UG103.16: Multiprotocol Fundamentals

This version of UG103.16 has been deprecated.

For the latest version, see docs.silabs.com.

****

This document discusses the ways in which more than one protocol can be used on a single chip. It discusses considerations when selecting protocols for multiprotocol implementations. It also describes the basics of the Silicon Labs Radio Scheduler, a required component of a dynamic multiprotocol solution.

KEY POINTS

- Multiprotocol implementation modes
- Core requirements
- Protocol considerations
- Overview of the Radio Scheduler

Silicon Labs' *Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as OpenThread, Zigbee EmberZNet PRO, or Silicon Labs *Bluetooth*®, and associated development tools. The documents can be used as a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

1. Introduction

Silicon Labs is a leading provider of IoT (Internet of Things) infrastructure for our connected world.. At a high level Silicon Labs' goals for IoT are to:

- Connect all the devices in the home with best-in-class mesh networking, whether with Wi-Fi, Bluetooth, Bluetooth mesh, Zigbee, Z-Wave, Wi-SUN, or OpenThread.
- Leverage the company’s expertise in low-power, constrained devices.
- Enhance established low-power, mixed-signal chips.
- Provide low-cost bridging to existing Ethernet and Wi-Fi devices.
- Enable cloud services and connectivity to smartphones and tablets that promote ease of use and a common user experience for customers.

Achieving all of these goals will increase adoption rates and user acceptance for IoT devices in the Connected Home.

In general, implementing more than one protocol on a single device allows for:

- Cost savings: A single device can perform more than one function.
- Space savings: End user product packaging can be smaller and simpler when protocols can share a single radio.
- Energy savings: The number of devices on a network is reduced.

This document describes different ways to implement multiprotocol devices and the infrastructure requirements for an effective implementation. It discusses some aspects of protocol operation to be considered when selecting a protocol for a multiprotocol implementation. Finally it summarizes the operation of the Silicon Labs Radio Scheduler, an essential component of a dynamic multiprotocol implementation.

This document is focused on multiprotocol on a single chip. For more information about coordinating multiple radio chips like a one from Silicon Labs and an external Wi-Fi radio see *UG103.17: Wi-Fi® Coexistence Fundamentals*.

2. Types of Multiprotocol Implementations

Multiple protocols can be implemented on a single device in different ways:

- Switched: The application switches between protocols using a bootloader.
- Dynamic: The application time-slices between two protocols.
- Concurrent: The application is able to receive multiple protocols full time by:
  - Running on the same RF channel of a single radio,
  - Using Silicon Labs Concurrent Listening technology on a single radio, or
  - Using a multi-chip or multi-radio solution.

The following sections discuss these implementations in more detail.

2.1 Switched Multiprotocol

In switched multiprotocol, a device is initially programmed with one protocol, and then uses a bootloader to switch to another protocol at some point in the future. Switched multiprotocol has two primary use cases.

Future Proofing: Device manufacturers may need to sell their devices into different protocol environments, or may wish to plan for an environment changing over time.

Commissioning by Smartphone: Device manufacturers who are committed to a single protocol environment, such as Zigbee Home Automation, may wish to make the process of adding a new device more secure and user friendly. In this case the device is initially programmed with a Bluetooth commissioning application. The Home Gateway / Trust Center is also programmed with a Bluetooth application, and the customer uses a commissioning app on their smartphone. Using the smartphone app they can join a new device onto the network, set up a pairing with other appropriate devices in the network, then switch over to the network protocol. Each wireless network protocol has its own mechanisms for joining and pairing devices, but all can be accommodated with this mechanism.

An effective switched multiprotocol implementations requires:

- A multiprotocol-enabled platform with sufficient memory.
- A consistent API to use the radio.
- A cross-compatible bootloader.

2.2 Dynamic Multiprotocol

In a dynamic multiprotocol implementation, two protocols run concurrently but the application time-slices the radio and rapidly changes radio configurations, such as channel, to enable different wireless protocols to operate reliably at the same time. In time-slicing, the software schedules tasks based around their priority and minimum duration and will default to a background listen task between scheduled tasks. The scheduler can even interrupt or delay a lower priority task if a higher priority task is scheduled. .

A dynamic multiprotocol implementation allows a device to perform multiple simultaneous functions. For example an end user can use a smartphone app to connect to the device via bluetooth to control it or perform diagnostics, at the same time the device is connected to the Zigbee network routing packets and acting on Zigbee Cluster Library commands..

In addition to a multiprotocol-enabled platform with sufficient memory, common code infrastructure, common radio interface, and common API to use the radio such as that provided by Silicon Labs RAIL, an effective dynamic multiprotocol implementation requires:

- An RTOS to support task switching and resource sharing.
- A radio scheduler to manage time-slicing. See section 4. Radio Scheduler for background on the Silicon Labs radio scheduler.

See *UG305: Dynamic Multiprotocol User's Guide* for additional information about Silicon Labs dynamic multiprotocol implementation and *UG103.13: RAIL Fundamentals* for more information about Silicon Labs RAIL.

2.3 Concurrent Multiprotocol

2.3.1 Single Channel

In a single radio, single channel concurrent multiprotocol implementation, two protocols are able to run simultaneously on a single radio by sharing the same radio channel.

For example, a Zigbee/Thread gateway or controller device could manage both a Zigbee-based and a Thread-based network at the same time. In the case of sharing a single channel, the networks must coordinate what channel they use. In addition, they must be able to accommodate reduced bandwidth either by scaling their traffic or limiting the number of devices on the network
.

2.3.2 Concurrent Listening

A Silicon Labs technology called Concurrent Listening allows the Zigbee and Thread stacks to operate on independent 802.15.4 channels when used with the multiprotocol radio co-processor (RCP). This feature works on EFR32xG21 and EFR32xG24 parts only. For detailed information on the multiprotocol RCP see *AN1333: Running Zigbee, OpenThread, and Bluetooth Concurrently on a Linux Host with a Multiprotocol Co-Processor.*

Concurrent Listening allows a single radio to listen on two channels simultaneously. This is accomplished by extremely rapid switching (on the order of tens of microseconds) that allows the radio to detect preambles for incoming packets on either channel. Once a preamble is detected, the radio stays on that channel until the packet is received, then resumes rapid switching.

Note that this fast switching differs from the time slicing used in dynamic multiprotocol, because the dwell on each channel is just sufficient to detect a preamble and shorter than the full preamble time, so incoming packets are not missed on either channel. In contrast, for dynamic multiprotocol the time slot is large enough to receive scheduled packets, which means that the other channel is not being listened to during that time.

When enabling the concurrent listening feature the PHY performance is slightly degraded. The sensitivity of the IEEE 802.15.4 PHY in this mode drops to around -98 dBm for both the EFR32xG21 and EFR32xG24.

3. Protocol Considerations

It is important to recognize that a single radio cannot receive or transmit two packets for two different protocols simultaneously. To share a single radio, both protocols have to accept they will not have 100% use of the radio. They must therefore be able to survive loss of radio without significantly decreasing performance or losing application messages. “Significantly” in this case means that the application is working outside of its intended operational parameters, for example message latency or message loss is higher than acceptable.

While some protocols and applications will have strict timing requirements for packet transmission, in most cases the critical area of concern with multiprotocol is the receipt of incoming packets from the networks.

In order for a protocol to be a good candidate for a dynamic or concurrent multiprotocol implementation, the following considerations apply.

- Neither protocol can take over the radio for long periods (longer than a few milliseconds at a time) under normal operating conditions (that is except for commissioning, firmware upgrades, and so on).
- One or both protocols must have a robust mechanism for managing loss of incoming packets, such as MAC retries.
- One or both protocols must have very short packets and/or a short time required on the radio;
- For dynamic multiprotocol, one or both protocols must implement strict time slots / connection intervals.

Note: When using dynamic multiprotocol, a radio has to switch between two different PHYs and it 'disappears' from one network or another. For protocols where the device might be the 'parent' of a sleepy end device, if the parent device is not available when the sleepy end device wakes up to send a message, regular dependency on retries will impact the sleepy end device’s battery life.

Note: If you have a protocol that uses multiple radio configurations, you do not need to implement multiprotocol. For example, Silicon Labs' RAIL and the radio configurator support multiple radio configurations in the same protocol. See *AN971: EFR32 Radio Configurator Guide* for more details. Multiprotocol makes sense if protocols have mostly independent protocol stacks.

3.1 Dynamic Multiprotocol Example

Bluetooth LE and Zigbee are suitable protocols for a dynamic multiprotocol implementation. Because of the low duty cycle for Zigbee traffic and the retry mechanisms in the Zigbee networking stack, a Zigbee Router can switch its radio to some other frequency/protocol for short periods without dropping any messages at an application level.

Bluetooth LE radio usage can be predicted and planned in advance. Bluetooth beacons are quite short packets, usually up to 30 bytes. The radio only needs about 1 ms to transmit the beacon and the interval between beacons is typically no shorter than 100 ms, thus providing a duty cycle of just 1%. This means that the radio can devote at least 99% of its time to the main Zigbee network.

If you are using Silicon Labs products, adding Bluetooth LE to most proprietary networks is also possible because of the short Bluetooth LE packets and the long delay between communications. The proprietary stack must be updated to work with the Bluetooth stack, and all proprietary communications should be examined to determine the priority and required timing accuracy of it. For more details, see *UG305: Dynamic Multiprotocol User's Guide* and *AN1134: Dynamic Multiprotocol Development with Bluetooth and Proprietary Protocols on RAIL*.

3.2 Concurrent Multiprotocol Example

Zigbee and Thread are one example of suitable protocols for a concurrent multiprotocol implementation.

The multiprotocol device can control which IEEE 802.15.4 channel it is operating on, to connect to other Zigbee and Thread devices. It only has a single IEEE 802.15.4 MAC/PHY to listen to/send on both networks concurrently, meaning that switching radio PHYs is not required.

The multiprotocol device manages incoming packets from either network by filtering both PAN IDs, directing to the appropriate networking stack

Operational constraints include:

The multiprotocol device must control selection of the same IEEE 802.15.4 channel for both Zigbee and Thread, which means that it is most likely a Zigbee Coordinator / Thread Leader, in practice the Gateway/Controller in both networks. If using Concurrent Listening on an EFR32xG21 or EFR32xG24, this constraint does not apply.

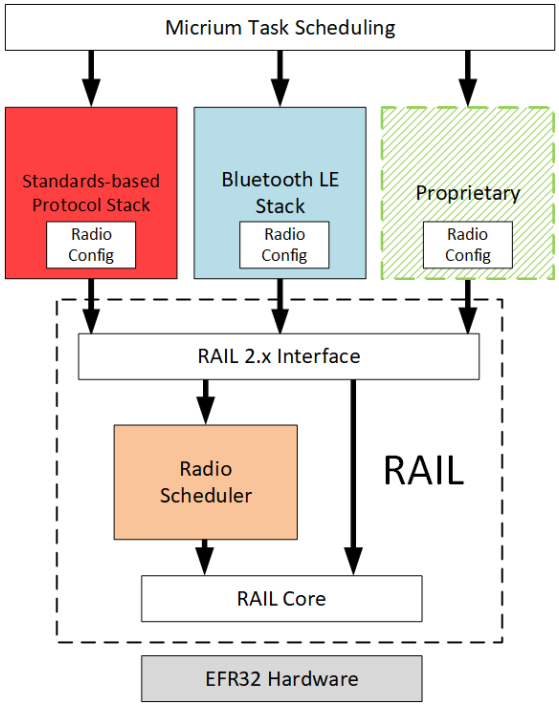
The device cannot be required to receive Zigbee and Thread packets simultaneously. However, because of MAC retries, this should not be a limitation in most use cases.

The combined duty cycle of the Zigbee and Thread traffic to/from the device must not exceed what would normally be tolerated by a single Zigbee or Thread device.

4. Radio Scheduler

A radio scheduler is an essential component of a dynamic multiprotocol implementation. It is a system for on-demand scheduling of radio tasks as requested by wireless stacks and the manufacturer's applications. This section introduces basic radio scheduler concepts. See *UG305: Dynamic Multiprotocol User's Guide* for details, including examples, of the Silicon Labs radio scheduler operation.

The Silicon Labs Radio Scheduler is part of the RAIL library. It operates above the radio hardware and below the RAIL API, as shown in the following figure.



Different radio events in each protocol may be more or less important, or more or less time sensitive, depending on the situation. The Radio Scheduler can take those into account when making decisions about conflicts and how to adjudicate them.

Micrium OS is an RTOS that allows stacks and application logic to share CPU execution time.

The Radio Scheduler uses the following concepts:

Radio Operation: A specific action to be scheduled. A radio operation has both a radio configuration and a priority. Each stack can request that the radio scheduler perform three radio operations:

- Background receive: Continuous receive, intended to be interrupted by other radio operation
- Scheduled receive: Receive at a future time with a minimum duration
- Scheduled transmit: Transmit at a future time with a minimum duration

Radio Config: Determines the state of the hardware that must be used to perform a radio operation.

Priority: Each operation from each stack has a default priority. An application can change default priorities.

Slip Time: The maximum time in the future when the operation can be started if it cannot begin at the requested start time.

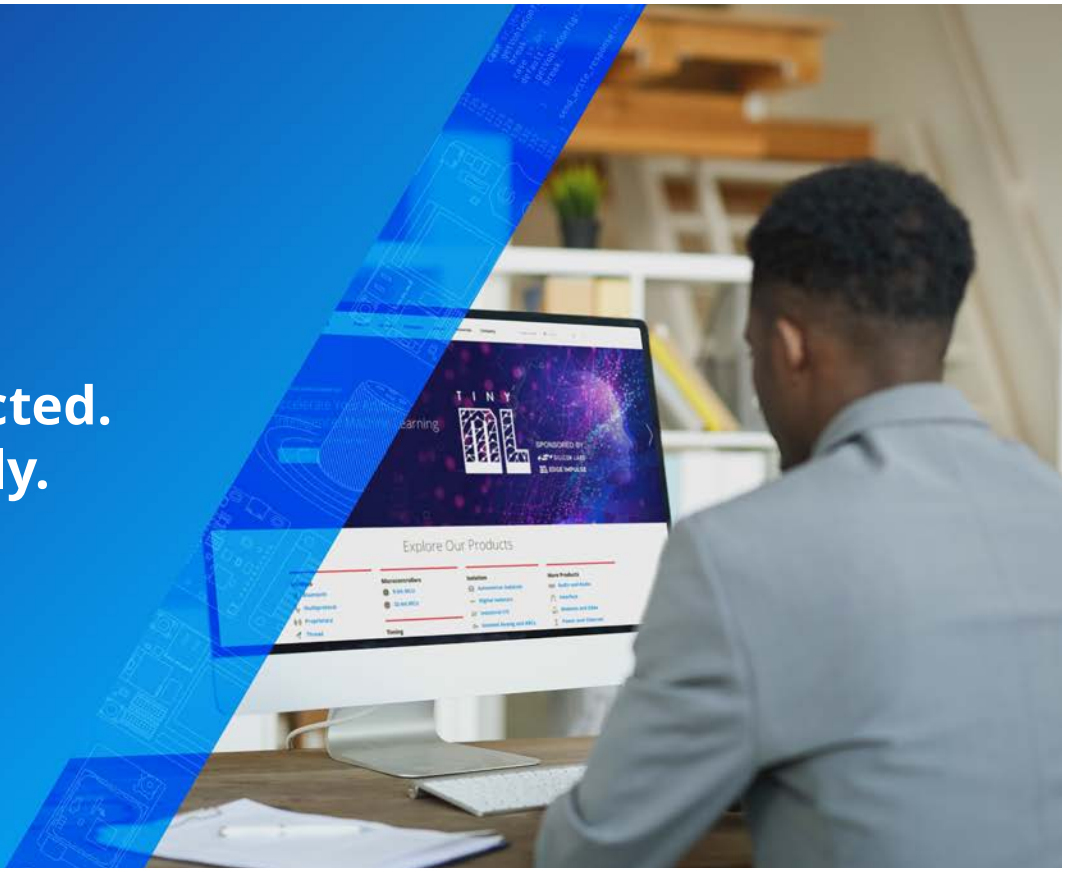
Yield: A stack must voluntarily yield at the end of an operation or sequence of operations, unless it is performing a background receive. Until the stack yields, the scheduler will not schedule lower priority tasks.

The Radio Scheduler can interrupt a scheduled radio operation if a higher priority task conflicts with it. This could occur in the following two circumstances:

1. A scheduled radio operation takes longer than expected and the corresponding stack does not yield before the higher priority radio operation must start.
2. A higher priority radio operation has just been scheduled to occur in the future and conflicts with a lower priority operation already scheduled.

Certain long-lived radio operations can have an outsized impact on the correct operation of the product. The application may need to coordinate these tasks between the protocols. If the application does not then the radio scheduler priorities will take precedence. This can result in the task being interrupted prematurely.

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com