AN1128: *Bluetooth*® Coexistence with Wi-Fi

This application note describes the Wi-Fi impact on Bluetooth® and methods to improve Bluetooth coexistence with Wi-Fi. It first describes design considerations to improve coexistence without direct interaction between Bluetooth and Wi-Fi radios. These techniques are applicable to the Mighty Gecko (EFR32MGx series) and Blue Gecko (EFR32BGx series). Next, this application note discusses the Silicon Labs Packet Traffic Arbitration (PTA) support to coordinate 2.4GHz RF traffic for co-located Bluetooth and Wi-Fi radios. This PTA feature set is available for the EFR32MGx series and EFR32BGx series.

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

KEY POINTS

- Wi-Fi impact on Bluetooth
- Improving unmanaged coexistence
- Implementing managed coexistence

Contents

1 Introduction

The 2.4GHz Industrial, Scientific and Medical (ISM) band supports Wi-Fi (IEEE 802.11b/g/n), Zigbee®/Thread (IEEE 802.15.4), Bluetooth, and Bluetooth Low Energy. The simultaneous and co-located operation of these different 2.4GHz radio standards can degrade performance of one or more of the radios. To improve interference robustness, each of the 2.4GHz ISM radio standards support some level of collision avoidance and/or message retry capability. At low data throughput rates, low power levels, and/or sufficient physical separation, these 2.4GHz ISM standards can co-exist without significant performance impacts. However, recent customer trends are making coexistence more difficult:

- Increased Wi-Fi transmit power level for “extended range”

  +30dBm Wi-Fi Access Points are now common.
- Increased Wi-Fi throughput.

  Depending on achievable Signal-to-Noise Ratio (SNR), high throughput requirements for file transfers and/or video streaming may result in high Wi-Fi duty cycle within the 2.4GHz ISM band.
- Integrating Wi-Fi, Zigbee, Thread, and Bluetooth Low Energy into the same device for gateway functionality

  This is required by Home Automation and Security applications, and provides easier end-node commissioning using Bluetooth Low Energy.

This application note describes the impact of Wi-Fi on Bluetooth Low Energy and methods to improve Bluetooth Low Energy coexistence with Wi-Fi on two Silicon Labs integrated circuit series, the Mighty Gecko (EFR32MGx) and the Blue Gecko (EFR32BGx).

- Section 3, Unmanaged Coexistence describes design considerations to improve coexistence without direct interaction between Bluetooth and Wi-Fi radios.
- Section 4, Managed Coexistence describes the Silicon Labs Packet Traffic Arbitration (PTA) support to coordinate 2.4GHz RF traffic for co-located Bluetooth and Wi-Fi radios.

Notes:

1. Zigbee and Thread devices (802.15.4) operate at less than +20dBm transmit power level. With normal network activity, Zigbee/Thread solutions have a relatively low RF duty cycle and Bluetooth solutions implement Automatic Frequency Hopping (AFH). Silicon Labs’ testing of Bluetooth blocked by Zigbee/Thread shows low impact on Bluetooth with AFH enabled and normal Zigbee/Thread network activity. However, if 100% RF duty cycle, Zigbee/Thread can degrade co-located Bluetooth performance. The Wi-Fi coexistence discussion and solutions described in this application note can be applied equally well to Zigbee/Thread coexistence. As such, all of the following solutions presented for “Wi-Fi Coexistence” can be applied to “Wi-Fi/Zigbee/Thread.15.4 Coexistence”.
2. This application note describes EFR32 Bluetooth and Bluetooth mesh coexistence support for Bluetooth 2.11.0 and Bluetooth Mesh 2.8.0. Not all coexistence support features in Bluetooth 2.11.0 and Bluetooth Mesh 2.8.0 are present in earlier versions.
3. Throughput this application note “Bluetooth Low Energy” is referenced as “Bluetooth”.
4. This application note addresses Bluetooth coexistence applications using EFR32 devices as per Bluetooth Core Specification v5.0 Vol 6 “Low Energy Controller” (point-to-point) and as per Bluetooth Specification Mesh Profile v1.0 (mesh network). These two applications have different coexistence considerations and, where necessary, this application note differentiates using the following terms:
   - “Bluetooth device” to reference Bluetooth Core Specification v5.0 Vol 6 “Low Energy Controller” (point-to-point) operation
   - “Bluetooth mesh device” or “Bluetooth mesh node” to reference Bluetooth Specification Mesh Profile v1.0 (mesh network) operation

2 Wi-Fi Impact on Bluetooth

Worldwide, Wi-Fi (IEEE 802.11b/g/n) supports up to 14 overlapping 20/22MHz bandwidth channels across the 2.4GHz ISM band with transmit power levels up to +30dBm. Similarly, Bluetooth supports 40 non-overlapping channels at 2MHz spacing with transmit powers up to +20dBm (Bluetooth Core Specification v5.0). For reference, 2.4GHz Zigbee and Thread (based on IEEE 802.15.4) support 16 non-overlapping 2MHz bandwidth channels at 5MHz spacing with transmit powers up to +20dBm. These Wi-Fi, Bluetooth, and Zigbee/Thread channel mappings are shown in Figure 1, where yellow highlighted channels are the three Bluetooth advertising (ADV) channels.

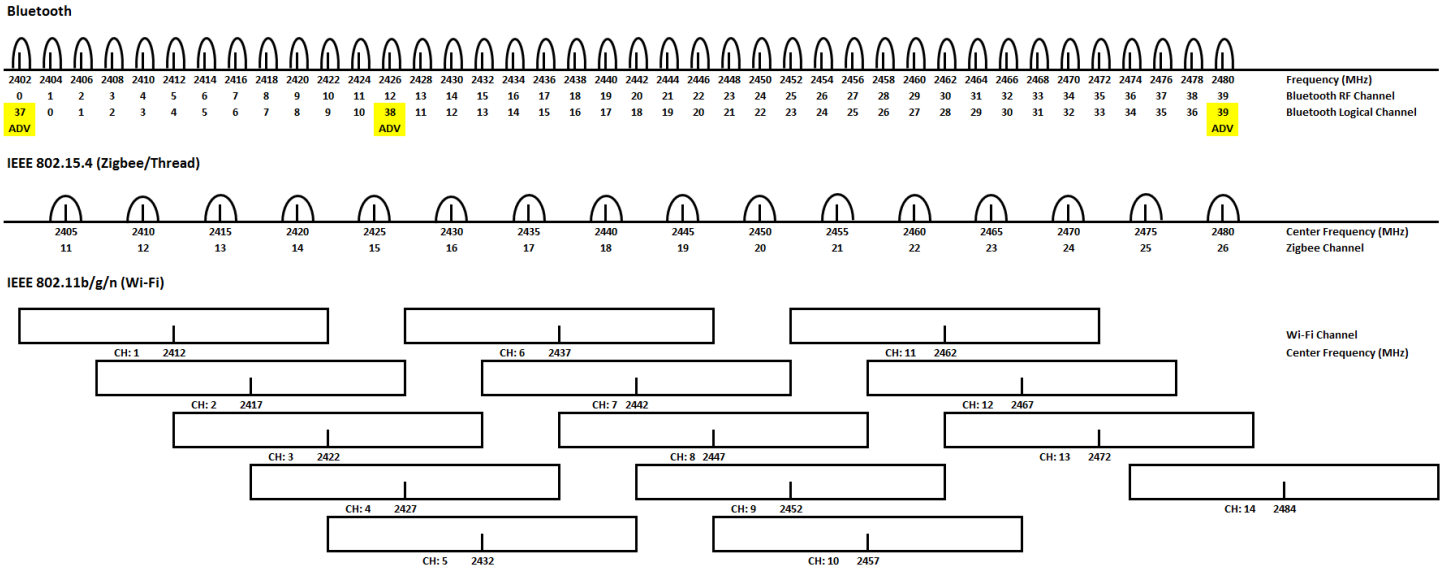


Figure 1. Bluetooth, 802.15.4, and 802.11b/g/n Channel Mapping (World-Wide)

Bluetooth channels 0 through 39 are available worldwide, but actual Wi-Fi channels available vary by country. For example, in the USA, only Wi-Fi channels 1 through 11 are available.

To better understand the effects of Wi-Fi on Bluetooth, Silicon Labs measured the impact of a 100% duty-cycled 802.11n (MCS3, 20MHz bandwidth) blocker transmitting at various power levels while receiving a Bluetooth 1Mbps 37-byte payload message transmitted at power level sufficient to achieve 0.1% BER (receive sensitivity). The results for co-channel, adjacent channel, and “far-away” channel are shown in Figure 2. All 802.11n and Bluetooth power levels are referenced to the Silicon Labs Mighty Gecko (EFR32MG13P732F512GM48) RF input. The test application was developed using the Silicon Labs Bluetooth 2.11.0 stack with the soc-dtm sample application running on the EFR32 DUT (Device Under Test) and a test script to control the DUT and RF test equipment.

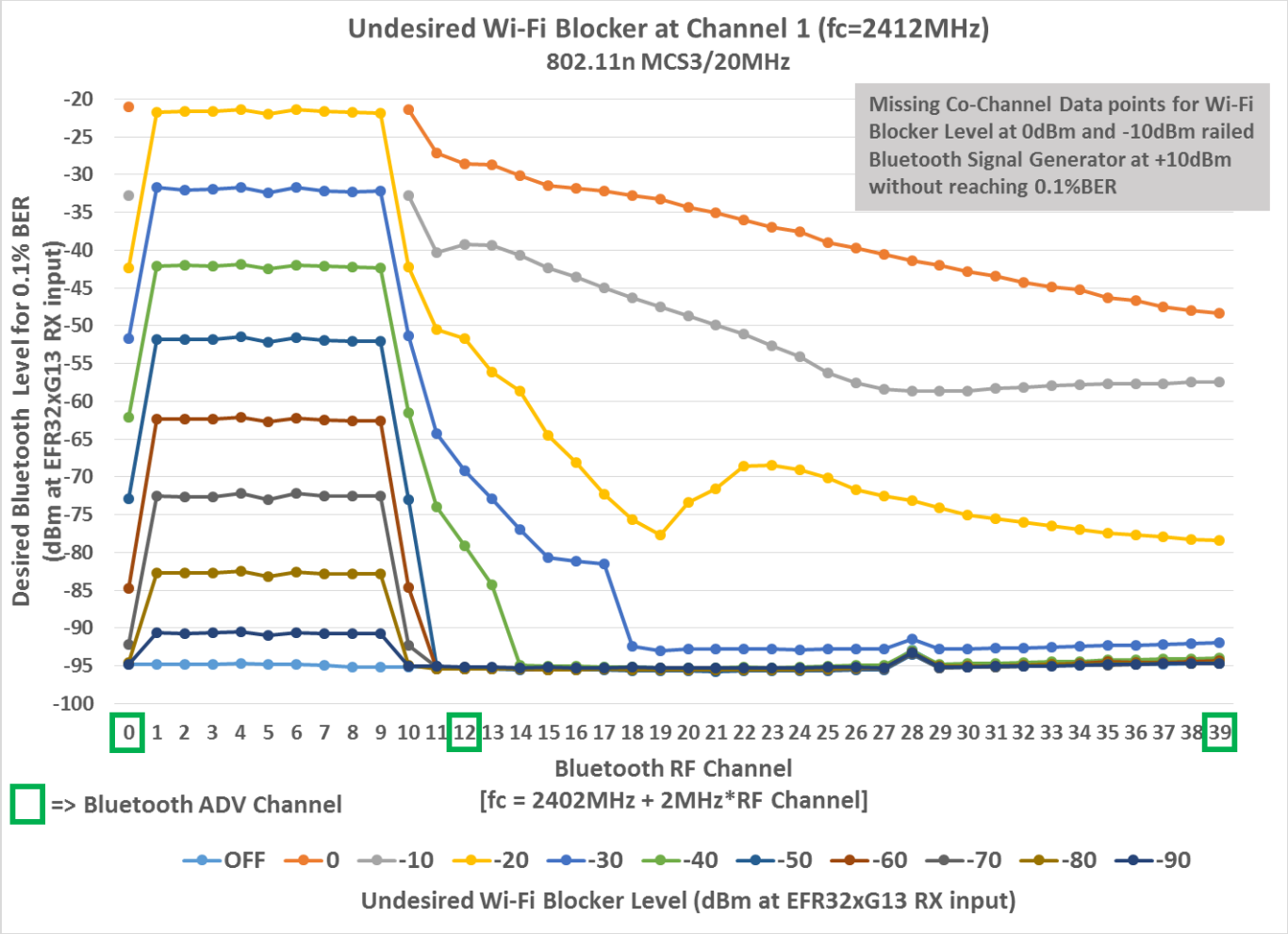


Figure 2. Bluetooth Low Energy Receive Sensitivity with 100% Duty-Cycled 802.11n (MCS3/20MHz) Wi-Fi Blocker

From Figure 2, the key observations about the impact of Wi-Fi (channel 1, MCS3/20MHz) on Bluetooth are:

Co-Channel (Bluetooth overlapping Wi-Fi):

- For Bluetooth RF channels 0 through 10, EFR32MG13P732F512GM48 can receive a Bluetooth 1Mbps signal at 2dB weaker than aggregate Wi-Fi transmit power (100% duty cycle).
  - This receive sensitivity limitation impacts both co-located and remote, not co-located, Bluetooth radios.

Adjacent Channel (Bluetooth within one Wi-Fi bandwidth):

- At Bluetooth RF channel 11, EFR32MG13P732F512GM48 can receive a -91.8dBm Bluetooth 1Mbps signal (RX sensitivity + 3dB) with -50dBm or weaker Wi-Fi transmit power (100% duty cycle).
- At Bluetooth RF channel 20, EFR32MG13P732F512GM48 can receive a -91.8dBm Bluetooth 1Mbps signal (RX sensitivity + 3dB) with -30dBm or weaker Wi-Fi transmit power (100% duty cycle).

"Far-Away" Channel (Bluetooth beyond one Wi-Fi bandwidth):

- At Bluetooth RF channels 21 through 39, EFR32MG13P732F512GM48 can receive a -91.8dBm Bluetooth 1Mbps signal (RX sensitivity + 3dB) with -30dBm or weaker Wi-Fi transmit power (100% duty cycle).

In a real-world environment, Wi-Fi is typically not 100% duty cycle and only approaches 100% duty cycle during file transfers or video stream in low Wi-Fi SNR conditions. As seen in Figure 3, the EFR32 receive sensitivity varies as the Wi-Fi blocker turns ON/OFF. The net result is the ability to see weaker signals when Wi-Fi is OFF, but not when strong Wi-Fi is ON (actively transmitting).

Figure 3 illustrates the receive range of a node (blue node) near a strong Wi-Fi transmitter. Relative to the blue Bluetooth node, the area inside the green circle represents the receive range when Wi-Fi is ON. The area between the green and yellow circles represents the receive range when Wi-Fi is OFF. From this figure:

- The green node is always receivable by the blue node.
- The yellow node is only receivable by the blue node when Wi-Fi is OFF.
- The red node is never receivable by the blue node.
- The yellow and red nodes are always receivable by the green node.

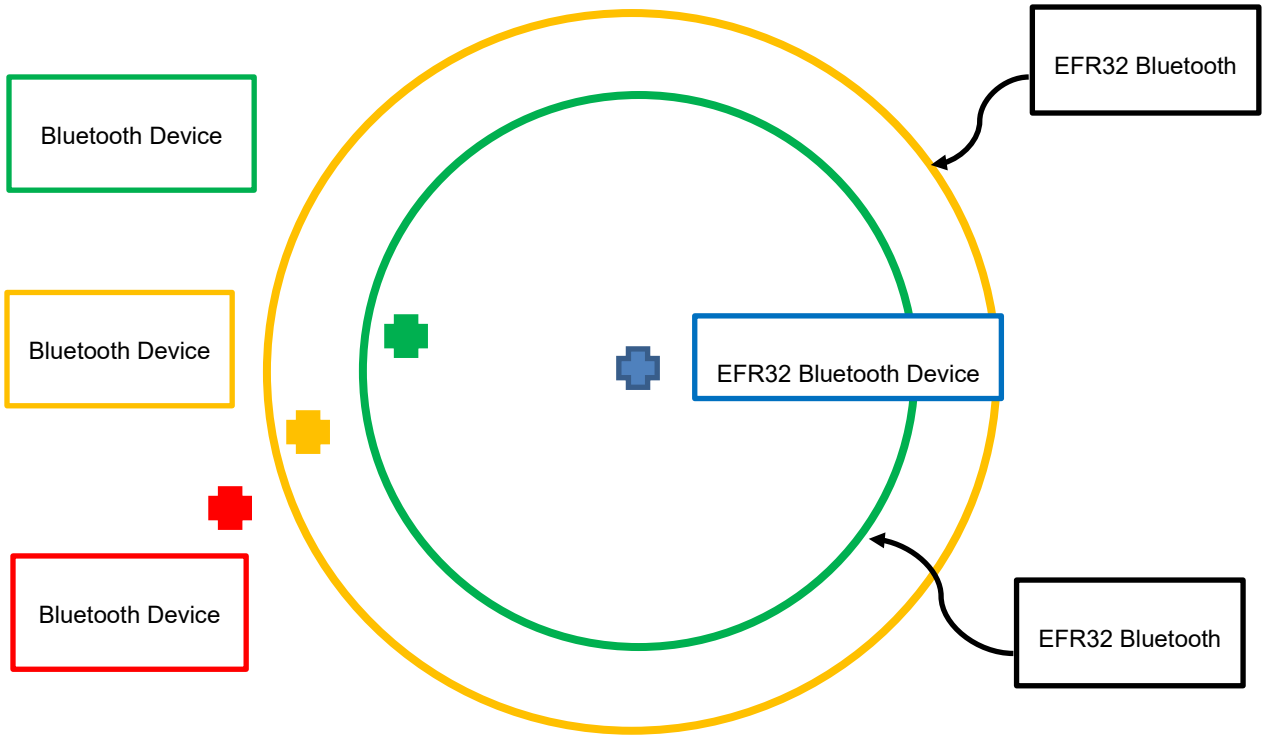


Figure 3. EFR32 Receiver Desensitized when Wi-Fi Transmitting

Depending on each Bluetooth device's TX level, RX sensitivity vs. blocker, channel, and relative attenuation and Wi-Fi TX level and duty cycle, the impact of strong Wi-Fi turning ON/OFF will vary. Based on Figure 3, the following example assumes:

- Wi-Fi co-located with Blue device
- Bluetooth channel is "far-away" (RF channel 39) from Wi-Fi channel (channel 1), indicating minimum Bluetooth RX sensitivity vs. Wi-Fi RX levels:

From Figure 2	Wi-Fi Level at Bluetooth RX Input				
	OFF	0	-10	-50	-90
0.1% BER RX Level (dBm) at Wi-Fi Level (dBm)	-95	-48	-57	-95	-95

- Typical radio TX levels:

Typical TX Levels (dBm)	Wi-Fi	Blue	Green	Yellow	Red
	20	10	10	10	10

- Attenuation between radios

Attenuation (Figure 3 example)		Transmitter				
		Wi-Fi	Blue	Green	Yellow	Red
Receiver	Wi-Fi	-	20	30	70	110
	Blue	20	-	30	70	110
	Green	30	30	-	40	80
	Yellow	70	70	40	-	40
	Red	110	110	80	40	-

- Bluetooth device RX **success**/**fail** with Wi-Fi **OFF**:

RX Level (dBm) (Figure 3 example)		Transmitter				
		Wi-Fi	Blue	Green	Yellow	Red
Receiver	Blue	OFF	-	-20	-60	-100
	Green	OFF	-20	-	-30	-70
	Yellow	OFF	-60	-30	-	-30
	Red	OFF	-100	-70	-30	-

- Bluetooth device RX **success**/**fail** with Wi-Fi **ON**:

RX Level (dBm) (Figure 3 example)		Transmitter				
		Wi-Fi	Blue	Green	Yellow	Red
Receiver	Blue	0	-	-20	-60	-100
	Green	-10	-20	-	-30	-70
	Yellow	-50	-60	-30	-	-30
	Red	-90	-100	-70	-30	-

For Figure 3 using example assumptions, all radio communication is maintained between Wi-Fi ON/OFF except:

- Blue device receives yellow device when Wi-Fi ON, but not Wi-Fi OFF.
- Green device receives red device when Wi-Fi ON, but not Wi-Fi OFF.

If devices are Bluetooth devices (point-to-point), blue device communication with:

- Green device is not impacted by Wi-Fi TX.
- Yellow device is erratic as Wi-Fi TX goes ON/OFF.
  - For high-duty cycle Wi-Fi TX, connection can become unstable as multiple connection intervals fail.
- Red device is not possible.

If devices are Bluetooth mesh devices (mesh network), blue device communication with:

- Green device is not impacted by Wi-Fi TX.
- Yellow device shows erratic communication as Wi-Fi TX goes ON/OFF.
  - For high-duty cycle Wi-Fi TX, communication would require a relay to forward/repeat missed RX messages from yellow device to blue device.
- Red device is not directly possible and a relay is required to forward messages between blue device and red device.
  - If green device is relay, communication with red device shows erratic communication as Wi-Fi TX goes ON/OFF.
  - If yellow device is relay, communication with red device is not impacted by Wi-Fi TX.

3 Unmanaged Coexistence

3.1 Implement Frequency Separation

From Figure 2, Bluetooth co-channel operation with Wi-Fi has the most impact on Bluetooth communication.

For Bluetooth devices (point-to-point) and Bluetooth mesh devices using Generic Attribute Profile (GATT) bearer communication, at least one ADV channel is minimally blocked and supports establishing a connection via Advertising, Scanning, and Initiating link-layer states. While establishing a connection, the Bluetooth connection master specifies the channel map, but the connection master can also update the channel map during connection. However, the Bluetooth connection slave must follow the channel map provided by master.

If EFR32 becomes the connection master, the Bluetooth channel map can be specified via:

```
static inline struct gecko_msg_le_gap_set_data_channel_classification_rsp_t*
gecko_cmd_le_gap_set_data_channel_classification(uint8 channel_map_len, const uint8* chan-
nel_map_data)
```

This command can be used to specify a channel classification for data channels. This classification persists until overwritten with a subsequent command or until the system is reset.

`channel_map` is 5 bytes and contains 37 1-bit fields. The *n*th such field (in the range 0 to 36) contains the value for the link layer channel index *n*:

0: Channel *n* is bad.

1: Channel *n* is unknown.

The most significant bits are reserved and shall be set to 0 for future use. At least two channels shall be marked as unknown.

For Bluetooth mesh devices using Advertising bearer communication, at least one ADV channel is minimally blocked and supports establishing communication via Advertising and Scanning.

3.2 Operate Wi-Fi with 20MHz Bandwidth

Because Wi-Fi 802.11n uses Orthogonal Frequency-Division Multiplexing (OFDM) sub-carriers, third-order distortion products from these sub-carriers extend one bandwidth on each side of the Wi-Fi channel. 802.11n can operate in 20MHz or 40MHz modes. If operated in 40MHz mode, 40MHz of the 80MHz ISM band is consumed by the Wi-Fi channel. However, an additional 40MHz on each side can be affected by third-order distortion products. These third-order products can block the Bluetooth receiver and is the primary reason adjacent channel performance is worse than “far-away” channel performance.

In proposing 40MHz mode for 802.11n, the Wi-Fi standard anticipated potential issues with other 2.4GHz ISM devices when Wi-Fi operated in 40MHz mode. During association, any Wi-Fi station can set the FortyMHz Intolerant bit in the HT Capabilities Information. This bit informs the Wi-Fi access point that other 2.4GHz ISM devices are present, forcing the entire Wi-Fi network to 20MHz mode.

If the Wi-Fi and Bluetooth radios are implemented with a common host, then the host should have the Wi-Fi radio set the FortyMHz Intolerant bit during association to force the Wi-Fi to 20MHz mode, increasing the number of channels available to Bluetooth and improving the Bluetooth performance.

If the application requires Wi-Fi to operate in 40MHz mode, frequency separation can be maximized by placing Wi-Fi channel at upper or lower end of 2.4MHz ISM band, minimizing the adjacent channels.

3.3 Use Bluetooth Retry Mechanisms

Bluetooth (point-to-point) messages requires responses. If a response is not received within programmable time, the application can resend the message up to a programmable limit.

Bluetooth mesh (mesh network) messages are sent via ADV payloads and responses are received during SCAN. Bluetooth mesh specifies:

- Optional relay nodes, which after a programmable time-out with no responses, can retransmit the original message for a programmable number of hops.
- Originator, after a programmable time-out with no response, can retransmit the original message for a programmable number of time-outs.

Both mechanisms improve Bluetooth mesh message success, but should be used with caution. More relays nodes, shorter time-outs, and more retries may improve an individual message's success, but these mechanisms can stress the mesh network by flooding too many identical messages. See *AN1137: Bluetooth® Mesh Network Performance* for details on these considerations.

3.4 Remove FEM (or Operate FEM LNA in Bypass)

EFR32 can deliver nearly +20dBm transmit power and has excellent receiver sensitivity without an external Front End Module (FEM). However, an external FEM can increase transmit power to +20dBm for increased range (in regions where this is permitted, for example, the Americas). The additional FEM Low-Noise Amplifier (LNA) receive gain also improves sensitivity. However, this additional gain also degrades the EFR32 linearity performance in the presence of strong Wi-Fi and degrades the EFR32 RX sensitivity (see Figure 2). An external FEM is not recommended and, if required for transmit power, the FEM LNA should be operated in bypass.

4 Managed Coexistence

The market trends of higher Wi-Fi transmit power, higher Wi-Fi throughput, and integration of Wi-Fi and Bluetooth radios into the same device has the following impacts:

- Advantages:
  - Host can implement frequency separation between Wi-Fi and Bluetooth.
  - Co-located Wi-Fi radio can force Wi-Fi network to operate with 20MHz bandwidth.
  - Co-located Wi-Fi and Bluetooth radios can communicate pending and/or in-progress activity on 2.4GHz ISM transmits and receives.
- Disadvantages:
  - Higher Wi-Fi transmit power requires greater antenna isolation.
  - Higher Wi-Fi throughput results in higher Wi-Fi duty cycle.
  - Antenna isolation is usually limited by the size of the product (only 15-20dB isolation is not unusual).

Assuming frequency separation achieves the “far-away” channel case and Wi-Fi only uses 20MHz bandwidth, a +20dBm Wi-Fi transmit power level at 100% duty cycle requires 50 dB antenna isolation to receive -92dBm Bluetooth 802.15.4 messages. This is generally not achievable in small devices with co-located Wi-Fi and 802.15.4.

Managed Coexistence takes advantage of communication between the co-located Wi-Fi and Bluetooth radios to coordinate each radio’s access to the 2.4GHz ISM band for transmit and receive. For the EFR32, Silicon Labs has implemented a coordination scheme compatible with Wi-Fi devices supporting PTA. This PTA-based coordination allows the EFR32 to signal the Wi-Fi device when receiving a message or wanting to transmit a message. When the Wi-Fi device is made aware of the EFR32 requiring the 2.4GHz ISM band, any Wi-Fi transmit can be delayed, improving Bluetooth message reliability.

Section 4.1, PTA Support Hardware Options discusses PTA support hardware options and section 4.2, PTA Support Software Setup discusses PTA support software setup. Section 7, Bluetooth Test Results in Multi-EFR32 Coexistence Test with 2-Wire Wi-Fi/PTA provides test results for EFR32 PTA implementation under various Wi-Fi operating conditions.

Note: EFR32 Bluetooth and Bluetooth mesh coexistence is supported in Bluetooth 2.11.0 and Bluetooth Mesh 2.8.0. Not all coexistence support features in Bluetooth 2.11.0 and Bluetooth Mesh 2.8.0 are present in earlier versions.

4.1 PTA Support Hardware Options

PTA is described in IEEE 802.15.2 (2003) Clause 6 and is a recommendation, not a standard. 802.15.2 originally addressed coexistence between 802.11b (Wi-Fi) and 802.15.1 (Bluetooth Classic) and does not describe an exact hardware configuration. However, 802.15.2 recommends that the PTA implementation consider the following:

- TX REQUEST from 802.11b to PTA and TX REQUEST from 802.15.1 to PTA
- TX CONFIRM from PTA to 802.11b and TX CONFIRM from PTA to 802.15.1
- STATUS information from both radios:
  - Radio state [TX, RX, or idle]
  - Current and future TX/RX frequencies
  - Future expectation of a TX/RX start and duration
  - Packet type
  - Priority (Fixed, Randomized, or QoS based)

Table 1 describes how 802.15.2 considers radio state, transmit/receive, and frequencies.

Table 1. IEEE 802.15.2 2.4GHz ISM Co-Located Radio Interference Possibilities

Co-located 802.11b State	Co-Located 802.15.1 State			
	Transmit		Receive	
	In-Band	Out-of-Band	In-Band	Out-of-Band
Transmit	Conflicting Transmits Possible packet errors	No Conflict	Conflicting Transmit-Receive Local packet received with errors	Conflicting Transmit-Receive Local packet received with errors or no errors if sufficient isolation for frequency separation
Receive	Conflicting Transmit-Receive Local packet received with errors	Conflicting Transmit-Receive Local packet received with errors or no errors if sufficient isolation for frequency separation	Conflicting Receives Possible packet errors	No Conflict

From Table 1, the frequency separation recommendations from section 3, Unmanaged Coexistence remain required for managed coexistence:

- 802.15.2 “In-Band” is equivalent to Co-Channel operation, which showed significant Wi-Fi impact on co-channel Bluetooth.
- 802.15.2 “Out-of-Band” covers both Adjacent and “Far-Away” Channel operation.

As such, for Managed Coexistence, Silicon Labs recommends continuing to implement all of the Unmanaged Coexistence recommendations.

- Frequency Separation
- Operate Wi-Fi in 20MHz Bandwidth
- Antenna Isolation
- Bluetooth Retry Mechanisms
- FEM LNA in Bypass

In reviewing existing PTA implementations, Silicon Labs finds the PTA master implementation has been integrated into many Wi-Fi devices, but not all Wi-Fi devices support a PTA interface. Figure 4 shows the most common Wi-Fi/PTA implementations supporting Bluetooth.

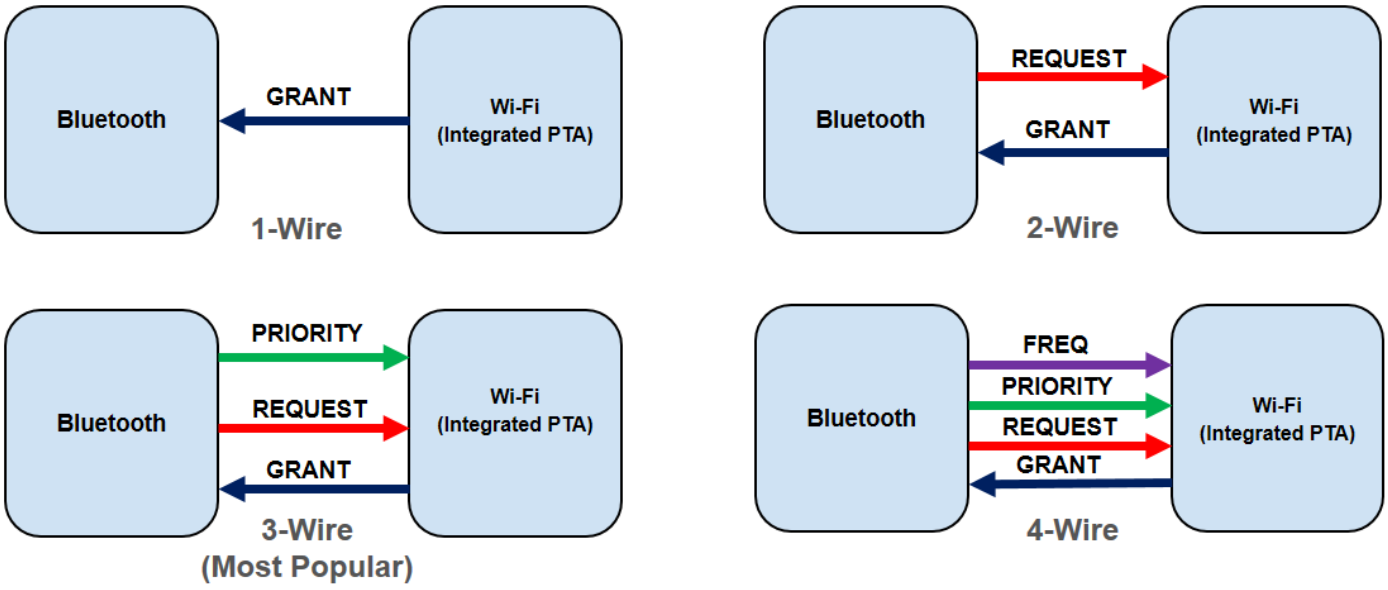


Figure 4. Typical Wi-Fi/Bluetooth PTA Implementations

1-Wire PTA

In 1-Wire, the Wi-Fi/PTA device asserts a GRANT signal when Wi-Fi is not busy transmitting or receiving. When GRANT is asserted, the Bluetooth radio is allowed to transmit or receive. This mode does not allow the external radio to request the 2.4GHz ISM and is not recommended.

An alternate 1-Wire implementation is a REQUEST signal from Bluetooth to Wi-Fi/PTA, where Bluetooth asserts REQUEST whenever it needs the 2.4GHz ISM band and expects Wi-Fi to always yield. This mode works very well for Bluetooth, but high priority Wi-Fi traffic can be compromised impacting Wi-Fi performance.

2-Wire PTA

In 2-Wire, the REQUEST signal is added, allowing the Bluetooth radio to request the 2.4GHz ISM band. The Wi-Fi/PTA device internally controls the prioritization between Bluetooth and Wi-Fi and, on a conflict, the PTA can choose to either GRANT Bluetooth or Wi-Fi.

3-Wire PTA

In 3-Wire, the PRIORITY signal is added, allowing the Bluetooth radio to signify a high- or low-priority message is either being received or transmitted. The Wi-Fi/PTA device compares this external priority request against the internal Wi-Fi priority, which may be high/low or high/mid/low and can choose to either GRANT Bluetooth or Wi-Fi.

PRIORITY can be implemented as static or directional (enhanced) priority. As of PTA support implementation in Bluetooth 2.11.0 and Bluetooth Mesh 2.8.0, Silicon Labs' EFR32 only supports static priority.

- Static: PRIORITY is either high or low during REQUEST asserted for the transmit or receive operation.
- Directional: PRIORITY is either high or low for a typically 20µs duration after REQUEST asserted, but switches to low during receive operation and high during transmit operation.

For platforms, such as Wi-Fi data routers that can achieve high Wi-Fi duty cycles, as well as IoT hubs that stream Bluetooth classic audio, implementing PRIORITY is highly recommended as it provides the Wi-Fi/PTA device with insight on the EFR32 REQUEST. PRIORITY is also configurable, both at compile time and at runtime, to address various product optimization requirements. However, static PRIORITY may not be necessary for platforms that do not experience high Wi-Fi duty cycles nor support Bluetooth audio streaming, freeing a GPIO pin on the EFR32 and SoC.

4-Wire PTA

In 4-Wire, the FREQ signal is added, allowing the Bluetooth radio to signify an "in-band" or "out-of-band" message is either being received or transmitted. Silicon Labs recommends maximizing frequency separation, making the FREQ signal mute. Silicon Labs' EFR32 does not support the FREQ signal and, for any 4-wire Wi-Fi/PTA with a FREQ input, Silicon Labs recommends asserting the FREQ input to the Wi-Fi/PTA.

4.1.1 Single EFR32 Connected to Wi-Fi/PTA

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.1.2 Multiple EFR32s Connected to Wi-Fi/PTA

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.1.3 Wi-Fi/PTA Considerations

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.2 PTA Support Software Setup

Bluetooth 2.6.0 and later and BT Mesh 1.3.0 and later contain EFR32 PTA support, enabling customers to implement EFR32 PTA configured for target Wi-Fi/PTA platform. **However, not all PTA support features are available in all revisions.**

Refer to Table 2 for PTA support features available in a particular release:

Table 2. PTA Coexistence Features Available in EmberZNet PRO and Thread Release

PTA Feature	Configure		Bluetooth						BT mesh	
	Compile-Time (hal-config.h)	Run-Time API [3]	2.6.0	2.7.0	2.8.0	2.9.0	2.10.0	2.11.0	1.3.0	1.4.0
PTA Enable/Disable	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REQUEST pin settings: Enable/disable, polarity, port, and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓
REQUEST signal is shared	✓		✓	✓	✓	✓	✓	✓	✓	✓
REQUEST signal max backoff mask	✓		✓	✓	✓	✓	✓	✓	✓	✓
REQUEST Window	✓	✓			✓	✓	✓	✓	✓	✓
GRANT pin settings: Enable/disable, polarity, port, and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓
Abort transmission mid packet if GRANT is lost	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PRIORITY pin settings: Enable/disable, polarity, port, and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓
PRIORITY signal is shared	✓				✓	✓	✓	✓	✓	✓
PRIORITY Escalation capability	✓	✓				✓	✓	✓	✓	✓
Debug Counters		✓		✓	✓	✓	✓	✓	✓	✓
Channel Map Masking		✓			✓	✓	✓	✓	✓	✓
RHO pin settings: enable/disable, polarity, port, and pin	✓					✓	✓	✓	✓	✓

4.2.1 Compile Time PTA Setup and Defaults

To enable PTA coexistence support, the following steps are required:

1. Create Bluetooth or BT mesh project in Simplicity Studio.
2. Ensure libcoex.a is included.
   - If NCP project, libcoex.a is included by default.
   - If SoC project, libcoex.a is added via the following steps:
   1. Copy libcoex.a from SDK
      - For Bluetooth: C:\SiliconLabs\SimplicityStudio\v<>\developer\sdks\gecko_sdk_suite\v<>\protocol\bluetooth\lib\<device>\IAR
      - For BT mesh: C:\SiliconLabs\SimplicityStudio\v<>\developer\sdks\blemesh\v<>\protocol\bluetooth\lib\<device>\IAR
   2. Paste to: <project folder>\protocol\bluetooth\lib\<device>\IAR
   3. In Studio IDE, right-click on the project in Project Explorer
   4. Go to **Project -> Properties -> C/C++ Build -> Settings**
   5. Go to **Tool Settings** tab -> **IAR Linker for ARM** -> **Library**
   6. Add the copied libcoex.a to project to list of libraries.
3. Add #defines to hal-config.h to enable/configure PTA coexistence.
   - Enable the PTA feature.

     The following #define is required:

     ```
     #define HAL_COEX_ENABLE                    (1)
     ```

   - REQUEST pin settings: Enable/disable, polarity, port, and pin

     The following #defines example enables active-high REQUEST on PC10:

     ```
     #define BSP_COEX_REQ_PIN                   (10)
     #define BSP_COEX_REQ_PORT                  (gpioPortC)
     #define BSP_COEX_REQ_ASSERT_LEVEL          (1)
     ```

     Note: In 1-Wire PTA configurations based on GRANT-only, REQUEST is not implemented. If REQUEST is not needed, remove the BSP_COEX_REQ_PORT and BSP_COEX_REQ_PIN #defines from hal-config.h.

   - REQUEST signal is shared.

     The following #define example disables Shared REQUEST for single-EFR operation as described in section 4.1.1, Single EFR32 Connected to Wi-Fi/PTA:

     ```
     #define HAL_COEX_REQ_SHARED                (0)
     ```

     The following #define example enables Shared REQUEST as described in section 4.1.2, Multiple EFR32s Connected to Wi-Fi/PTA:

     ```
     #define HAL_COEX_REQ_SHARED                (1)
     ```

   - REQUEST signal max backoff mask

     REQUEST signal max backoff determines the random REQUEST delay mask (only valid if REQUEST signal is shared). The random delay (in µs) is computed by masking the internal random variable against the entered mask. The mask should be set to a value of 2n-1 to ensure a continuous random delay range.

     The following #define sets backoff to recommended value:

     ```
     #define HAL_COEX_REQ_BACKOFF               (15)
     ```

   - REQUEST Window

     REQUEST Window adjusts the lead-time for REQUEST assertion before first Bluetooth TX or RX operation after REQUEST asserted. A TX operation will proceed if GRANT is asserted at the end of the REQUEST Window. An RX operation will attempt to proceed regardless of GRANT asserted or deasserted as Bluetooth RX does not impact other co-located radios. This feature's setting needs to at least exceed the maximum time for Wi-Fi/PTA to provide GRANT asserted or deasserted after REQUEST asserted.

The following #define example sets the REQUEST Window to 50µs:

```
#define HAL_COEX_REQ_WINDOW                (50)
```

Note: In Bluetooth SDK 2.11.0, REQUEST Window may exhibit an offset, depending on if EFR32BG entered CONNECTION state as a master device or a slave device, where REQUEST Window is ~15µs short. In earlier versions of the stack, this offset is ~25µs for master and ~40µs for a slave device. To ensure no issues with a short REQUEST Window, add at least 15µs for SDK 2.11.0, or at least 40µs for earlier SDKs, to Wi-Fi/PTA maximum REQUEST to GRANT delay.

- GRANT pin settings: Enable/disable, polarity, port, and pin

  The following #defines example enables active-low GRANT on PF3:

```
#define BSP_COEX_GNT_PIN                   (3)
#define BSP_COEX_GNT_PORT                  (gpioPortF)
#define BSP_COEX_GNT_ASSERT_LEVEL          (0)
```

  Notes:

  - Many Wi-Fi/PTA devices use the term WLAN_DENY or BT_DENY and describe as active-high. These active-high deny signals correlate with EFR32 active-low GRANT.
  - In 1-Wire PTA configurations based on REQUEST-only, GRANT is not implemented. If GRANT is not needed, remove the BSP_COEX_GNT_PORT and BSP_COEX_GNT_PIN #defines from hal-config.h.

- Abort transmission mid packet if GRANT is lost.

  If enabled, losing GRANT during a Bluetooth TX will abort the Bluetooth TX. If not enabled, losing GRANT after the start of a Bluetooth TX will abort the Bluetooth TX.

  The following #defines example disables *Abort transmission mid packet if GRANT is lost*:

```
#define HAL_COEX_TX_ABORT                  (0)
```

  The following #defines example enables *Abort transmission mid packet if GRANT is lost*:

```
#define HAL_COEX_TX_ABORT                  (1)
```

- PRIORITY pin settings: Enable/disable, polarity, port, and pin

  The following #defines example enables active-high PRIORITY on PD12:

```
#define BSP_COEX_PRI_PIN                   (12)
#define BSP_COEX_PRI_PORT                  (gpioPortD)
#define BSP_COEX_PRI_ASSERT_LEVEL          (1)
```

  Note: In 1-Wire or 2-Wire PTA configurations, PRIORITY is not implemented. If PRIORITY is not needed, remove the BSP_COEX_PRI_PORT and BSP_COEX_PRI_PIN #defines from hal-config.h.

- PRIORITY signal is shared

  The following #define example disables Shared PRIORITY for single-EFR operation as described in section 4.1.1, Single EFR32 Connected to Wi-Fi/PTA:

```
#define HAL_COEX_PRI_SHARED                (0)
```

  The following #define example enables Shared PRIORITY as described in section 4.1.2, Multiple EFR32s Connected to Wi-Fi/PTA2

```
#define HAL_COEX_PRI_SHARED                (1)
```

- PRIORITY Escalation capability

  The following #define example defaults PRIORITY to deasserted:

```
#define HAL_COEX_RX_HIPRI                  (0)
#define HAL_COEX_TX_HIPRI                  (0)
```

  The following #define example defaults PRIORITY to asserted:

```
#define HAL_COEX_RX_HIPRI                  (1)
#define HAL_COEX_TX_HIPRI                  (1)
```

Other combinations of HAL_COEX_RX_HIPRI and HAL_COEX_TX_HIPRI will also set PRIORITY to asserted.

- RHO pin settings: enable/disable, polarity, port and pin

  Radio hold-off (RHO) is effectively a second GRANT signal. However, when RHO is asserted, Bluetooth TX operations are blocked.

  The following #defines example enables active-low RHO on PC11:

```
#define BSP_COEX_RHO_PIN                    (11)
#define BSP_COEX_RHO_PORT                   (gpioPortC)
#define BSP_COEX_RHO_ASSERT_LEVEL           (0)
```

  Note: In most EFR32BG coexistence applications, RHO is not needed. If RHO is not needed, remove the BSP_COEX_RHO_PORT and BSP_COEX_RHO_PIN #defines from hal-config.h.

4.2.2 Run-Time PTA Re-configuration

The following PTA options can also be re-configured at runtime:

1. Disable/Enable the PTA feature.

   At runtime, the following code disables the PTA feature:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_ENABLE, 0);
```

   At runtime, the following code enables the PTA feature:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_ENABLE, 1);
```

2. REQUEST Window

   At runtime, the following code can be used to change the REQUEST_WINDOW:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_REQUEST_WINDOW_MASK, desired_request_window <<
GECKO_COEX_OPTION_REQUEST_WINDOW_SHIFT);
```

   Where `desired_request_window` is the REQUEST_WINDOW in µs.

   Note: In Bluetooth SDK 2.11.0, REQUEST Window may exhibit an offset, depending on if EFR32BG entered CONNECTION state as a master device or a slave device, where REQUEST Window is ~15µs short. In earlier versions of the stack, this offset is ~25µs for master and ~40µs for a slave device. To ensure no issues with a short REQUEST Window, add at least 15µs for SDK 2.11.0, or at least 40µs for earlier SDKs, to Wi-Fi/PTA maximum REQUEST to GRANT delay.

3. Abort transmission mid packet if GRANT is lost.

   At runtime, the following code disables Abort transmission mid packet if GRANT is lost:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_TX_ABORT, 0);
```

   At runtime, the following code enables Abort transmission mid packet if GRANT is lost:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_TX_ABORT, 1);
```

4. PRIORITY Escalation capability

   At runtime, the following code disables PRIORITY assertion:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_HIGH_PRIORITY, 0);
```

   At runtime, the following code enables PRIORITY assertion:

```
gecko_cmd_coex_set_options(GECKO_COEX_OPTION_HIGH_PRIORITY, 1);
```

5. Channel Map Masking

   If a EFR32BG device enters CONNECTION state as a master device, it controls which of the 37 data channels are used during the AFH. As a CONNECTION master, the EFR32BG can also update this channel map and communicate this update to a slave device. This feature can be used to make Bluetooth avoid being co-channel to Wi-Fi. See Figure 2 and section 3.1, Implement Frequency Separation for additional details

   If EFR32 becomes the connection master, the Bluetooth channel map can be specified via:

```
static inline struct gecko_msg_le_gap_set_data_channel_classification_rsp_t*
gecko_cmd_le_gap_set_data_channel_classification(uint8 channel_map_len, const uint8*
channel_map_data)
```

This command can be used to specify a channel classification for data channels. This classification persists until overwritten with a subsequent command or until the system is reset.

`channel_map` is 5 bytes and contains 37 1-bit fields. The *n*th such field (in the range 0 to 36) contains the value for the link layer channel index *n*:

0: Channel *n* is bad.

1: Channel *n* is unknown.

The most significant bits are reserved and shall be set to 0 for future use. At least two channels shall be marked as unknown.

4.2.3 Run-Time PTA Debug Counters

At runtime, PTA Debug Counters are also available and can be accessed and reset via the following function:

```
static inline struct gecko_msg_system_get_counters_rsp_t* gecko_cmd_system_get_counters(uint8
reset);
```

where:

- `reset = 0` leaves counters unchanged
- `reset = 1` resets all counters to 0 (after reading current counter values)
- `struct gecko_msg_system_get_counters_rsp_t` contains:

```
struct gecko_msg_system_get_counters_rsp_t
{
    uint16              result;
    uint16              tx_packets;
    uint16              rx_packets;
    uint16              crc_errors;
    uint16              failures;
}
```

where, since startup or last reset:

- `result` is success (== 0) or failure (!= 0) of `gecko_cmd_system_get_counters()` command
- `tx_packets` is number of successful packets transmitted
- `rx_packets` is number of successful packets received
- `crc_errors` is number of packets received with CRC failures
- `failures` is number of packets failures, which includes:
  - TX/RX abort
  - Scheduler failures
  - Shared REQUEST busy, GRANT denial, or RHO asserted, including Abort TX
  - RX buffer overflow
  - TX buffer underflow

4.3 Coexistence Configuration Setup Examples for Different Wi-Fi/PTA Applications

Example 1: Configure EFR32 PTA support to operate as single EFR32 with typical 3-Wire Wi-Fi/PTA

- Single EFR32 radio
- REQUEST unshared, active high, PC10
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as RF_ACTIVE or BT_ACTIVE (active high)
- GRANT, active low, PF3
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as WLAN_DENY (deny is active high, making grant active low)
- PRIORITY, active high, PD12
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as RF_STATUS or BT_STATUS (active high)

  - PRIORITY is static, not directional. If operated with a 3-Wire Wi-Fi/PTA expecting directional:
    - Static high PRIORITY is interpreted as high PRIORITY and always in TX mode, regardless of actual TX or RX
    - Static low PRIORITY is interpreted as low PRIORITY and always in RX mode, regardless of actual TX or RX
- REQUEST_WINDOW is 50 µs
- Disabled Abort transmission mid packet if GRANT is lost
- PRIORITY is always high
- RHO unused

The required #defines in hal-config.h are:

```
// $[COEX]
#define HAL_COEX_ENABLE                   (1)

#define BSP_COEX_REQ_PIN                  (10)
#define BSP_COEX_REQ_PORT                 (gpioPortC)
#define BSP_COEX_REQ_ASSERT_LEVEL         (1)
#define HAL_COEX_REQ_SHARED               (0)
#define HAL_COEX_REQ_BACKOFF              (15)
#define HAL_COEX_REQ_WINDOW               (50+15)

#define BSP_COEX_GNT_PIN                  (3)
#define BSP_COEX_GNT_PORT                 (gpioPortF)
#define BSP_COEX_GNT_ASSERT_LEVEL         (0)
#define HAL_COEX_TX_ABORT                 (0)

#define BSP_COEX_PRI_PIN                  (12)
#define BSP_COEX_PRI_PORT                 (gpioPortD)
#define BSP_COEX_PRI_ASSERT_LEVEL         (1)
#define HAL_COEX_PRI_SHARED               (0)
#define HAL_COEX_RX_HIPRI                 (1)
#define HAL_COEX_TX_HIPRI                 (1)
// [COEX]$
```

The logic analyzer capture in Figure 5 shows the PTA interface, Wi-Fi TX state, and EFR32 radio state for an EFR32 radio configured for typical 3-Wire Wi-Fi/PTA duing a CONNECTION event (slave):

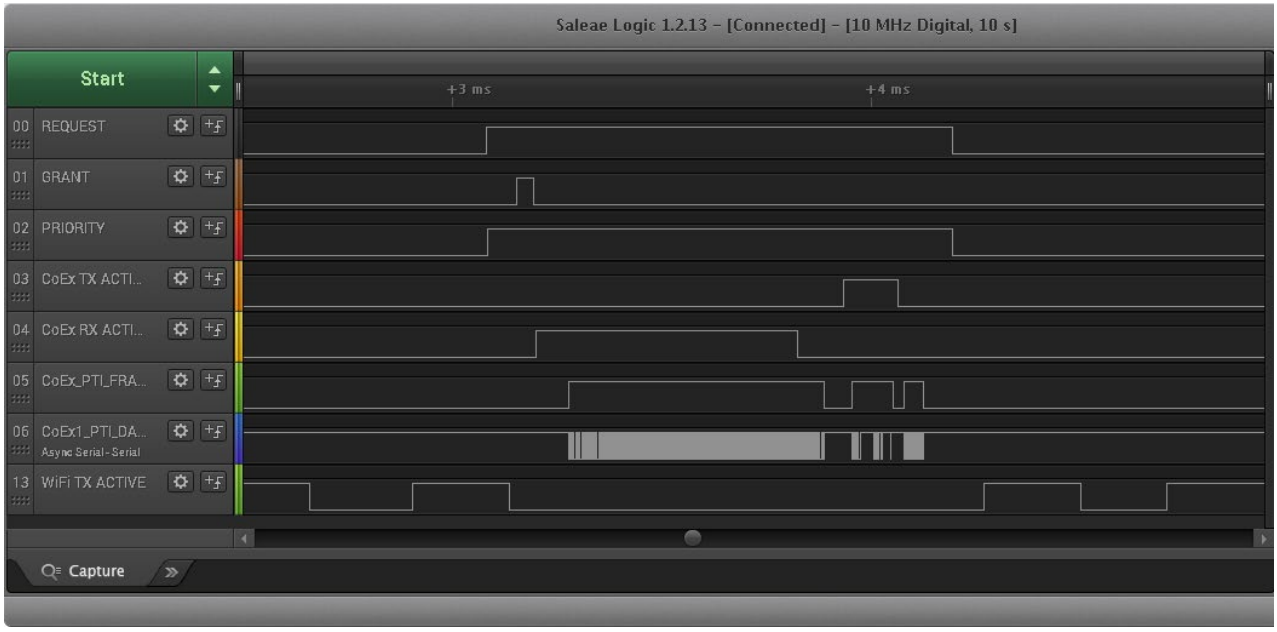


Figure 5. Example CONNECTION event (slave) for Single EFR32 typical 3-Wire Wi-Fi/PTA Logic Analyzer Capture

where:

- **REQUEST:** active high, push-pull REQUEST output
- **nGRANT:** active low GRANT input
- **PRIORITY:** active high PRIORITY output
- **CoEx TX ACTIVE:** EFR32 TX Active control signal (configured via sample code in section 5.1, Example TX_ACTIVE/RX_ACTIVE)
- **CoEx RX ACTIVE:** EFR32 RX Active control signal (configured via sample code in section 5.1, Example TX_ACTIVE/RX_ACTIVE)
- **CoEx PTI FRAME:** EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **CoEx PTI DATA:** EFR32 Frame Control Data Out signal (packet trace data)
- **WiFi TX ACTIVE:** Wi-Fi TX Active signal

The logic analyzer sequence in Figure 5 shows:

1. Wi-Fi is transmitting and EFR32BG asserts REQUEST, then high PRIORITY.
2. GRANT is momentarily deasserted by Wi-Fi/PTA, but is reasserted as Wi-Fi finished.
3. EFR32 radio enables RX mode awaiting master TX.
4. EFR32 radio receives the master TX.
5. EFR32 radio exits receive mode.
6. At start of 150µs IFS, EFR32 radio transmits back to master.
7. After transmit, EFR32 reasserts PRIORITY and then REQUEST.
8. Wi-Fi resumes transmission.

Example 2: Configure EFR32 PTA support to operate with multi-radio 2-Wire PTA with active-low REQUEST

- Multiple EFR32 radios (external 1 kΩ ±5% pull-up required on REQUEST)
- REQUEST shared, active low, PC10
- GRANT, active low, PF3
- PRIORITY unused
- REQUEST_WINDOW is 50 µs
- Disabled Abort transmission mid packet if GRANT is lost
- RHO unused

The required #defines in hal-config.h are:

```
// $[COEX]
#define HAL_COEX_ENABLE                   (1)

#define BSP_COEX_REQ_PIN                  (10)
#define BSP_COEX_REQ_PORT                 (gpioPortC)
#define BSP_COEX_REQ_ASSERT_LEVEL         (0)
#define HAL_COEX_REQ_SHARED               (1)
#define HAL_COEX_REQ_BACKOFF              (15)
#define HAL_COEX_REQ_WINDOW               (50+15)

#define BSP_COEX_GNT_PIN                  (3)
#define BSP_COEX_GNT_PORT                 (gpioPortF)
#define BSP_COEX_GNT_ASSERT_LEVEL         (0)
#define HAL_COEX_TX_ABORT                 (0)
// [COEX
```

The logic analyzer capture in Figure 6 shows the PTA interface, Wi-Fi radio state, and EFR32 radio state for an EFR32 radio configured for multi-radio 2-Wire PTA with active-low REQUEST:

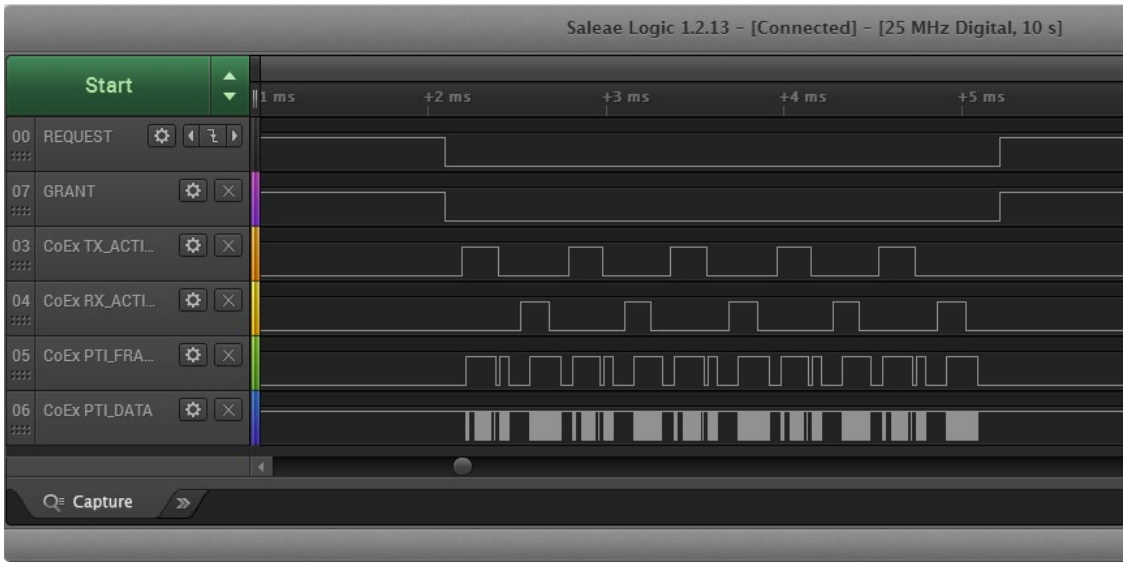


Figure 6. Example CONNECTION event (master) for Multi-EFR32 2-Wire Wi-Fi/PTA Logic Analyzer Capture (first anchor point in CONNECTION, using active-low REQUEST)

where:

- **REQUEST:** active low, shared (open-drain) REQUEST input/output
- **GRANT:** active low GRANT input
- **CoEx TX ACTIVE:** EFR32 TX Active control signal (configured via sample code in section 5.1, Example TX_ACTIVE/RX_ACTIVE)
- **CoEx RX ACTIVE:** EFR32 RX Active control signal (configured via sample code in section 5.1, Example TX_ACTIVE/RX_ACTIVE)
- **CoEx PTI FRAME:** EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **CoEx PTI DATA:** EFR32 Frame Control Data Out signal (packet trace data)

The logic analyzer sequence in Figure 6 shows:

1. At REQUEST_WINDOW before the CONNECTION event, Shared REQUEST signal is tested and found not asserted by another EFR32 radio, so EFR32 radio asserts REQUEST.
2. Wi-Fi/PTA responds with GRANT asserted.
3. At end of REQUEST_WINDOW (start of CONNECTION event), EFR32 tests GRANTS, which is asserted.
4. With GRANT asserted at start of CONNECTION event, EFR32 executes transmit.
5. After transmit is complete and before end if 150µs IFS, EFR32 enables receive to capture expected response from CONNECTION slave device.
6. EFR32 device receives device and disables receive.
7. EFR32 repeats transmit/receive for four additional cycles as part of this first anchor point.
8. After last receive, EFR32 deasserts REQUEST.
9. Wi-Fi/PTA responds with GRANT deasserted.

5 Application Code Coexistence Extensions

5.1 Example TX_ACTIVE/RX_ACTIVE

It is helpful to access the EFR32 radio state during PTA coexistence debugging. The following code examples create the TX_ACTIVE and RX_ACTIVE signals seen in the previous logic analyzer captures. This EFR32MG1P232F256GM48 example pushes TX_ACTIVE out PD10 and RX_ACTIVE out PD11. Other GPIOs can be used with changes in #defines. Consult the design-specific EFR32xG datasheet and reference manual for details on changing #defines values to other EFR32 devices and to alternate GPIOs.

```
// Enable TX_ACT signal through GPIO PD10
#define _PRS_CH_CTRL_SOURCESEL_RAC2            0x00000020UL
#define PRS_CH_CTRL_SOURCESEL_RAC2             (_PRS_CH_CTRL_SOURCESEL_RAC2 << 8)
#define _PRS_CH_CTRL_SIGSEL_RACPAEN            0x00000004UL
#define PRS_CH_CTRL_SIGSEL_RACPAEN             (_PRS_CH_CTRL_SIGSEL_RACPAEN << 0)
#define TX_ACTIVE_PRS_SOURCE PRS_CH_CTRL_SOURCESEL_RAC2
#define TX_ACTIVE_PRS_SIGNAL PRS_CH_CTRL_SIGSEL_RACPAEN

#define TX_ACTIVE_PRS_CHANNEL 5
#define TX_ACTIVE_PRS_LOCATION 0
#define TX_ACTIVE_PRS_PORT gpioPortD
#define TX_ACTIVE_PRS_PIN 10
#define TX_ACTIVE_PRS_ROUTELOC_REG ROUTELOC1
#define TX_ACTIVE_PRS_ROUTELOC_MASK (~0x00003F00UL)
#define TX_ACTIVE_PRS_ROUTELOC_VALUE PRS_ROUTELOC1_CH5LOC_LOC0 // PD10
#define TX_ACTIVE_PRS_ROUTEPEN PRS_ROUTEPEN_CH5PEN

// Enable RX_ACT signal through GPIO PD11
#define _PRS_CH_CTRL_SOURCESEL_RAC2            0x00000020UL
#define PRS_CH_CTRL_SOURCESEL_RAC2             (_PRS_CH_CTRL_SOURCESEL_RAC2 << 8)
#define _PRS_CH_CTRL_SIGSEL_RACRX              0x00000002UL
#define PRS_CH_CTRL_SIGSEL_RACRX               (_PRS_CH_CTRL_SIGSEL_RACRX << 0)
#define RX_ACTIVE_PRS_SOURCE PRS_CH_CTRL_SOURCESEL_RAC2
#define RX_ACTIVE_PRS_SIGNAL PRS_CH_CTRL_SIGSEL_RACRX

#define RX_ACTIVE_PRS_CHANNEL 6
#define RX_ACTIVE_PRS_LOCATION 13
#define RX_ACTIVE_PRS_PORT gpioPortD
#define RX_ACTIVE_PRS_PIN 11
#define RX_ACTIVE_PRS_ROUTELOC_REG ROUTELOC1
#define RX_ACTIVE_PRS_ROUTELOC_MASK (~0x003F0000UL)
#define RX_ACTIVE_PRS_ROUTELOC_VALUE PRS_ROUTELOC1_CH6LOC_LOC13 // PD11
#define RX_ACTIVE_PRS_ROUTEPEN PRS_ROUTEPEN_CH6PEN

CMU_ClockEnable(cmuClock_PRS, true); // enable clock to PRS

// Setup PRS input as TX_ACTIVE signal
PRS_SourceAsyncSignalSet(TX_ACTIVE_PRS_CHANNEL, TX_ACTIVE_PRS_SOURCE, TX_ACTIVE_PRS_SIGNAL);
// enable TX_ACTIVE output pin with initial value of 0
GPIO_PinModeSet(TX_ACTIVE_PRS_PORT, TX_ACTIVE_PRS_PIN, gpioModePushPull, 0);
// Route PRS CH/LOC to TX Active GPIO output
PRS->TX_ACTIVE_PRS_ROUTELOC_REG = (PRS->TX_ACTIVE_PRS_ROUTELOC_REG &
TX_ACTIVE_PRS_ROUTELOC_MASK) | TX_ACTIVE_PRS_ROUTELOC_VALUE;
PRS->ROUTEPEN |= TX_ACTIVE_PRS_ROUTEPEN;

// Setup PRS input as RX_ACTIVE signal
PRS_SourceAsyncSignalSet(RX_ACTIVE_PRS_CHANNEL, RX_ACTIVE_PRS_SOURCE, RX_ACTIVE_PRS_SIGNAL);
// enable RX_ACTIVE output pin with initial value of 0
GPIO_PinModeSet(RX_ACTIVE_PRS_PORT, RX_ACTIVE_PRS_PIN, gpioModePushPull, 0);
// Route PRS CH/LOC to RX Active GPIO output
PRS->RX_ACTIVE_PRS_ROUTELOC_REG = (PRS->RX_ACTIVE_PRS_ROUTELOC_REG &
RX_ACTIVE_PRS_ROUTELOC_MASK) | RX_ACTIVE_PRS_ROUTELOC_VALUE;
PRS->ROUTEPEN |= RX_ACTIVE_PRS_ROUTEPEN;
```

6 Coexistence Backplane Evaluation Board (EVB)

Silicon Labs' EFR32 coexistence solution can be evaluated by ordering an EFR32™ Mighty Gecko Wireless SoC Starter Kit (WSTK) #SLWSTK6000B and a Coexistence Backplane EVB (#SLWSTK-COEXBP). For more information, see *UG350: Silicon Labs Coexistence Development Kit (SLWSTK-COEXBP)* for details.

7 Bluetooth Test Results in Multi-EFR32 Coexistence Test with 2-Wire Wi-Fi/PTA

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

8 Conclusions

Co-located, strong Wi-Fi can have a substantial impact on Bluetooth performance. 802.15.4 performance with co-located Wi-Fi can be improved through unmanaged and managed coexistence techniques. Silicon Labs recommends the following unmanaged coexistence strategies:

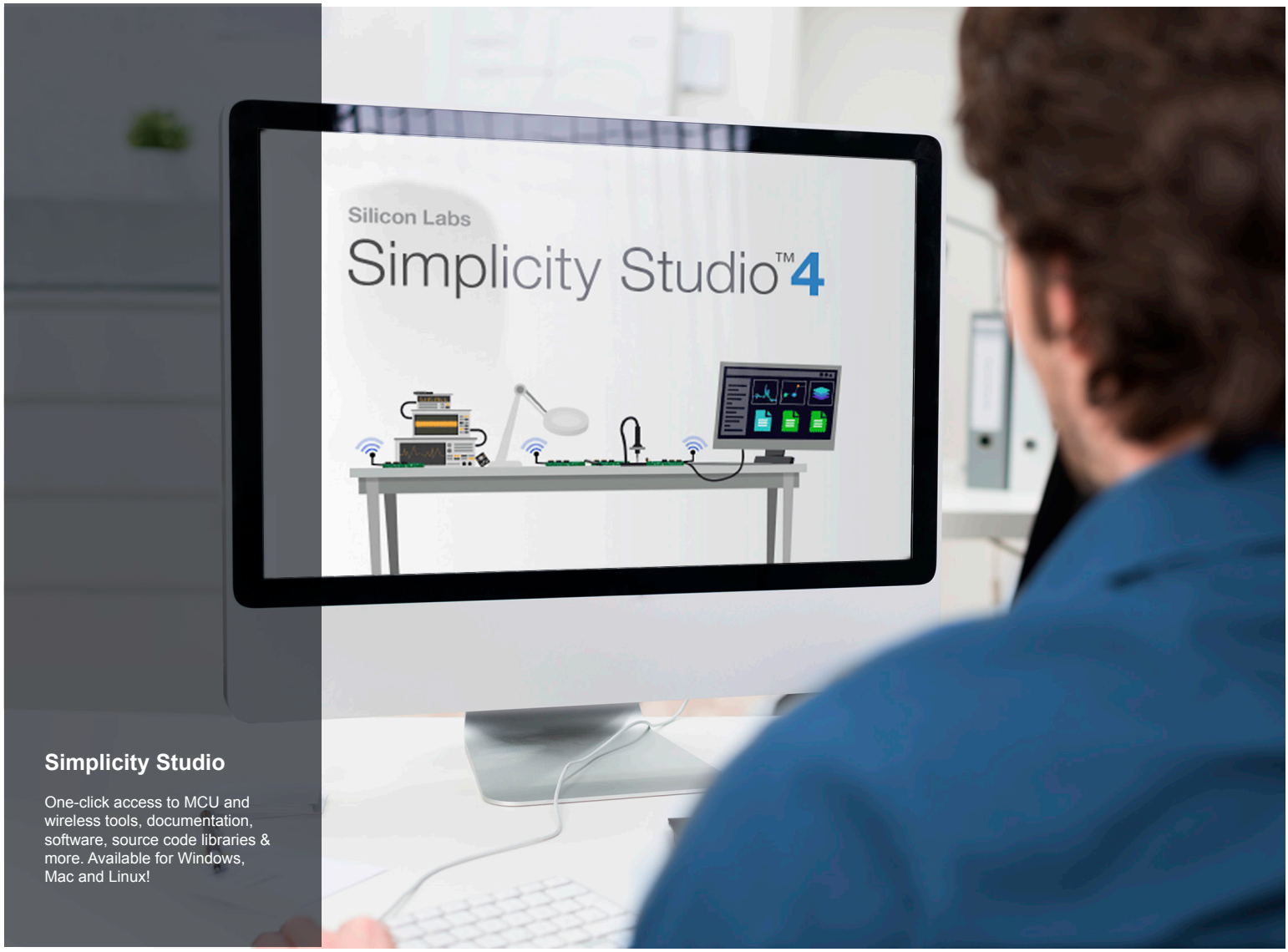
1. Implement frequency separation.
2. Operate Wi-Fi with 20.MHz bandwidth.
3. Increase antenna isolation.
4. Use Bluetooth Retry Mechanisms.
5. Remove FEM (or operate FEM LNA in bypass).

With market trends toward higher Wi-Fi TX power, higher Wi-Fi throughput, and integration of Wi-Fi and Bluetooth, and other IoT radios, into the same device, unmanaged techniques alone may prove insufficient, so that a managed coexistence solution is required. Even with a managed coexistence solution, all unmanaged coexistence recommendations are still necessary. Silicon Labs recommends the following managed coexistence strategies:

1. Wi-Fi/PTA devices providing 802.15.2-derived Packet Traffic Arbitration.
2. Silicon Labs' EFR32 PTA solution:
    1. One to four GPIOs implementing a combination of REQUEST, GRANT, PRIORITY, and RHO
    2. Supports both single-EFR32 and multi-EFR32 configurations with single Wi-Fi/PTA interface
    3. Silicon Labs' coexistence library and hal-config.h #define settings to configure EFR32 PTA support for available GPIO pins and for compatibility with the chosen Wi-Fi/PTA device
    4. Silicon Labs' API, supporting runtime PTA reconfiguration (see section 4.2.2, Run-Time PTA Re-configuration and section 4.2.3, Run-Time PTA Debug Counters

Wi-Fi/802.15.4 coexistence test results show substantial Bluetooth performance improvements when PTA is utilized:

1. Connection stability
    1. Prevent user frustration with unstable product function as Wi-Fi throughput varies.
2. Substantially reduced message failure with associated throughput improvement:
    1. Improves end-node battery life.
    2. Reduces message latency.
    3. Bluetooth remains operational, even during high Wi-Fi duty cycles.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com