitions, events, and conferences, including Bluetooth Asia 2017 in Shenzhen and CEATEC JAPAN 2017 in Tokyo, where we got a chance to meet with a wide range of developers and engineers. Many of them asked the same question: How do I choose the right Bluetooth mesh hardware platform for my product or prototype?

There are hundreds of Bluetooth® Low Energy (LE) chips on the market from a variety of module suppliers, and it is easy to order them from manufacturers or distributors. With the increasing adoption of Bluetooth mesh by Bluetooth member companies, the market will soon have hundreds of Bluetooth mesh hardware platforms that developers and engineers can choose from.

However, with the release of the Bluetooth mesh specification, there are a lot of new concepts and technical terms for developers and engineers to understand. This paper provides some tips and guidance for selecting the right architecture, memory, and power consumption strategy to ensure your Bluetooth mesh solution aligns with your specific product requirements.However, with the release of the Bluetooth mesh specification, there are a lot of new concepts and technical terms for developers and engineers to understand. This paper provides some tips and guidance for selecting the right architecture, memory, and power consumption strategy to ensure your Bluetooth mesh solution aligns with your specific product requirements.

However, with the release of the Bluetooth mesh specification, there are a lot of new concepts and technical terms for developers and engineers to understand. This paper provides some tips and guidance for selecting the right architecture, memory, and power consumption strategy to ensure your Bluetooth mesh solution aligns with our specific product requirements.

2.0 Architecture

The first step of an embedded system design is to select a suitable architecture to meet the system requirements or engineering target specifications. Different architectures can have various benefits and limitations. In this paper, we divided the architecture into two types: single or dual architecture.

2.1 Single vs Dual Architectures

For a Bluetooth mesh architecture (even for a non-mesh Bluetooth LE solution) there are two common architectures to choose from (shown in Figure 1): single and dual chip/module architecture. The single chip/module hardware platform relies on one chip or module to deal with all tasks and system requirements. The dual chip/module hardware platform uses one host microcontroller unit (MCU) plus a Bluetooth mesh co-processor. The co-processor is a dedicated part for Bluetooth mesh communication.

2.2 Single Chip/Module

As the name suggests, single chip implies that the application on this system is just running on a single chip or module. No extra processor unit is needed to help with computing and task scheduling; sometimes this is called an all-in-one solution.

Hardware

As for chip and module, especially from a hardware perspective, there are some design differences between the single and dual chip/module hardware platform.

A single chip usually doesn't include a crystal oscillator, power-up circuit, RF (radio frequency) coupling, and antenna, making it necessary for you to design those circuits yourself and add a custom designed button, LED, etc. RF circuit design is difficult and the designer should have a corresponding background. Ideally, you or someone on your team should be an expert at this. Otherwise, you could be facing a potential barrier.

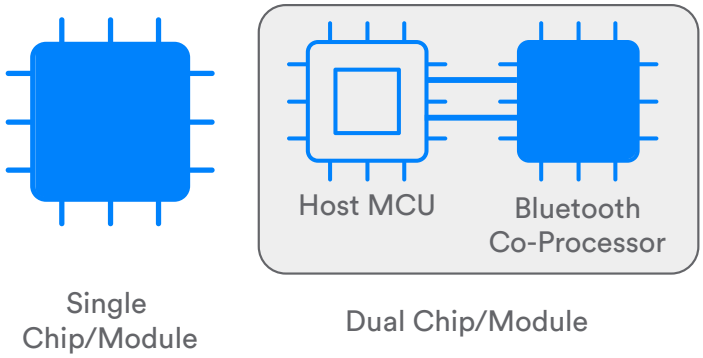

Figure 1 -Architecture comparison: single vs dual

With a single module, the vendor may help you complete the RF circuit, RF coupling, chip/PCB/external antenna, crystal oscillator circuit, and other necessary components needed to make the module work. As a developer or engineer, you need to mount this module on a motherboard, connect VCC and GND to the corresponding pins (caution: don't get them wrong or it may smoke), power them up, and add some customized components like a button and LED — then you are done.

Software/Firmware

Regardless of whether you are using a single chip or module, you need to take part in the firmware design, as you need to implement your application on the chip/module. Alternatively, you can outsource the firmware development to a contractor or system integration company.

Cost

In theory, a single-chip solution should be cheaper than a single module because a module vendor will charge a design and component fee. However, because the quality of chips and modules can differ, the cost may vary. It is important to get quotes from different vendors before making your choice and shop around for the best deal.

Flexibility

When using a single-chip solution, regardless of hardware and firmware, you manage all the design for your application and have greater flexibility than you would with a module. For example, if there is a bug on the antenna or in the firmware, you can fix it as soon as possible since you do not need to wait for a response from your module vendor.

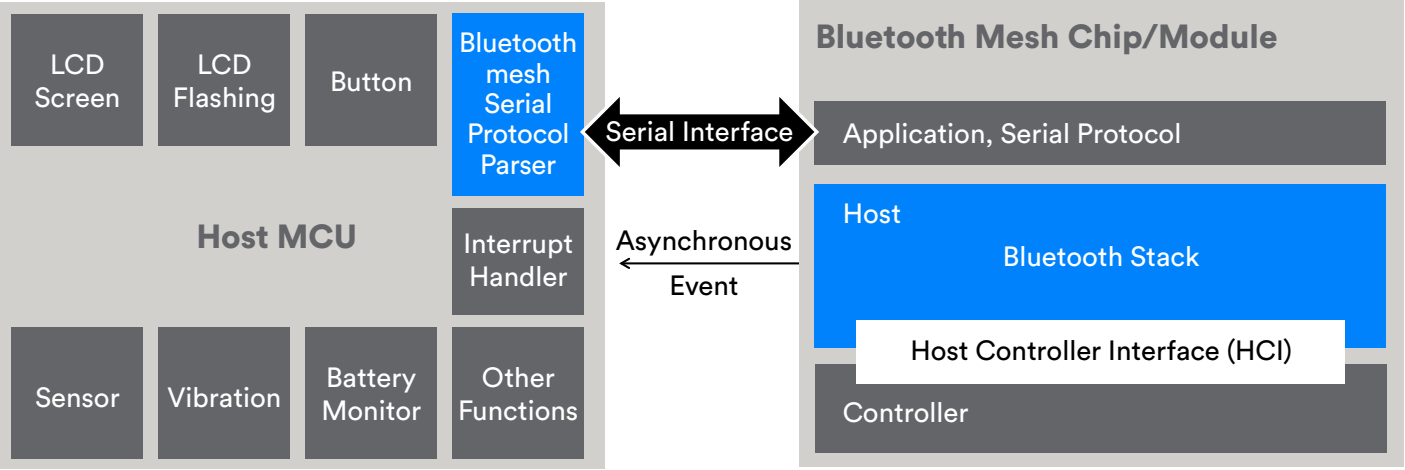

Figure 2 – Dual chip/module architecture

2.3 Dual Chip/Module

With this platform, there are two parts: the host MCU and the Bluetooth mesh co-processor, as shown in Figure 2.

Hardware

Throughout the process of embedded system design, the technical requirements can be complex, such as with handheld or portable instruments or meters, automatic control management systems, smart home gateways, etc. In these system designs, the functional requirements not only include wireless communication, like Bluetooth mesh, but also include key/button scan, LCD display driver, LED light flashing, buzzer PWM driver, vibrator driver, power management (especially in battery-powered equipment), sensor signal acquisition and processing, and other wireless communication technology support. For such a complex system, if the Bluetooth mesh function module is independent, such as a separated network co-processor (Bluetooth mesh co-processor), it is helpful not only for maintaining a modular system, but also for software/firmware debugging.

As shown in Figure 2, the host MCU has an interface to communicate with the Bluetooth mesh module. The interface can be a UART (Universal Asynchronous Receiver/Transmitter) or another serial communication protocol, such as SPI (Serial Peripheral Interface). This interface can be used for the interaction of control commands and data stream between the Bluetooth mesh network co-processor and the host MCU. The host MCU can send commands to the co-processor through the interface to control the co-processor's behavior in the network like provisioning, configuration, message transmission, checking network status and network information, and so on. Meanwhile, the co-processor can also be used to notify the host MCU by some asynchronous events, such as receiving message and network key refreshment.

Firmware

For dual chips/modules, the firmware on the host MCU must be designed by developers and engineers because the application on the host MCU is customized.

For example, different instruments have different vibration frequencies or LED flashing durations. You can migrate and reuse source code from other designs.

For example, you can use LCD driver source code from some open source repositories, but it is very hard to find a mature and existing hardware platform in the market which has the same functionalities that meet your host MCU's requirement. Please note, the host MCU should implement the serial protocol shown in Figure 2 because the host MCU needs to use this protocol to communicate with the Bluetooth mesh co-processor. In other words, you need to write code on the host MCU.

Flexibility

The Bluetooth mesh co-processor is more flexible, giving you two choices: a Bluetooth chip/module with Bluetooth mesh functionality or a mature module.

Selecting a **Bluetooth chip/module with Bluetooth mesh functionality** allows you to define the serial protocol for serial communication between the host MCU and the co-processor. The commands set over the serial protocol can be an AT command set or any private command set you would like to use, then program the firmware following the serial protocol you selected or defined. It seems complex, but the advantage is that you control and manage all software/firmware; it is easy to trace and debug if there is any abnormal phenomenon on the co-processor.

A Bluetooth chip/module with Bluetooth mesh functionality is suitable for complex embedded system design and allows you to define your own private serial protocol between the host MCUand the mesh co-processor.

With a **mature module**, you can take advantage of module vendors that provide a total solution; one that not only includes the module, but also includes the firmware that combines the serial protocol on the module. You will need a serial protocol user guide or application note to design firmware on the host MCU to interact with the module and make the Bluetooth mesh co-processor perform to your specifications. With a mature module, you need to take care of the firmware on the host MCU. The coding load is less than if you had gone with a Bluetooth chip/module with Bluetooth mesh functionality, but if there is a bug in the co-processor, you cannot fix it unless the module vendor releases the patches. So, if you select this option, you need to carefully evaluate the module.

3.0 Memory

When evaluating options for a Bluetooth mesh hardware platform, it is important to estimate the impact of memory consumption on your network configuration.

A main function of an MCU is data information processing. To deal with the data, the MCU needs some containers to store this data. We call these containers memory. A specified embedded system design, like Bluetooth mesh, follows this principle, but there are some special cases.

3.1 RAM

In an MCU, RAM (Random Access Memory) is mainly used to store various kinds of input/output data, intermediate calculation results, global variables and arrays, C function parameters that are passed into a function, and stack/heap. Its storage unit can be read or rewritten based on specific needs. However, RAM can only be used to temporarily store programs and data; once the power is off or should a power outage occur, the data in RAM is lost.

The advantage of flash is that it not only retains its content even while power is off, but also this content can be overwritten.

RAM and Friendship

RAM is important to an MCU and especially important to a Bluetooth mesh network running on the MCU. In the embedded design of a Bluetooth mesh network, one thing that may impact RAM is friendship. Friendship is the ability to help a node supporting the low power feature operate efficiently. A node that supports the friend feature, and has this feature enabled, is known as a Friend Node. The Friend Node stores messages destined for the Low Power Node (LPN) and only delivers them when the Low Power Node polls the Friend Node. The relationship between the Friend Node and the Low Power Node is known as Friendship. The Friend Node stores messages for its Low Power Node, in the RAM. The Friend Node needs to allocate an area in RAM to store those messages. Meanwhile, three factors influence RAM consumption:

- **LPNCount:** How many Low Power Nodes the Friend Node supports
- **bufferCount:** For each Low Power Node, how many messages can a Friend Node buffer for this Low Power Node
- **bufferLength:** For each message, how many bytes should it allocate

With these three key factors, it is easy to figure out the number of bytes of RAM consumed by a Friend Node in friendship.

```
RAM consumption = LPNCount * bufferCount * bufferLength
```

With this consumption, plus RAM consumed by the Bluetooth mesh stack and the customized application, developers can estimate RAM size.

3.2 Program Memory

Program memory contains the written program after it is burned. Currently, most MCUs use flash as their program memory media, compared with other program memory media like ROM (Read-Only Memory). The advantage of flash is that it not only retains its content even while power is off, but also this content can be overwritten. At present, most Bluetooth enabled MCUs are based on flash as their program memory.

In the process of selecting a suitable embedded hardware platform, it is common to see flash memory size from tens of Kbytes to 1 ~ 2 Mbytes. In the application of Bluetooth mesh, the function of flash program memory is similar to general and common embedded applications. What sets Bluetooth mesh apart is its ability to provide Over-the-Air (OTA) firmware updates.

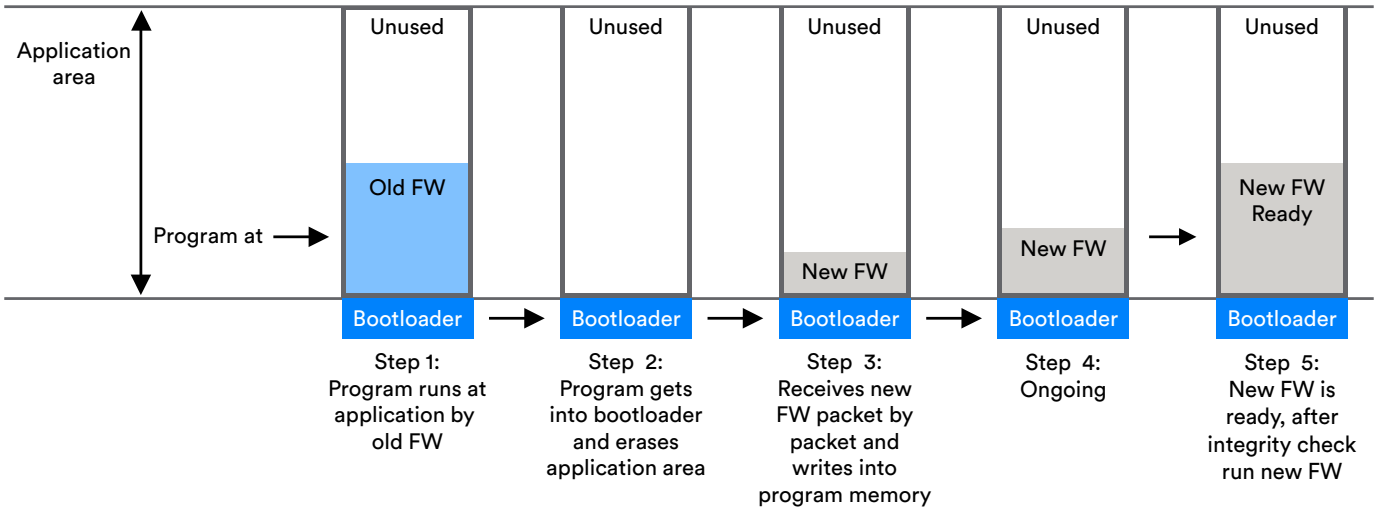

Figure 3 – Memory map by OTA through bootloader.

OTA is common for Bluetooth LE applications based on GATT. There may be one customized service and several customized characteristics that are dedicated for OTA. In which case, the firmware image can be transferred packet by packet through Bluetooth LE connectivity from smartphones, tablets, or handheld devices to the device that needs to be upgraded. For a Bluetooth mesh network, it becomes even more important because there could be thousands of nodes in the network. If a vendor or manufacturer needs to release a new firmware version for a bug fix, adding new features, or performance improvements, OTA is the best way to perform the firmware upgrade.

For OTA, there are two common methods to achieve:

Method 1: As shown in step 1 in Figure 3, the MCU needs to get into its bootloader. The bootloader includes the functionality to interact with a peer device like smartphones, tablets, or handheld devices. The firmware image is fragmented into packets to fit the length restriction of the packet and send them through Bluetooth LE connectivity to the target device. In steps 2, 3, and 4, the bootloader receives the fragmented image packet by packet and writes them into the application area of the flash. When image downloading is completed, and after the integrity check is successful, the MCU can run the program on the application area of the flash memory. At step 5, the OTA process is finished.

Method 2: In Figure 4, you can see there is no need to trigger the MCU into the bootloader (Step 1). The program can stay in the application. The MCU can continue to work as a node in the Bluetooth mesh network. At the same time, the MCU can build a connection with smartphones, tablets, or other handheld devices by GATT through Bluetooth LE connectivity to download a new firmware image and store the image in a New FW storage area (Step 2, 3, and 4). When the downloading process is complete, you need to make the MCU get into the bootloader to check image integrity. If the integrity checking is successful, the bootloader can copy the new firmware to the application area and erase the New FW storage area. In the final step (Step 6), you must make the MCU run the new firmware.

When you pursue the selection of components for an embedded hardware platform using Bluetooth mesh, a decision must be made for an OTA method. Then you need to estimate how much flash you need on the MCU.

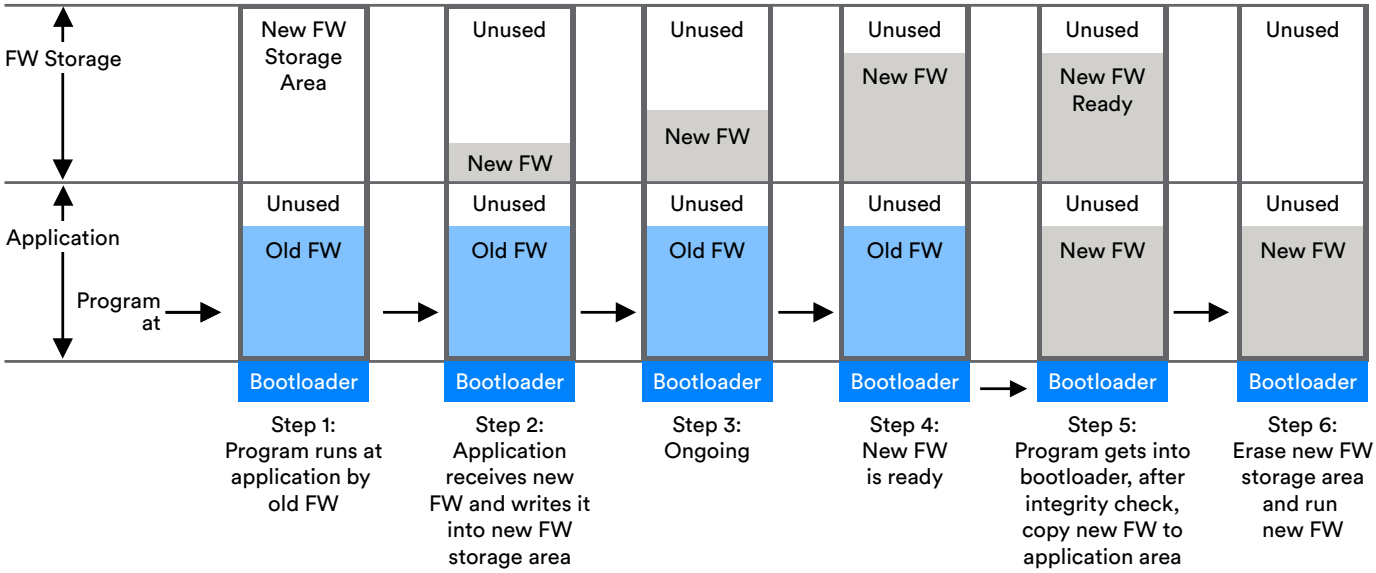

Figure 4 – Memory map by OTA through application.

Comparing methods 1 and 2, and you can see that:

- If **method 1** is used, the program needs to stay in the bootloader for a while. During that time, this node is offline in the Bluetooth mesh network, but program memory consumption is more forgiving than with method 2.

- With **method 2**, the program always stays in the application until the integrity check and new firmware is copied. The progress of those two operations is quick, and you can assume that the node is always online when this kind of OTA is running. The problem is that more flash is needed for the new firmware storage area, which means that for every single byte in the firmware image, two bytes of flash are needed: one byte for program memory and one byte for OTA new firmware storage.

3.3 Non-Volatile Memory

Non-volatile memory (NVM), or **non-volatile storage**, is a type of MCU memory that can retrieve stored information even after having power cycled or performed different types of resets on the MCU, such as brown-out, watchdog, and external GPIO pin reset. In a Bluetooth mesh application, you are likely to use NVM because persistent data needs to be stored permanently.

Last year, we created an article on provisioning that looked at the security keys used in a Bluetooth mesh network. Those keys are distributed by a provisioner and can be refreshed if the security keys have been compromised or could be compromised. All those keys should be stored in the on-chip NVM persistently and can be loaded into the application when the node is power cycled or reset, even if the firmware is updated, because the node MUST have those keys for network communication and encryption. Meanwhile, when performing a key refresh, two sets of keys need to save into the NVM for the key refresh process, in case there is an unpredictable power failure.

With Friendship, regardless of resetting or power cycling the Friend Node or Low Power Node, each node should know who is who; the Friend Node should know whose message it should buffer and the Low Power Node should know which Friend Node will poll the message back. So, all the corresponding information about friendship should be stored persistently. Additionally, important information like unicast address and model configuration data should be stored into NVM.

There are two common types of NVM on the MCU: flash and EEPROM (electrically erasable programmable read-only memory). The difference between flash and EEPROM is:

- On-chip flash is written/erased by page, whereas EEPROM is written/erased by byte
- On-chip flash is read by page, whereas EEPROM is read by byte
- Usually, on-chip flash volume is bigger than EEPROM

If you want to design a hand-held provisioner, on-chip flash is preferred because the provisioner needs to store lot of data for every node in the network. If you want to design a Low Power Node, EEPROM is enough.

4.0 Lowering Power Consumption

4.1 Lowering Power Consumption in an Embedded Design

Tuning connection parameters significantly reduce synchronizations... it's an ideal method for lowering the power consumption of the GATT Bearer.

Lowering the power consumption of an embedded system is always an important topic among engineers and developers and is motivated by the need to run applications as long as possible while consuming minimum power, especially in power-constrained systems. Optimal platforms for developing efficient, single-cell-powered architectures can enable designers to build compact battery-operated devices at the lowest cost and highest power efficiency. The following tips are useful for lowering power consumption for common embedded systems.

- Disable the clock source of unused peripherals like UART, I2C, or SPI
- All unused pins must be connected to a certain logic level
- Keep the system clock as low as possible, but only if it can meet the requirements

4.2 Bearer Layer Power Consumption

As you can see in Figure 5, the bearer layer is near the bottom of Bluetooth mesh architecture. A bearer is a communications system or protocol stack that is used to transport data between end points on behalf of another system or protocol stack. Bluetooth mesh has two types of bearers: the GATT Bearer and the Advertising Bearer. For each of these bearers, the method to lower consumption is different.

The **GATT Bearer** allows a device that does not support an Advertising Bearer to communicate indirectly with nodes of a mesh network, using a protocol known as the Proxy Protocol (Bluetooth Specification Mesh Profile V1.0, Section 6). A node that relays mesh messages between nodes which use the Advertising Bearer and nodes which use the GATT Bearer is known as a Proxy Node. The Proxy Node supports the GATT Bearer and can establish a connection with the node that just supports the GATT Bearer.

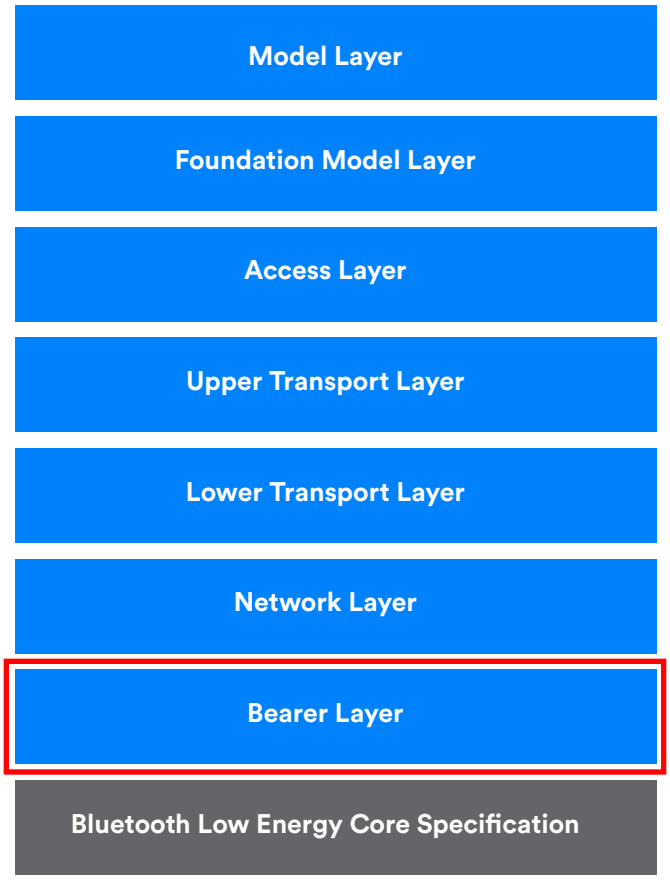

Figure 5 – Bluetooth mesh system architecture

After connection is established between the Proxy Node and the device that only has a GATT Bearer, the Proxy Node exposes its mesh proxy service and may add a mesh provisioning service (all of them are GATT services).

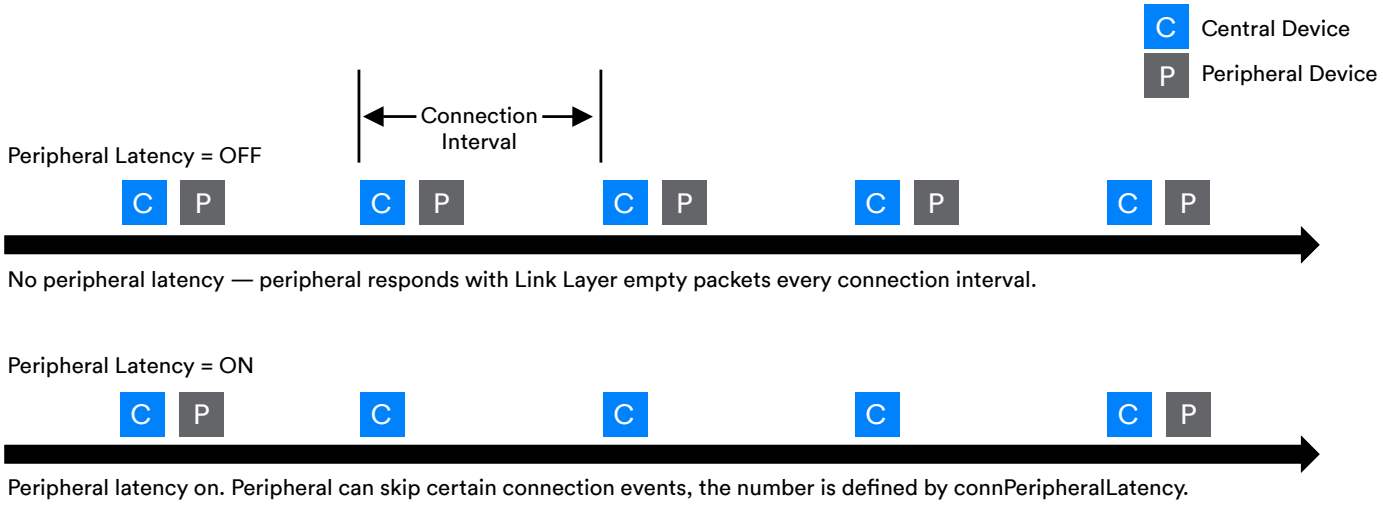

Figure 6 – Connection Interval and Peripheral Latency

Owing to its connection based, two connection parameters are vital for lowering power consumption of the GATT Bearer. Figure 6 shows these two parameters: connection interval and peripheral latency.

Connection Interval is an important parameter for Bluetooth LE connection-based applications and is defined as the specified duration in which two peer devices each send and receive data from only the other side. The window where two peer devices send and receive data is known as a connection event. If there is no application data to be sent or received, the two peer devices exchange empty link layer data packets to maintain the connection.

Peripheral Latency is a parameter that gives the peripheral device (link layer) the right to skip a number of connection events, which is defined by connPeripheralLatency, when the connection is established. This ability gives the peripheral device some flexibility to send or receive only if it needs to, otherwise connPeripheralLatency timed out.

Tuning connection parameters significantly reduce transceiver activity and is an ideal method for lowering power consumption. Increasing the connection interval lowers the power consumption for both devices and reduces the throughput in both directions.

Increasing the peripheral latency reduces power consumption for the peripheral during periods when the peripheral is idle to send.

The other bearer layer on the Bluetooth mesh network is the **Advertising Bearer**. In certain Bluetooth mesh networks, this bearer makes use of Bluetooth GAP advertising and scanning to receive and broadcast messages from other nodes. Because

it's based on GAP advertising and scanning, the aforementioned method for lowering power consumption over the GATT Bearer is not suitable for this bearer.

4.3 Low Power Feature

Another option for lowering power consumption is to use the low power feature. This allows a device's receiver to operate within a Bluetooth mesh network at significantly reduced duty cycles. A node that supports the low power feature, and has the low power feature enabled, is a Low Power Node.

4.4 Low Power Node Transition

From Figure 7, you can see there are several steps required to reduce a receiver's activity and put the core processor of the MCU into sleep mode. In runtime, if a Low Power Node's task scheduler is idle, it is time for it to go into sleep mode to save power. The Low Power Node then puts the receiver in standby mode to save power. In standby mode, the receiver consumes little power, but can still wake up rapidly. Then, the Low Power Node:

- Disables all unused peripherals like UART, PWM, SPI, or I2C during sleep mode
- Switches the clock source from high frequency, like external crystal oscillator with PPL, to low frequency, such as the internal RC oscillator/32KHz crystal oscillator, which can make the MCU operate at a low speed for greater power savings
- Enables corresponding interrupt sources in order to wake up the Low Power Node by interrupting events
- Makes the core processor go into sleep mode, at which time the Low Power Node goes into a power-saving state

The Low Power Node can also be awakened by an external interrupt event like a button click, sensor reading alert, vibration detection, or interval timer fired because it needs to send something to a Friend Node in case friendship is terminated. From Figure 7, you can also see there are several steps required after waking up the Low Power Node.

- During any interrupt events, the core processor is awakened and the system clock is switched from low frequency to high frequency to make sure there is enough computing resource for task processing
- When the system clock is stabilizing, the Low Power Node enables corresponding peripherals
- Then, it makes the receiver switch from standby mode to running mode

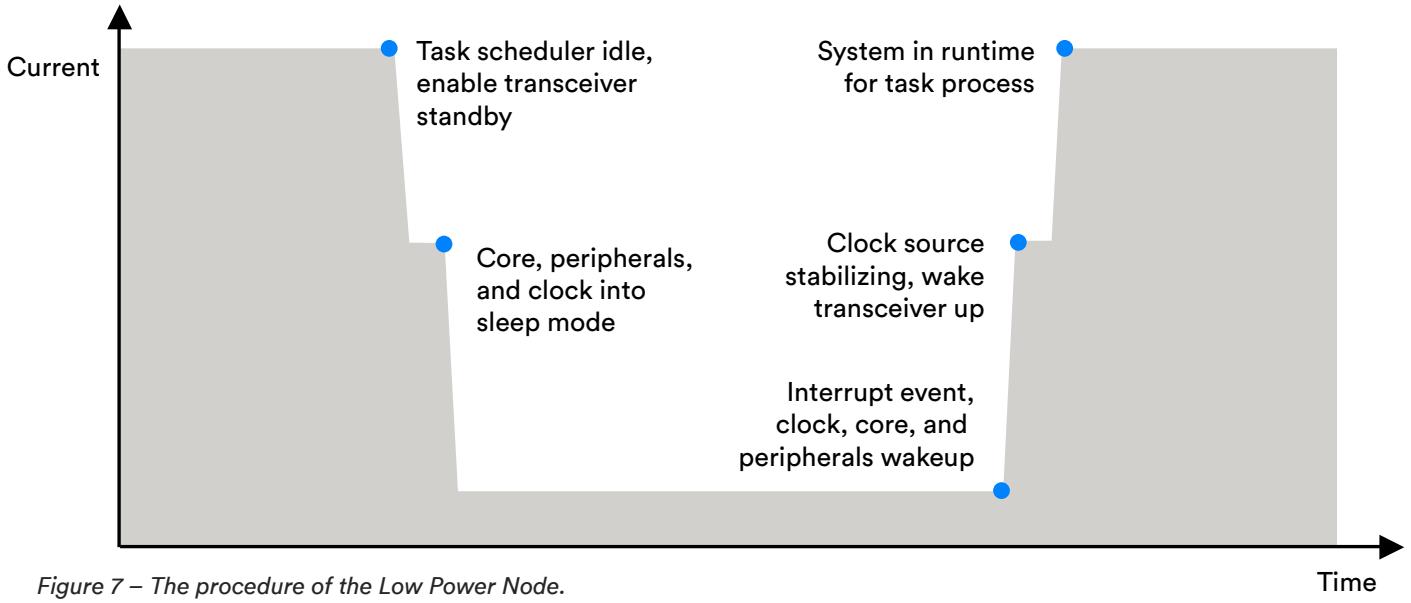

Figure 7 – The procedure of the Low Power Node.

4.5 Low Power Considerations

When you select a Bluetooth mesh solution for an Advertising Bearer, and it needs to use the low power feature, you should be aware of:

- Which sleep mode is the most suitable for your application, since modern MCUs have multiple sleep modes, each with different power consumptions, computing capabilities, and force-quit triggers
- Types of interrupt sources that can wake the core processor from the sleep mode you selected
- How long it takes to wake the core processor and receiver if the application is time sensitive
- The duration it will wait for clock source stabilizing from sleep mode
- Selecting a suitable clock source when the application is in low power mode (e.g. interval RC oscillator or external 32KHz crystal oscillator)

5.0 Building a Solid Knowledge Base

It is important to have a good understanding of the available technology before initiating a project that takes advantage of all that a Bluetooth mesh network offers. Now that you know what to look for in a suitable hardware platform, you should have the foundation you need to start working with a Bluetooth mesh network. By selecting the right chip/module; understanding key coding, data retention, and OTA factors; and knowing how to lower power consumption across your network, you will be better positioned to select the most appropriate Bluetooth mesh hardware platform that best meets your specified needs.

Once you have selected a chip/module, you can work with that silicon vendor to get the datasheet, recommended development environment, sample source code, reference design, and application note you need to proceed. You can also contact the Bluetooth SIG for Help & Support should you need any assistance.

To learn more about Bluetooth mesh, check out the Bluetooth Mesh Networking Specifications.
Once you have selected a chip/module, you can work with that silicon vendor to get the datasheet, recommended development environment, sample source code, reference design, and application note you need to proceed. You can also contact the Bluetooth SIG for Help & Support should you need any assistance.

To learn more about Bluetooth mesh, check out the Bluetooth Mesh Networking Specifications.