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 EFR32MGx family and EFR32BGx family. Next, this application note discusses the Silicon Labs Packet Traffic Arbitration (PTA) support to coordinate 2.4 GHz RF traffic for co-located Bluetooth and Wi-Fi radios. This PTA feature set is available for the EFR32MGx series and EFR32BGx series.

This application note describes EFR32 Bluetooth coexistence support for Silicon Labs Bluetooth 3.0.0.0 and Bluetooth Mesh 1.6.3.0. See section 8 Document Revision History for a summary of key changes in previous revisions of this application note.

Additional details about the implementation of managed coexistence are included in *AN1243: Timing and Test Data for EFR32 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

The information in this document is based on the SLThread implementation of Thread. SLThread reached ‘end of service’ in December 2019. Silicon Labs is replacing SLThread with an implementation of the more popular OpenThread. We anticipate that the results from OpenThread will be very close to the SLThread results.

Table of Contents

1. Introduction

The 2.4 GHz 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 2.4 GHz radio standards can degrade performance of one or more of the radios. To improve interference robustness, each of the 2.4 GHz 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 2.4 GHz 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”

  +30 dBm 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 2.4 GHz 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 EFR32MGx family and the EFR32BGx family.

- 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 2.4 GHz RF traffic for co-located Bluetooth and Wi-Fi radios.

Notes:

1. Zigbee and Thread devices (802.15.4) operate at less than +20 dBm 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 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 3.0.0.0 and Bluetooth Mesh 1.6.3.0. Not all coexistence support features in Bluetooth 3.0.0.0 and Bluetooth Mesh 1.6.3.0 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 20/22 MHz bandwidth channels across the 2.4 GHz ISM band with transmit power levels up to +30 dBm. Similarly, Bluetooth supports 40 non-overlapping channels at 2 MHz spacing with transmit powers up to +20 dBm (Bluetooth Core Specification v5.0). For reference, 2.4 GHz Zigbee and Thread (based on IEEE 802.15.4) support 16 non-overlapping 2 MHz bandwidth channels at 5 MHz spacing with transmit powers up to +20 dBm. These Wi-Fi, Bluetooth, and Zigbee/Thread channel mappings are shown in the following figure, where yellow highlighted channels are the three Bluetooth advertising (ADV) channels.

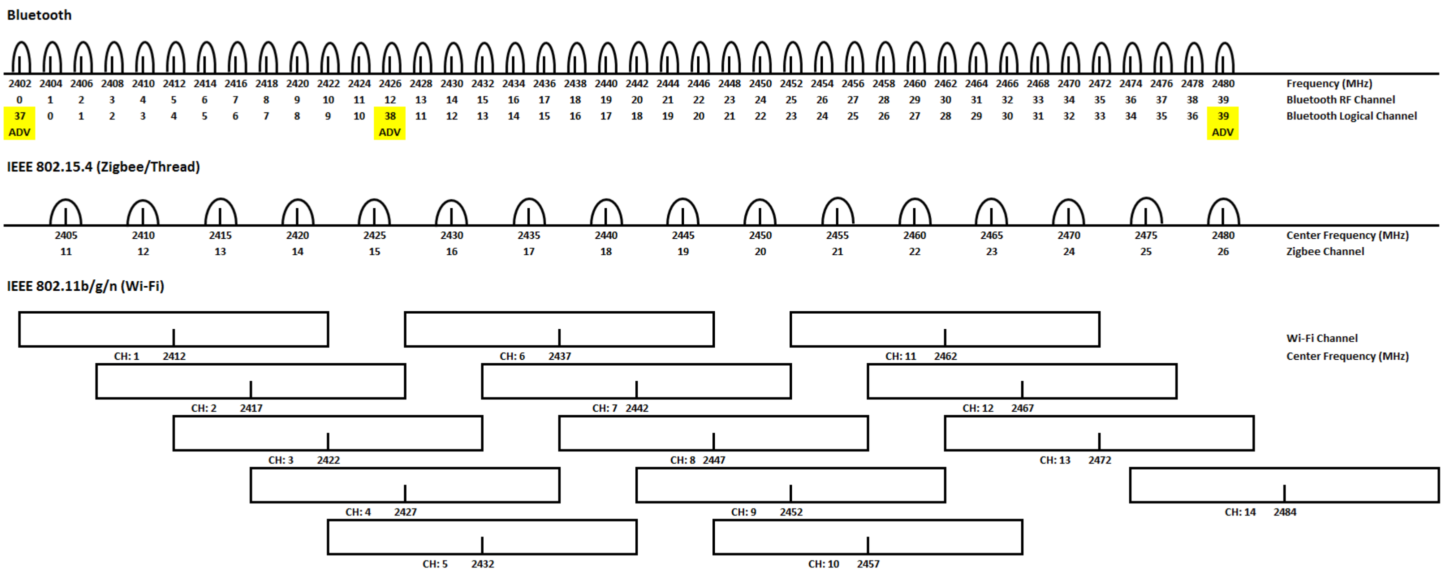


Figure 1. Bluetooth, 802.15.4, and 802.11b/g/n Channel Mapping (World-Wide)

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, 20 MHz 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" channels are shown in the following figure. All 802.11n and Bluetooth power levels are referenced to the Silicon Labs 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.

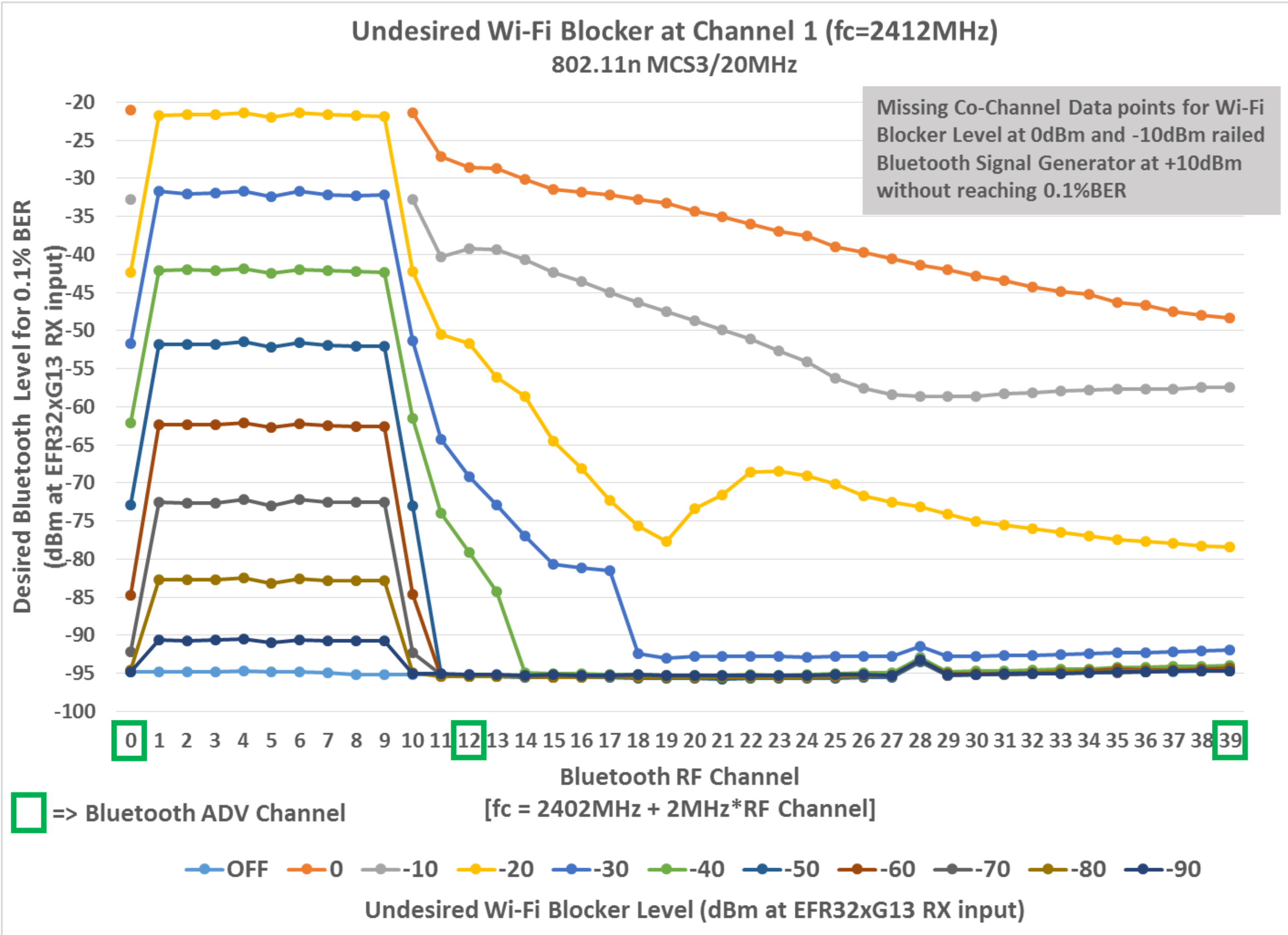


Figure 2. Bluetooth Low Energy Receive Sensitivity with 100% Duty-Cycled 802.11n (MCS3/20 MHz) Wi-Fi Blocker

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

Co-Channel (Bluetooth overlapping Wi-Fi):

- For Bluetooth RF channels 0 through 10, EFR32MG13P732F512GM48 can receive a Bluetooth 1Mbps signal at 2 dB 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 -91.8 dBm Bluetooth 1Mbps signal (RX sensitivity + 3 dB) with -50 dBm or weaker Wi-Fi transmit power (100% duty cycle).
- At Bluetooth RF channel 20, EFR32MG13P732F512GM48 can receive a -91.8 dBm Bluetooth 1Mbps signal (RX sensitivity + 3 dB) with -30 dBm 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 -91.8 dBm Bluetooth 1Mbps signal (RX sensitivity + 3 dB) with -30 dBm 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 the following figure, 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).

The following figure 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.

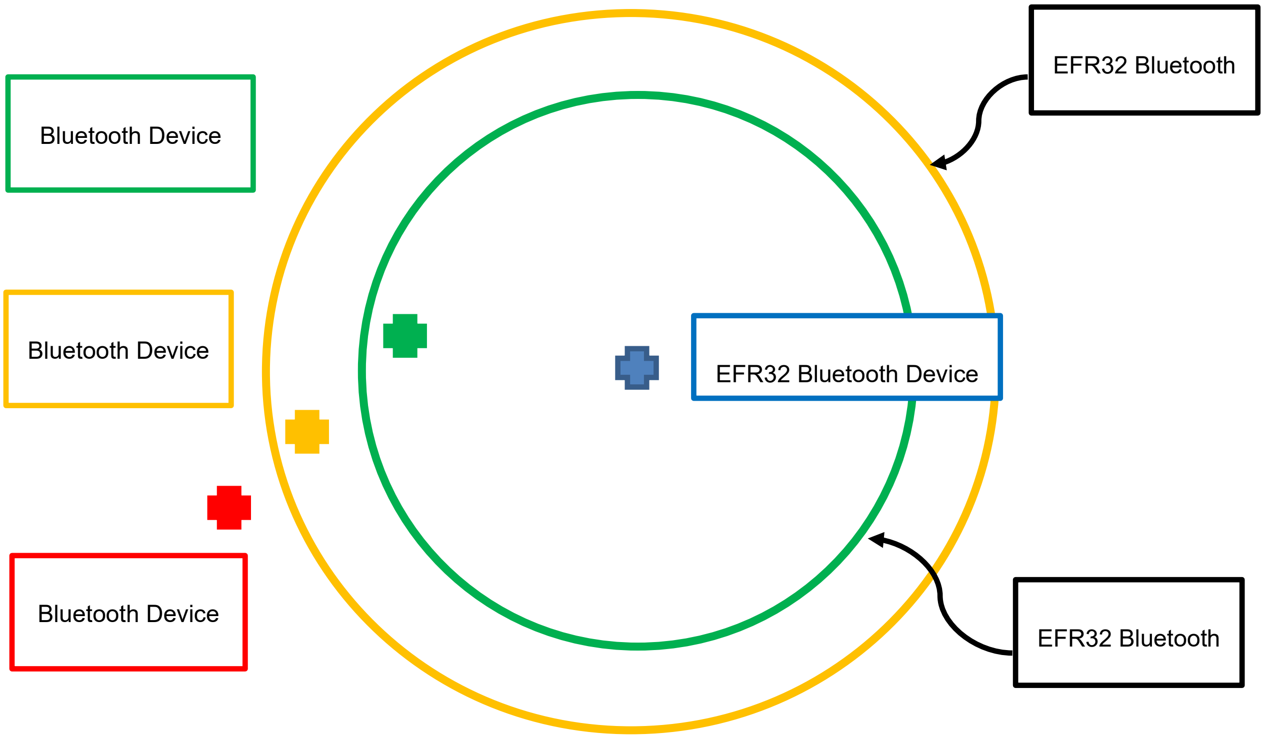


Figure 3. EFR32 Receiver Desensitized when Wi-Fi Transmitting

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 the figure above, 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 the 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:

```
sl_status_t sl_bt_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 20 MHz 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 20 MHz or 40 MHz modes. If operated in 40 MHz mode, 40 MHz of the 80 MHz ISM band is consumed by the Wi-Fi channel. However, an additional 40 MHz 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 40 MHz mode for 802.11n, the Wi-Fi standard anticipated potential issues with other 2.4 GHz ISM devices when Wi-Fi operated in 40 MHz mode. During association, any Wi-Fi station can set the Forty MHz Intolerant bit in the HT Capabilities Information. This bit informs the Wi-Fi access point that other 2.4 GHz ISM devices are present, forcing the entire Wi-Fi network to 20 MHz 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 40 MHz Intolerant bit during association to force the Wi-Fi to 20 MHz mode, increasing the number of channels available to Bluetooth and improving the Bluetooth performance.

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

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 +20 dBm transmit power and has excellent receiver sensitivity without an external Front End Module (FEM). However, an external FEM can increase transmit power to +20 dBm 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 20 MHz bandwidth.
  - Co-located Wi-Fi and Bluetooth radios can communicate pending and/or in-progress activity on 2.4 GHz 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 15-20 dB isolation is not unusual).

Assuming frequency separation achieves the “far-away” channel case and Wi-Fi only uses 20 MHz bandwidth, a +20 dBm Wi-Fi transmit power level at 100% duty cycle requires 50 dB antenna isolation to receive -92 dBm or 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 2.4 GHz 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 2.4 GHz ISM band, any Wi-Fi transmit can be delayed, improving Bluetooth message reliability.

Section 4.1 PTA Support Options discusses PTA support hardware options and section 4.5 PTA Support Software Setup discusses PTA support software setup. *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi,* available under non-disclosure from Silicon Labs technical support, provides test results for EFR32 PTA implementation under various Wi-Fi operating conditions.

Note: EFR32 Bluetooth and Bluetooth Mesh coexistence is supported in Bluetooth 2.13.x and Bluetooth Mesh 2.10.x. Not all coexistence support features in Bluetooth 2.13.x and Bluetooth Mesh 2.10.x are present in earlier versions.

4.1. PTA Support 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)

The following table describes how 802.15.2 considers radio state, transmit/receive, and frequencies.

Table 1. IEEE 802.15.2 2.4 GHz 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 the table above, 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 these Unmanaged Coexistence recommendations.

- Frequency Separation
- Operate Wi-Fi in 20 MHz 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. The following figure shows the most common Wi-Fi/PTA implementations supporting Bluetooth.

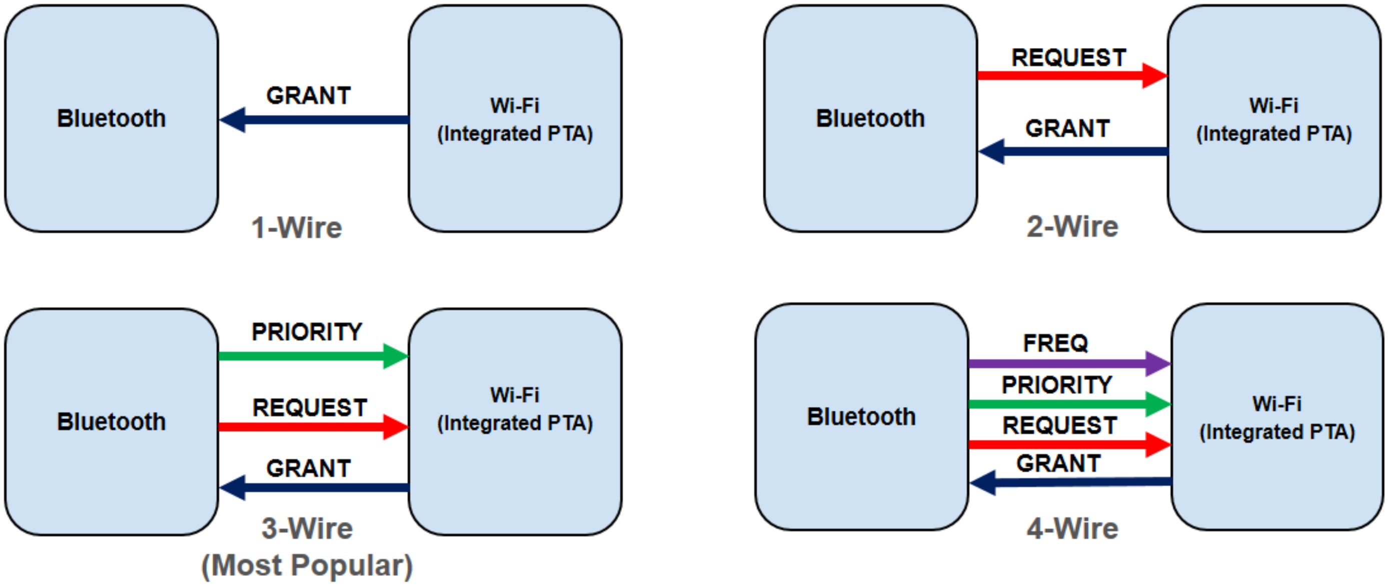


Figure 4. Typical Wi-Fi/Bluetooth PTA Implementations

4.1.1. 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 2.4 GHz 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 2.4 GHz 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.

4.1.2. 2-Wire PTA

In 2-Wire, the REQUEST signal is added, allowing the Bluetooth radio to request the 2.4 GHz 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.

4.1.3. 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.

- 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, 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.

4.1.4. 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.

Additional details about the implementation of managed coexistence and test results are available in an expanded version of this application note, *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.2. Wi-Fi/PTA Considerations

From Silicon Labs’ testing, Wi-Fi/PTA implementations vary. Best Bluetooth results are attained when the Wi-Fi/PTA implementation has the following characteristics.

4.2.1. Wi-Fi/PTA Supports Wi-Fi TX Pre-emption

Wi-Fi RF duty cycle varies based on the Modulation and Coding Scheme (MCS) index, guard interval, bandwidth, and target data rate. Wi-Fi signal strength and SNR vary with distance and objects/walls between AP and STA, particularly with mobile Wi-Fi devices. When Wi-Fi signal strength and SNR drop, the MCS index is lowered to modulations able to operate at the lower SNR. However, lowering the MSC index also lowers the maximum data rate and the Wi-Fi RF duty cycle increases as the AP and STA attempt to maintain the target data rate.

After REQUEST asserted, Wi-Fi/PTA devices not supporting Wi-Fi TX pre-emption delay GRANT to EFR32 until the end of the in-progress packet. At MCS0 with 20MHz bandwidth and large aggregated Wi-Fi packets, this delay can be as long as 16ms. This delay is much longer than the typical REQUEST_WINDOW. As such, the lack of Wi-Fi TX pre-emption during high Wi-Fi RF duty cycle increases Bluetooth retries, increases end-node battery consumption, and increases Bluetooth message loss.

After REQUEST asserted, Wi-Fi/PTA devices supporting TX pre-emption stop the Wi-Fi transmit, ramp down the Wi-Fi PA (within-gate delays up to tens of μs), and quickly GRANT the 2.4GHz band to the 802.15.4 radio. The short the time from REQUEST to quiet Wi-Fi and GRANT improves the Bluetooth message success on first attempt, which decreases Bluetooth retries, decreases end-node battery consumption, and decreases Bluetooth message loss.

Wi-Fi TX pre-emption impacts Wi-Fi performance as in-progress Wi-Fi TX packets are aborted and corrupted. In response to a corrupted packet, most Wi-Fi devices assume poor SNR or an interferer and temporarily lower the MCS index to maintain connection but recover to optimum MSC index as subsequent messages are successful.

However, most Bluetooth connections and Bluetooth mesh networks are low duty cycle and the infrequent Wi-Fi TX pre-emption events have a low impact on Wi-Fi performance.

4.2.2. Wi-Fi/PTA Prevents Wi-Fi Transmit when GRANT Asserted

Some Wi-Fi/PTA devices have been optimized to work with classic Bluetooth devices and have also been optimized to continue Wi-Fi TX events for a short time after asserting GRANT. This works only for that particular Wi-Fi/PTA and classic Bluetooth combination, as the minimum time from REQUEST asserted to Bluetooth TX start was used to determine how long Wi-Fi TX can continue after GRANT asserted. For EFR32 Bluetooth receives, the Wi-Fi/PTA device must stop all Wi-Fi TX when GRANT asserted and/or the EFR32 REQUEST_WINDOW must be increased to equal or exceed the Wi-Fi/PTA continued TX event.

Some Wi-Fi/PTA devices also attempt Wi-Fi TX based on the FREQ feature described in the 4-Wire PTA. If the FREQ feature is set by GPIO pin (or Wi-Fi/PTA internal register) to different frequencies, Wi-Fi/PTA will assert GRANT, but continue to Wi-Fi transmit. Given co-located Wi-Fi and Bluetooth radios with insufficient antenna isolation, the FREQ feature must be set to always blocking, FREQ GPIO pin (or Wi-Fi/PTA internal register) held asserted.

4.2.3. Wi-Fi/PTA and Application Implement Reasonable Prioritization

The Bluetooth connection interval and window are fixed. If co-located Bluetooth misses too many connection intervals, the connection is dropped and must be re-established. Frequent drops are unacceptable to end-users. Silicon Labs has implemented priority levels and priority escalation within EFR32. When EFR32 Bluetooth asserts high PRIORITY, the Wi-Fi/PTA devices should GRANT to EFR32, except for only highest priority Wi-Fi traffic.

4.2.4. Wi-Fi/PTA Implements Aggregation

Ensure aggregation is enabled on the Wi-Fi/PTA device. During suitable conditions, aggregation combines multiple smaller packets into fewer larger packets, reducing total packet overhead. As such, aggregation aids Wi-Fi and Bluetooth performance as Wi-Fi TX idle periods combine into larger idle periods for freer RF airtime and improved Bluetooth mesh reception.

4.2.5. Multiple EFR32s Connected to Wi-Fi/PTA

The 802.15.2 recommendation only addresses a single 802.11b radio connected to a single 802.15.1 radio. However, market trends are requiring multiple co-located 2.4GHz ISM radios (Bluetooth, Zigbee, and Thread) to operate with a Wi-Fi/PTA device only designed for one external radio. Silicon Labs has addressed this requirement by enhancing the REQUEST signal with the “shared” REQUEST feature implemented.

The “shared” REQUEST feature has the following characteristics:

- Operate REQUEST in open-drain or open-source (an external 1kΩ ±5% pull-up for open-drain or pull-down for open source is required).
- Before asserting REQUEST, test REQUEST to determine if another EFR32 has already asserted REQUEST.
- If not already asserted, assert REQUEST.
- If already asserted:
  - Wait for REQUEST to deassert.
  - Delay random time (programmable).
  - Re-test REQUEST.
  - If not already asserted, assert REQUEST.
  - If already asserted:
    - Then another radio has secured REQUEST.
    - Return to wait for REQUEST to deassert.

With these enhanced REQUEST features, multiple EFR32s implement a PTA interface, which to the Wi-Fi/PTA device appears to be a single external radio.

An example of three EFR32 radios (Zigbee, Thread, and BLE) using 2-Wire mode interfaced to one Wi-Fi/PTA interface using 2-Wire mode is shown in the following figure.

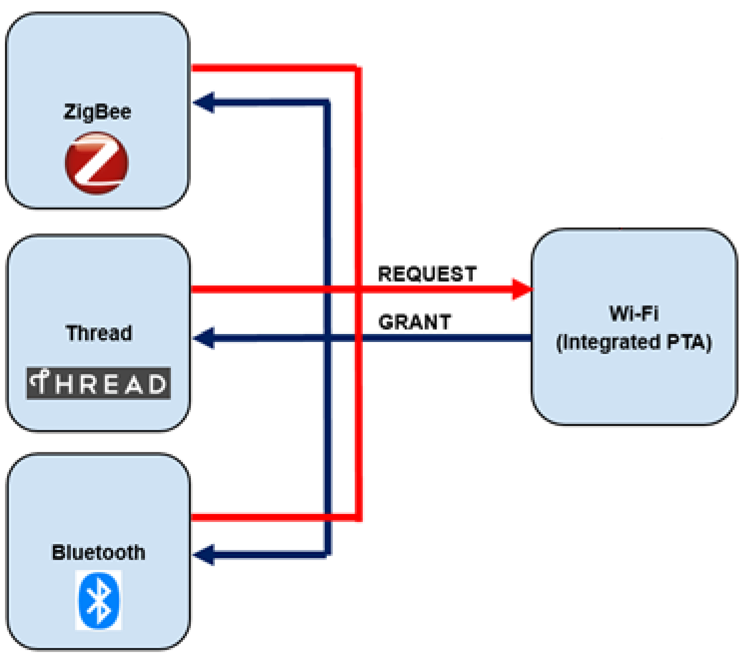


Figure 5. Three EFR32s Supporting Wi-Fi/PTA with a Single 2-Wire PTA Interface
(REQUEST pull-up or pull-down not shown)

With these enhanced REQUEST features and "shared" PRIORITY, an example of three EFR32 radios (Zigbee, Thread, and BLE) using 3-Wire mode interfaced to one Wi-Fi/PTA interface using 3-Wire mode is shown in the following figure.

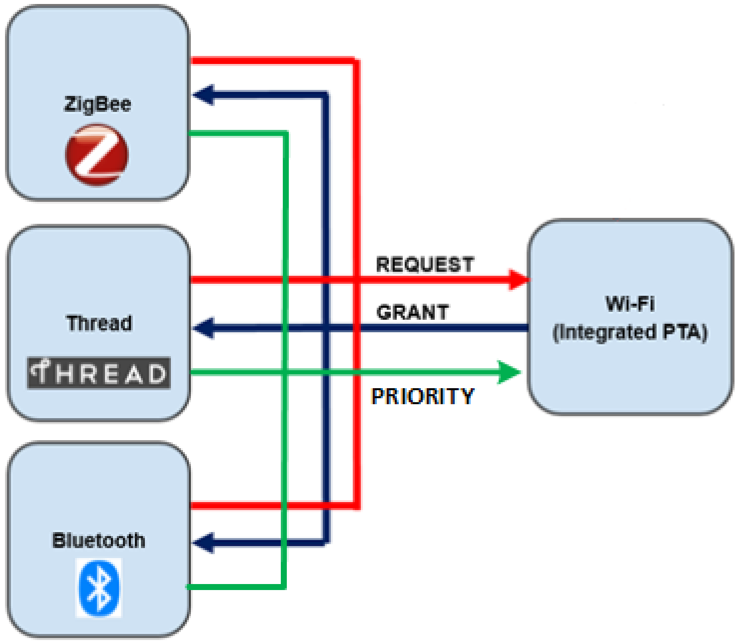


Figure 6. Three EFR32s Supporting Wi-Fi/PTA with a Single 3-Wire PTA Interface
(REQUEST and PRIORITY pull-ups or pull-downs not shown)

4.3. PWM for High Duty Cycle Wi-Fi

PWM is a feature where the EFR32 reserves a timed slot for REQUEST. This is important in high duty-cycle Wi-Fi because planned EFR32 TX and RX, allows Wi-Fi to be more amenable to GRANT airtime when it is expected. Note that GRANT is not guaranteed; Wi-Fi still makes the decision based on its discretion.

100% Passive SCAN and Bluetooth mesh have no knowledge of when an incoming packet will arrive. As the co-located Wi-Fi TX RF duty cycle increases, the Wi-Fi TX idle periods become smaller and there are fewer idle periods allowing Passive SCAN or Bluetooth mesh receives. As such, the probability of a packet arriving during an acceptably large Wi-Fi TX idle period decreases significantly.

The PWM feature implements regular PTA REQUESTs to interrupt the high Wi-Fi RF duty cycle and ensure adequate windows for receives. However, this feature does degrade Wi-Fi performance. Additionally, the exact PWM period and duty cycle must be carefully selected to not alias PWM period with Wi-Fi beacons to avoid collapsing the Wi-Fi network.

Note: Ensure aggregation is enabled on Wi-Fi/PTA device. During suitable conditions, aggregation combines multiple smaller packets into fewer larger packets which reduces the total packet overhead. As such, aggregation aids Passive SCAN and Bluetooth mesh ADV-Bearer packet detection as Wi-Fi TX idle periods combine into larger idle periods.

4.3.1. Background

The following figure is a 15.485ms capture of a Wi-Fi TX Active signal while Wi-Fi is transmitting iperf TCP at maximum throughput. Within this waveform, there are multiple Wi-Fi TX idle periods of which the first group of idle periods is expanded.

Figure 7. High-Duty Cycle Wi-Fi TX Active

If a Bluetooth packet were 160us, from the first group, only the 0.306ms idle period is sufficient for packet receive with a 0.146ms detection window (0.306ms–0.160ms). The following table lists all Wi-Fi TX idle period durations in the above 15.485ms capture.

Table 2. High-Duty Cycle Wi-Fi TX Active Durations

Wi-Fi TX Idle (ms)	Detection Window (Idle–0.160ms) (ms)
0.306	0.146
0.113	0.000
0.061	0.000
0.246	0.086
0.077	0.000
0.260	0.100
0.061	0.000
0.065	0.000
0.016	0.000
0.065	0.000
0.016	0.000
0.065	0.000
0.016	0.000
0.245	0.085
0.105	0.000
0.098	0.000
0.016	0.000
0.171	0.011

The sum of all Wi-Fi TX Idle periods is 2.002ms. This Wi-Fi TX Activity indicates 87.1% duty cycle [(15.485–2.002) / 15.485 x 100%], but this calculation does not include the additional Wi-Fi RX activity.

The sum of all Detection Windows is 0.428ms, providing only 2.8% probability of receiving the packet [0.428 / 15.485 x 100%]. But again, this calculation does not include Wi-Fi RX Activity. With only 2.8% probability of packet reception and a target 1% or less BT Mesh message loss, 165 retries are required [CEILING(LOG(1%)/LOG(100%–2.8%)]. This large number of required retries is far beyond the Bluetooth mesh network retries and, even if retries are extended, the message latency becomes impractical. Large retries also quickly degrade lifetime for battery power end-nodes.

To resolve this issue, Bluetooth radios need more RF airtime to listen for incoming packets without being blocked by co-located Wi-Fi TX. Two options are possible:

- Co-located Wi-Fi device enforces a maximum allowed Wi-Fi TX RF duty cycle.
- Co-located Bluetooth radio asserts REQUEST at high PRIORITY to force regular quiet Wi-Fi TX.

Silicon Labs has not found option #1 to be reliable over various Wi-Fi/PTA devices. Silicon Labs has developed a PWM extension feature to support option #2. This PWM feature can be applied to any use case where EFR32 incoming Bluetooth Passive SCAN or Bluetooth mesh packet receive is impaired by a co-located, controllable, external device. This includes, but is not limited to, the following use cases:

- High duty-cycle co-located Wi-Fi TX
- Bluetooth co-channel with Wi-Fi
- External T/R switch requiring Bluetooth to share a single antenna with Wi-Fi

4.3.2. PWM Feature Description

The PWM feature is used to periodically stop co-located Wi-Fi TX activity to insure sufficient idle TX time windows to receive Bluetooth packets during Passive SCAN and Bluetooth mesh ADV-Bearer receive. The PWM feature does not require any additional GPIOs for single EFR32 designs. However, multi -EFR32 designs require an additional GPIO to be available for each EFR32.

The optimum PWM period and duty cycle can vary based on Bluetooth operation and Wi-Fi activity. If Wi-Fi activity is low, PWM could be disabled and fall back to normal PTA activity as described in section 4.2 Wi-Fi/PTA Considerations. During very high Wi-Fi duty cycle for Bluetooth mesh ADC-Bearer, a PWM programmed to 39ms period and >44% duty is effective in reducing message loss to less than 1%. As noted, Wi-Fi throughput is impacted and, under 39ms/>44%% condition, the high duty cycle Wi-Fi TX throughput noted above dropped ~50%.

4.3.3. PWM Implementation

The optimum PWM implementation varies with single-EFR32 vs. multi-EFR32 applications. This implementation difference is primarily due to the REQUEST sharing feature used in multi-EFR32 applications. However, if addressed during circuit board design, this difference is easily resolved.

Use of an BGAPI or CLI command are needed to allow the host application to control the PWM period, duty cycle, and priority.

4.3.3.1 Multi-EFR32 Radio with PWM

For multi-EFR32 radio applications not implementing PWM, the Shared REQUEST signal is used to arbitrate which EFR32 radio has access to the 2.4GHz ISM band when GRANT is asserted from the Wi-Fi/PTA. If the PWM signal were to be applied to the shared REQUEST pin, other EFR32 radios would interpret the REQUEST as busy when it might just be forcing Wi-Fi TX idle to allow any of the IoT radios to detect an incoming packet. Instead, the Shared REQUEST feature is accommodated by separating the Shared REQUEST signal from the REQUEST||PWM signal resulting in a separate non-Wi-Fi/PTA signal connection between the EFR32 devices for Shared REQUEST.

The PWM enabled Wi-Fi/PTA connections between the Wi-Fi and Zigbee, and BLE EFR32 radios are the same as the multi-EFR32 radio with a typical 3-Wire PTA schema. Only one EFR32 radio—selected at run time—controls the PWM and ORs the PWM signal in software with its own REQUEST (arbitrated by the Shared REQUEST signal). The other EFR32 radios must also implement PWM in Hardware Configurator to provide their REQUEST to the PTA interface (also arbitrated by Shared REQUEST). Each EFR32 radio’s REQUEST||PWM GPIO is set to open-drain/open-source, allowing a hardware OR function of any EFR32 radio’s REQUEST with REQUEST||PWM.

If the radio driving REQUEST||PWM needs to go off-line (for example, for a firmware update), its PWM outputs can be halted and another EFR32 radio’s PWM outputs can then be activated for continued PWM operation. In the following figure, the Zigbee radio implements the PWM feature, the Silicon Labs Thread radio, and Bluetooth radio provide PWM backup.

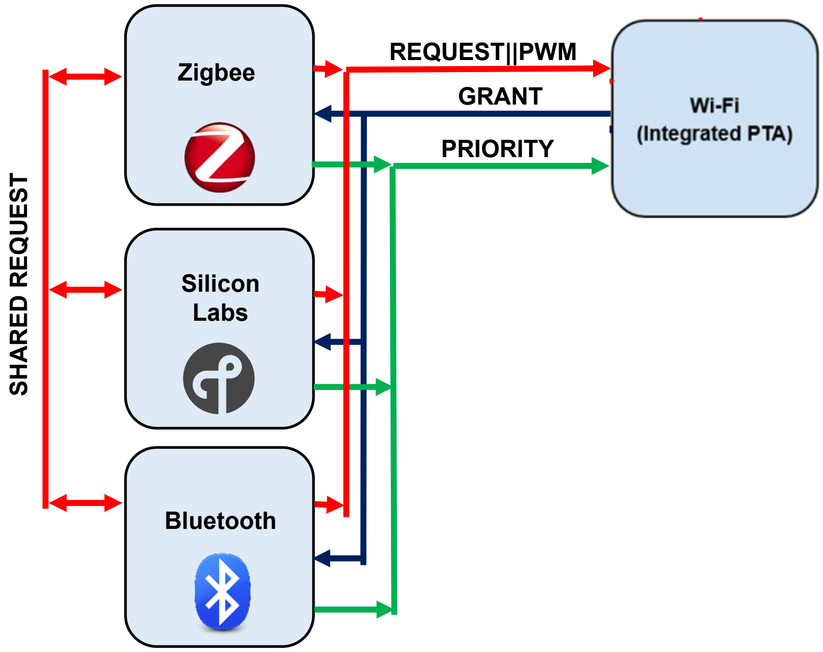


Figure 8. Three EFR32s Supporting PWM and Wi-Fi/PTA with Single 3-Wire PTA Interface
(REQUEST and PRIORITY pull-ups and pull-downs not shown)

4.4. Directional PRIORITY

Directional PRIORITY provides radio state information on the same signal line as PRIORITY by a timed pulse indicating a priority transaction, followed by a radio state indicating to the Wi-Fi part that the EFR32 is transmitting or receiving.

PRIORITY signal must be enabled for implementing **Directional PRIORITY**.

Directional PRIORITY requires the use of an additional GPIO, one Timer, and either four PRS channels (EFR32xG2x) or six PRS channels (EFR32xG1x). When **Directional PRIORITY** is enabled, the designer will need to verify the Timer and PRS channels selected for **Directional PRIORITY** are not used by the protocol SDK stack, plugins, or custom application code.

In addition, RACLNAEN and REQUEST is either one PRS channel which inverts ORs in one operation (EFR32xG2x) or two separate PRS channels (EFR32xG1x).

4.4.1. Single-EFR32 PTA with Directional PRIORITY

The following figures show the GPIO, Radio, and Timer signals for Single-EFR32 PTA with or without SDK PWM and with Directional PRIORITY.

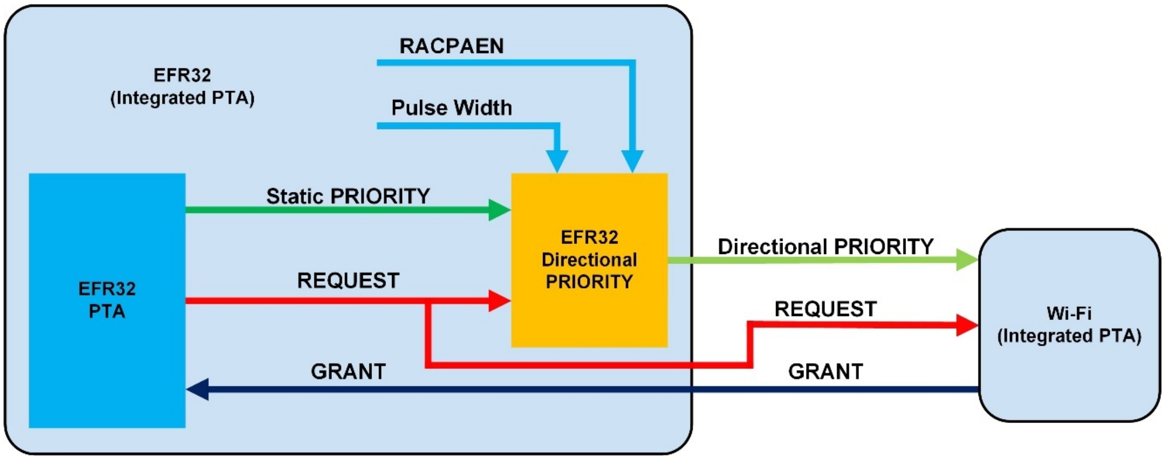


Figure 9. Single-EFR32 PTA without SDK PWM, with Directional PRIORITY

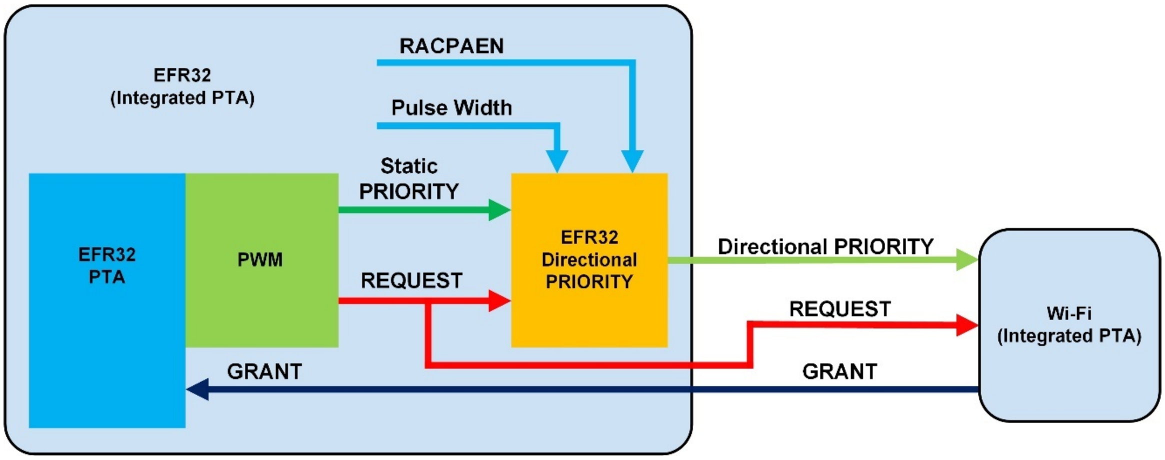


Figure 10. Single-EFR32 PTA with SDK PWM, with Directional PRIORITY

Directional PRIORITY

- EFR32 Directional Priority output
- GPIO connects to Wi-Fi PTA.
- Requires active high Static PRIORITY and active high REQUEST.

Static PRIORITY

- EFR32 PTA output
- Directional PRIORITY input
- The active high PRIORITY is not assigned to a GPIO.
- Not connected to any external circuit.

REQUEST

- EFR32 PTA output
- Directional PRIORITY input
- Compared in Pulse Width Timer.
- GPIO connects to Wi-Fi PTA.

GRANT

- EFR32 PTA input
- GPIO connects to Wi-Fi PTA.

RACPAEN

- EFR32 radio transmit PA enable output.
- Directional PRIORITY input

Pulse Width

- EFR32 Timer compared with REQUEST GPIO.
- Directional PRIORITY input

4.4.2. Multi-EFR32 PTA with Directional PRIORITY

The following figure shows the GPIO, Radio, and Timer signals for Multi-EFR32 PTA with SDK PWM and Directional PRIORITY.

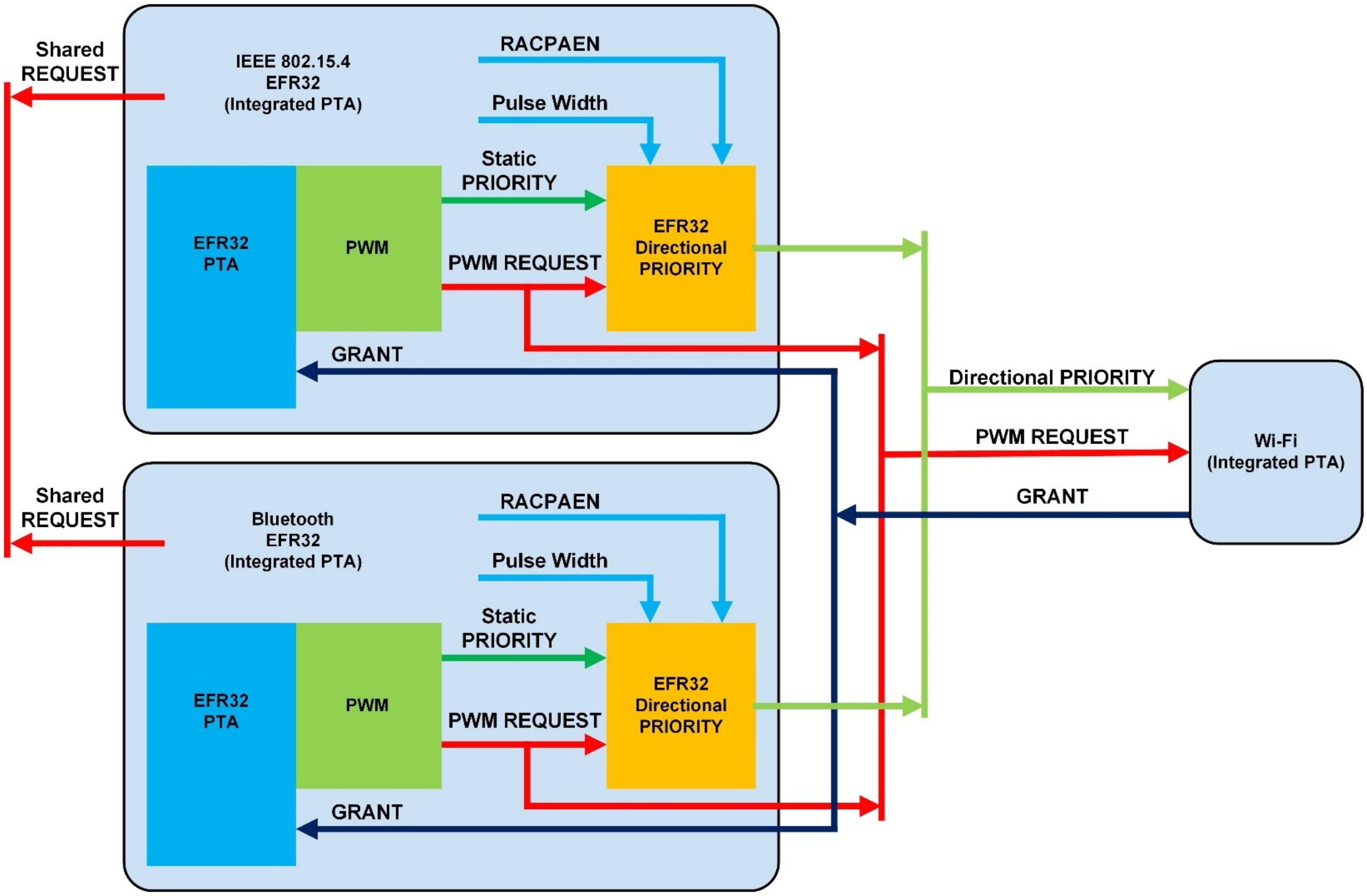


Figure 11. Multi-EFR32 PTA with SDK PWM and with Directional PRIORITY (1.5K pull-ups and pull-downs not shown)

Shared REQUEST

- EFR32 PTA input / output
- GPIO connects to all EFR32s for PTA bus arbitration between EFR32s.
  - Configured as open source / drain
  - The active high Shared REQUEST is open-source and an external 1 kΩ ±5% pull-down is required.

Directional PRIORITY

- EFR32 Directional Priority output
- GPIO connects to Wi-Fi PTA and all EFR32s.
  - Configured as open source / drain.
    - The active high Directional PRIORITY is open-source and an external 1 kΩ ±5% pull-down is required.

Static PRIORITY

- EFR32 PTA output
- Directional PRIORITY input
- The active high PRIORITY is not assigned to a GPIO.
- Not connected to any external circuit.

PWM REQUEST

- EFR32 PTA output
- Directional PRIORITY input
- GPIO connects to Wi-Fi PTA and all EFR32s.
  - Configured as open source / drain.
    - If active high, PWM REQUEST is open-source and an external 1 kΩ ±5% pull-down is required.
  - Compared with Pulse Width in EFR32 Timer.

GRANT

- EFR32 PTA input
- GPIO connects to Wi-Fi PTA and all EFR32s.

RACPAEN

- EFR32 Radio transmit PA enable output
- Directional PRIORITY input

Pulse Width

- EFR32 Timer compared with PWM REQUEST GPIO.
- Directional PRIORITY input.

The following figure shows the PRS and Timer logic diagram for Directional PRIORITY for EFR32xG1x.

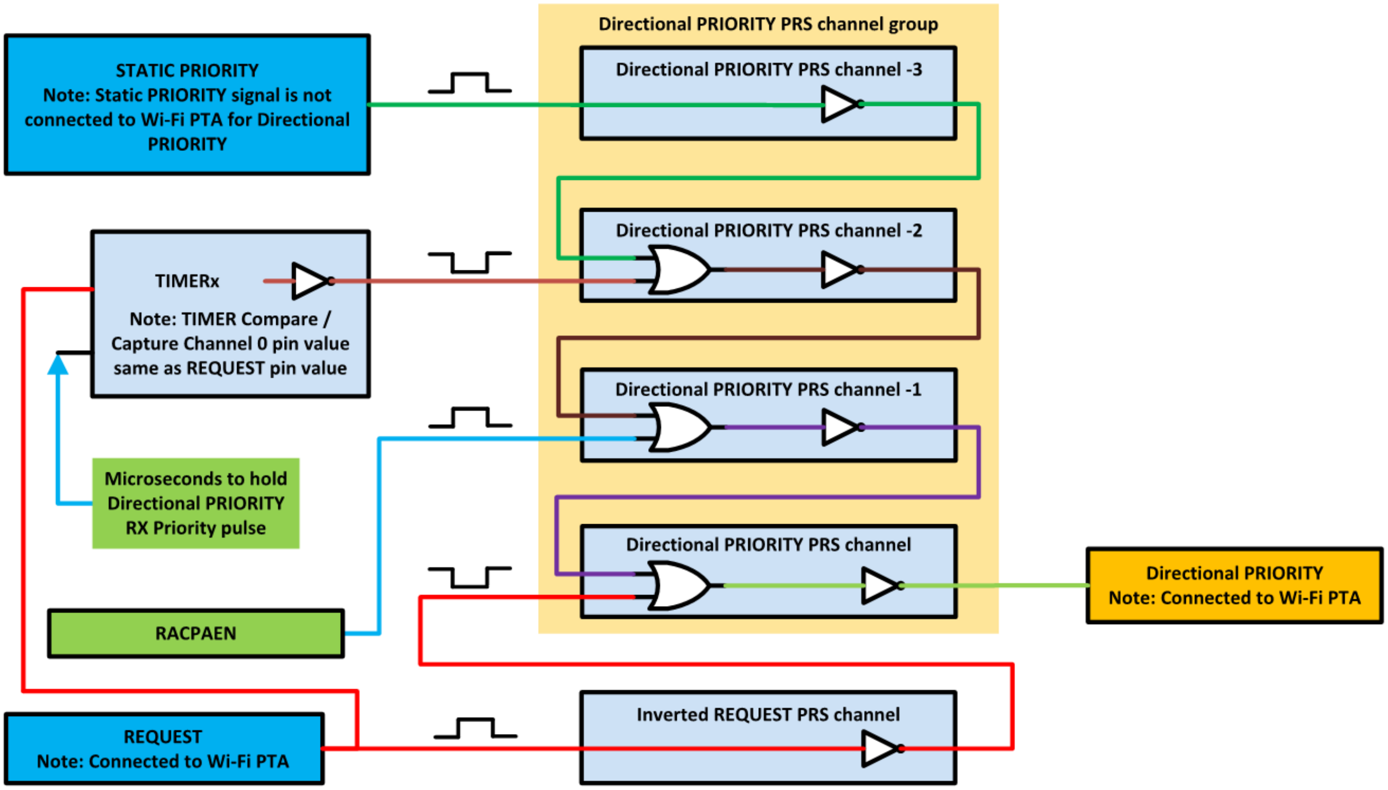


Figure 12. EFR32xG1x Directional PRIORITY Logic Diagram

The following figure shows the PRS and Timer logic diagram for Directional PRIORITY for EFR32xG2x.

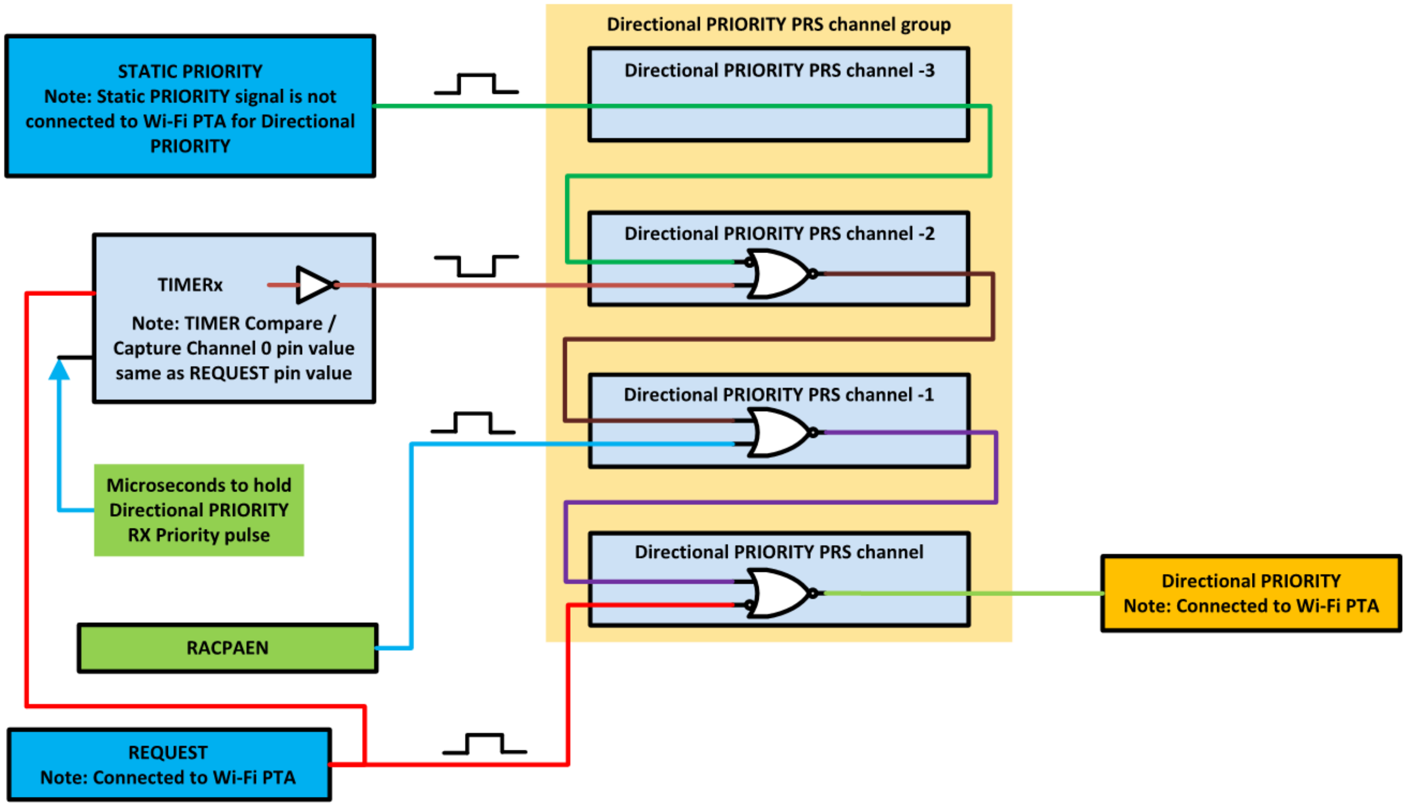


Figure 13. EFR32xG2x Directional PRIORITY Logic Diagram

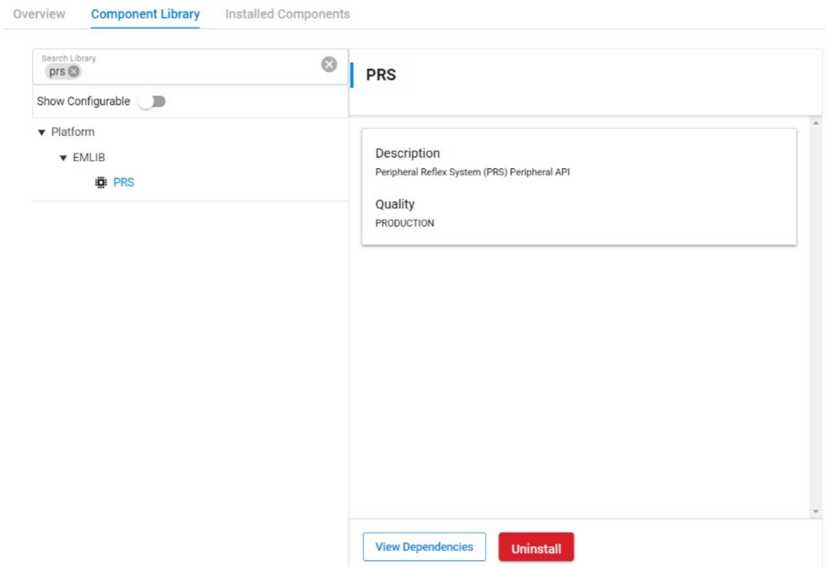
4.5. PTA Support Software Setup

Note: GPIO interrupt numbers are based on the GPIO pin numbers and not the port. This can cause conflicts if the same pin is selected for different ports—for example, PD15 will conflict with PB15. Silicon Labs recommends avoiding these conflicts. If the conflict exists in hardware, manual macros can be added with the assistance of Silicon Labs Support.

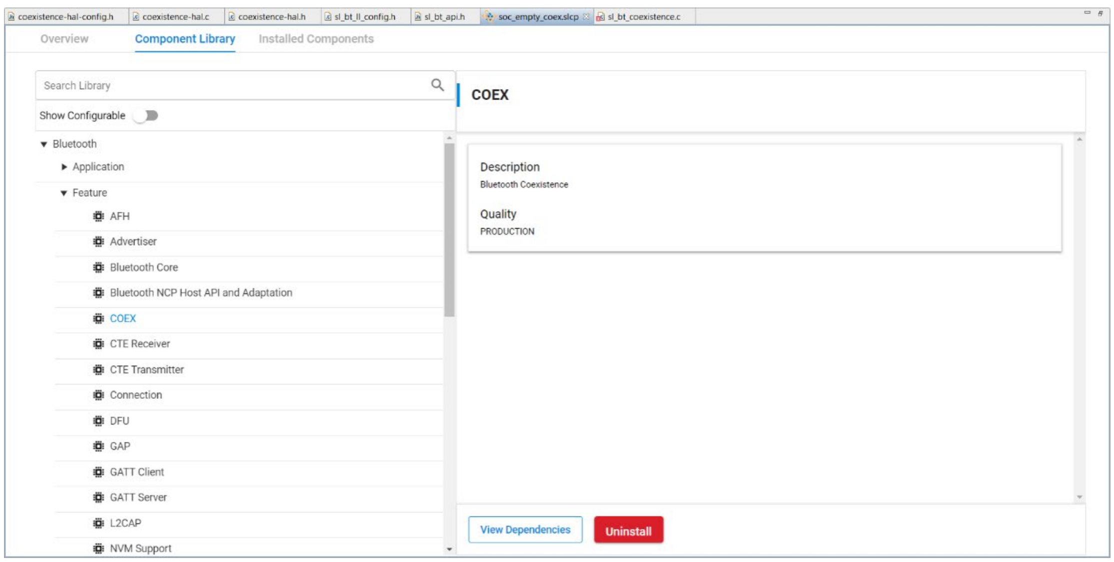
4.5.1. Compile Time PTA Setup and Defaults

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

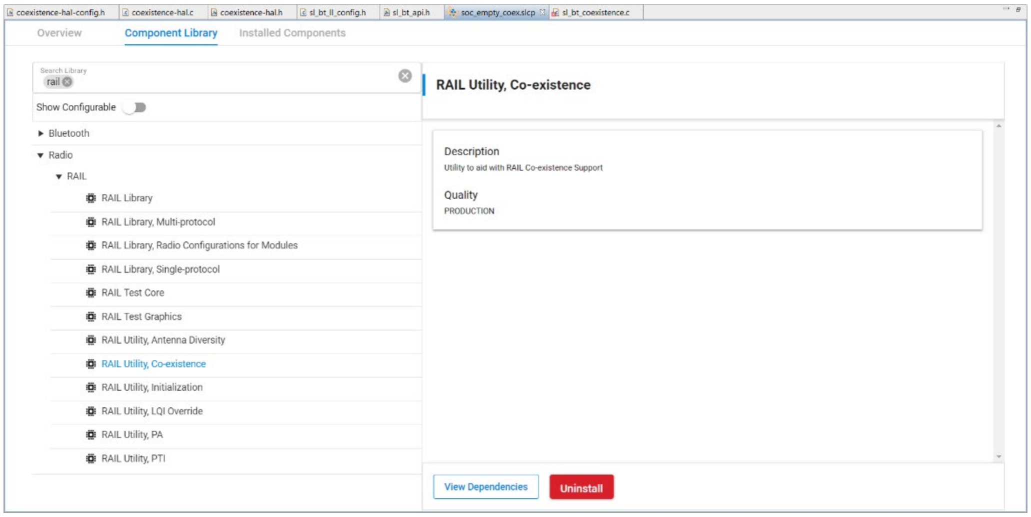
1. Create Bluetooth or Bluetooth Mesh project in Simplicity Studio.
2. Add the PRS component to your project. Double-click the slcp file of your project and then browse to the Component Library menu. From the Component Library menu, select **PRS** and click **Install.**



3. Enable the COEX component to your project. Double-click the slcp file of your project and then browse to the Component Library menu. From the Component Library menu, select **COEX** under the Bluetooth/Feature menu, and click **Install**.



4. Enable the RAIL Utility, Co-existence component to your project. Double-click the slcp file of your project and then browse to the Component library menu. From the Component Library menu, select **RAIL Utility. Co-existence** under RAIL/RAIL Library and click **Install**.



5. Create the following project folders:
   - <project>\platform\radio\rail_lib\plugin\coexistence\common
   - <project>\platform\radio\rail_lib\plugin\coexistence\hal\efr32
   - <project>\platform\radio\rail_lib\plugin\coexistence\protocol\ble
   - <project>\platform\emdrv\gpiointerrupt\src
   - <project>\protocol\bluetooth\inc\
   - <project>\protocol\bluetooth\src\
6. Copy contents from following 3.0 Bluetooth SDK folders to new project folders.

   From: SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.0\platform\radio\rail_lib\plugin\coexistence\common
   To: <project>\platform\radio\rail_lib\plugin\coexistence\common

   From: SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.0\platform\radio\rail_lib\plugin\coexistence\hal\efr32
   To: <project>\platform\radio\rail_lib\plugin\coexistence\hal\efr32

   From: SimplicityStudio\ v5\developer\sdks\gecko_sdk_suite\v3.0\platform\emdrv\gpiointerrupt\src
   To: <project>\platform\emdrv\gpiointerrupt\src

   From: SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.0\ protocol\bluetooth\inc
   To: <project>\ protocol\bluetooth\\inc
   Note: Only copy coexistence_ll-ble.h and coexistence-ble.h

   From: SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.0\ protocol\bluetooth\src
   To: <project>\ protocol\bluetooth\src
   Note: Only copy sl_bt_coexistence_counters.c and sl_bt_coexistence.c.

   From: SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.0\platform\radio\rail_lib\plugin\coexistence\protocol\ble
   To: <project>\ platform\radio\rail_lib\plugin\coexistence\protocol\ble
   Note: Only copy coexistence-ble.h and coexistence-ble.c

7. Edit project includes to include additional paths to files added to project:
   1. Right-click on project and select **Properties**.
   2. Expand **C/C++ General** and select **Paths and Symbols**.
   3. Select **Includes** tab.

4. In languages, select desired C compiler (for example, **GNU C**).
5. Press **add**, check **Is a workspace path**, and enter each of the following paths:

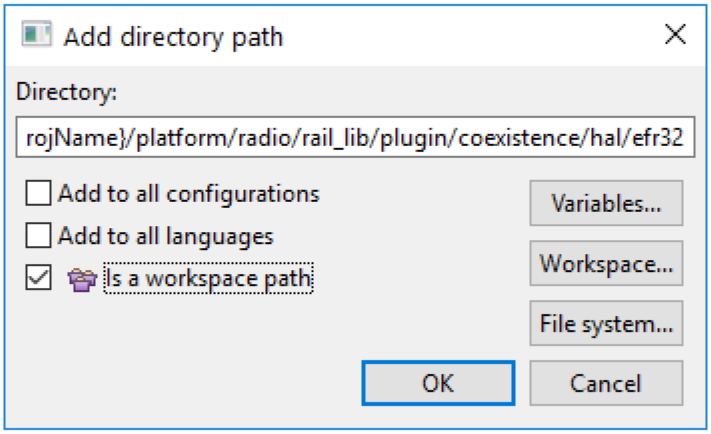
   /${ProjName}/platform/radio/rail_lib/plugin

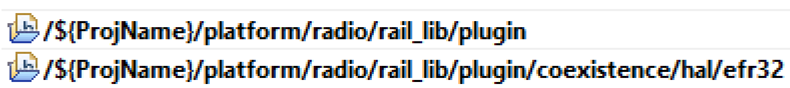
   /${ProjName}/platform/radio/rail_lib/plugin/coexistence/hal/efr32

   /${ProjName}/platform/radio/rail_lib/plugin/coexistence/protocol/ble

   /${ProjName}/protocol/bluetooth/inc

   For example:





6. Then press “OK” and “Yes” to rebuild index now

8. Add the following #defines to coexistence-hal-config.h:

```
#define HAL_COEX_OVERRIDE_GPIO_INPUT              (1)
#define HAL_COEX_ENABLE                           (1)
#define HAL_COEX_DP_TIMER                         (HAL_TIMER_TIMER1)
#define HAL_COEX_PWM_PRIORITY                     (0)
#define HAL_COEX_RETRYRX_ENABLE                   (0)
#define HAL_COEX_MAC_FAIL_THRESHOLD               (0U)
#define HAL_COEX_REQ_WINDOW                       (500U)
#define HAL_COEX_PWM_DEFAULT_ENABLED              (0)
#define HAL_COEX_TX_ABORT                         (0)
#define BSP_COEX_RHO_ASSERT_LEVEL                 (1)
#define HAL_COEX_REQ_SHARED                       (0)
#define HAL_COEX_DP_PULSE_WIDTH_US                (20U)
#define BSP_COEX_PRI_ASSERT_LEVEL                 (1)
#define HAL_COEX_TX_HIPRI                         (1)
#define BSP_COEX_REQ_ASSERT_LEVEL                 (1)
#define HAL_COEX_RX_HIPRI                         (1)
#define BSP_COEX_GNT_ASSERT_LEVEL                 (1)
#define HAL_COEX_CCA_THRESHOLD                    (4U)
#define HAL_COEX_PWM_REQ_DUTYCYCLE                (20U)
#define HAL_COEX_RETRYRX_HIPRI                    (1)
#define HAL_COEX_RETRYRX_TIMEOUT                  (16U)
#define HAL_COEX_REQ_BACKOFF                      (15U)
#define HAL_COEX_PRI_SHARED                       (0)
#define HAL_COEX_DP_ENABLED                       (1)
#define HAL_COEX_PRIORITY_ESCALATION_ENABLE       (1)
#define HAL_COEX_ACKHOLDOFF                       (1)
#define HAL_COEX_PWM_REQ_PERIOD                   (78U)
#define HAL_COEX_PHY_ENABLED                      (0)
#define BSP_COEX_PWM_REQ_ASSERT_LEVEL             (1)
```

```
#define HAL_COEX_RUNTIME_PHY_SELECT                    (1)

#define BSP_COEX_RX_ACTIVE_PIN                         (14U)
#define BSP_COEX_RX_ACTIVE_PORT                        (gpioPortD)
#define BSP_COEX_RX_ACTIVE_ASSERT_LEVEL                (1)
#define BSP_COEX_RX_ACTIVE_CHANNEL                     (5)
#define BSP_COEX_RX_ACTIVE_LOC                         (4)

#define BSP_COEX_PHY_SELECT_PIN                        (14U)
#define BSP_COEX_PHY_SELECT_PORT                       (gpioPortC)
#define BSP_COEX_PHY_SELECT_ASSERT_LEVEL               (1)
#define HAL_COEX_DEFAULT_PHY_SELECT_TIMEOUT            (10U)

#ifdef _SILICON_LABS_32B_SERIES_1
#define BSP_COEX_GNT_PIN                               (9U)
#define BSP_COEX_GNT_PORT                              (gpioPortC)

#define BSP_COEX_PRI_PIN                               (13U)
#define BSP_COEX_PRI_PORT                              (gpioPortD)

#define BSP_COEX_DP_PIN                                (12U)
#define BSP_COEX_DP_PORT                               (gpioPortD)
#define BSP_COEX_DP_LOC                                (11U)

#define BSP_COEX_REQ_PIN                               (10U)
#define BSP_COEX_REQ_PORT                              (gpioPortC)

#define BSP_COEX_DP_REQUEST_INV_CHANNEL                (4)
#define BSP_COEX_DP_CHANNEL                            (3)
#define BSP_COEX_DP_CC0_PIN                            (10U)
#define BSP_COEX_DP_CC0_PORT                           (gpioPortC)
#define BSP_COEX_DP_CC0_LOC                            (15U)
#else //!_SILICON_LABS_32B_SERIES_1
#define BSP_COEX_GNT_PIN                               (3U)
#define BSP_COEX_GNT_PORT                              (gpioPortC)

#define BSP_COEX_PRI_PIN                               (8U)
#define BSP_COEX_PRI_PORT                              (gpioPortD)

#define BSP_COEX_REQ_PIN                               (10U)
#define BSP_COEX_REQ_PORT                              (gpioPortC)

#define BSP_COEX_DP_PIN                                (11U)
#define BSP_COEX_DP_PORT                               (gpioPortD)

#define BSP_COEX_DP_CHANNEL                            (3)
```

9. Enable and edit new #defines 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 coexistence-hal-config.h.

- 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 500µs:

```
#define HAL_COEX_REQ_WINDOW                (500)
```

- REQUEST signal is shared.

  The following #define example disables Shared REQUEST for single-EFR operation.:

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

  The following #define example enables Shared REQUEST.

```
#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)
```

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

  The following #defines example enables active-low GRANT on PC3:

```
#define BSP_COEX_GNT_PIN                   (3)
#define BSP_COEX_GNT_PORT                  (gpioPortC)
#define BSP_COEX_GNT_ASSERT_LEVEL          (1)
```

  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 coexistence-hal-config.h.

- Abort transmission mid packet if GRANT is lost.

  If enabled, losing GRANT (or RHO asserted) during a Bluetooth TX will abort the Bluetooth TX. If not enabled, losing GRANT (or RHO asserted) after the start of a Bluetooth TX will not 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                   (8)
#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 coexistence-hal-config.h.

- PRIORITY Assert Enable

  The following #define example defaults PRIORITY to always deasserted:

```
#define HAL_COEX_PRIORITY_DEFAULT          (0)
```

  The following #define example defaults PRIORITY to asserted or deasserted based on link layer priority and *threshold_coex_pri* as described below:

```
#define HAL_COEX_PRIORITY_DEFAULT          (1)
```

- PRIORITY signal is shared

  The following #define example disables Shared PRIORITY for single-EFR operation.

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

  The following #define example enables Shared PRIORITY.

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

- 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 coexistence-hal-config.h.

- PWM enabled at reset, period, duty-cycle, priority, polarity, port and pin.

  PWM asserts REQUEST and optionally PRIORITY at a regular period and duty-cycle. PWM can be employed to create idle Wi-Fi TX windows to improve 100% Passive SCAN performance and is essential for Bluetooth mesh using ADV-Bearer to allow sufficient idle Wi-Fi TX time windows.

  The following #defines example disables PWM at reset:

```
#define HAL_COEX_PWM_DEFAULT_ENABLED       (0)
```

  The following #defines example enables PWM at reset:

```
#define HAL_COEX_PWM_DEFAULT_ENABLED       (1)
```

  Note: An issue where enabling PWM at reset or later at run-time prevent TX operations, as a result **PWM should not be used in that case**. This issue will be fixed in a future release.

  The following #defines example sets PWM period (ms) and PWM duty-cycle (%) at reset:

```
#define HAL_COEX_PWM_REQ_PERIOD            (39U)
#define HAL_COEX_PWM_REQ_DUTYCYCLE         (20U)
```

  Note: PWM period should not be an integer sub-multiple of Wi-Fi beacon (typically 102.4 ms). This is required to prevent Wi-Fi from losing many beacons and disassociating. Also, the lowest duty-cycle providing sufficient BT performance is recommended as higher PWM duty-cycles reduce RF time available to Wi-Fi with associated reduction in Wi-Fi throughput.

  However, for Bluetooth mesh using ADV-Bearer method, a period of 39 ms and duty-cycle greater than 44% may be required to receive 99% of ADV-bearer messages (exact PWM requirement depends on Bluetooth mesh retry settings). If possible, Bluetooth mesh should use GATT-bearer method from the co-located Bluetooth mesh radio to relay node.

  The following #defines example deasserts PRIORITY during PWM REQUEST asserted:

```
#define HAL_COEX_PWM_PRIORITY              (0)
```

The following #defines example deasserts PRIORITY during PWM REQUEST asserted:

```
#define HAL_COEX_PWM_PRIORITY              (1)
```

If HAL_COEX_PWM_PRIORITY is set to 1, then REQUEST is “Shared REQUEST” between multiple EFR32 radios and is used to arbitrate which EFR32 controls PTA interface to Wi-Fi. Operating PWM on Shared REQUEST is incompatible with arbitration. As such, the PWM_REQUEST pin becomes necessary. Shared REQUEST interconnects all EFR32 radios for arbitration and PWM_REQUEST is connected to all EFR32 radios, but drives the REQUEST signal to Wi-Fi/PTA.

If HAL_COEX_PWM_PRIORITY is set to 0, then REQUEST is not shared and is used to drive all PTA request to Wi-Fi, both from radio states requests and from PWM.

The following #defines example enables active-high PWM_REQUEST on PC6:

```
#define BSP_COEX_PWM_REQ_PIN             (6U)
#define BSP_COEX_PWM_REQ_PORT            (gpioPortC)
#define BSP_COEX_PWM_REQ_ASSERT_LEVEL    (1)
```

Note: If PWM_REQUEST is not needed (no Shared REQUEST), then remove the BSP_COEX_PWM_REQ_PIN, BSP_COEX_PWM_REQ_PPORT, and BSP_ BSP_COEX_PWM_REQ_ASSERT_LEVEL #defines from coexistence-hal-config.h.

- Directional PRIORITY compiled into image, enabled at reset, pulse width, TIMER, and PRS resources:

PRIORITY can be “static” where it is asserted or deasserted for the entire TX/RX/… or RX/TX/… event. Directional Priority can be used to provide priority information and radio state (TX or RX). The EFR32 implementation of Directional PRIORITY is accomplished using static PRIORITY, REQUEST (or PWM_REQUEST if multi-EFR32 using Shared REQUEST), a TIMER, and up to 6 PRS channels. Because on-chip hardware resources are used with this feature, it is very important to understand which are used and ensure no conflicts. Directional PRIORITY is only supported for PTA implementations where REQUEST (PWM_REQUEST) and PRIORITY are active high.

If enabled, Directional PRIORITY drives a programmable pulse-width (1µs to 255µs) to indicate the priority of TX/RX/… or the priority of RX/TX/… event. Following pulse, Directional PRIORITY signal is low for radio in RX state and high for radio in TX state. The Wi-Fi/PTA device can monitor the Directional PRIORITY signals to understand priority of TX/RX/… or RX/TX/… event and the current radio state. In this manner, simultaneous TX/TX and RX/RX can be allowed and conflicting TX/RX and RX/TX events can be prioritized by PTA mechanism.

The following #defines example prevents compiling Directional PRIORITY into application:

```
#define HAL_COEX_DP_ENABLE                 (0)
```

The following #defines example compiles Directional PRIORITY into application and initializes hardware resources as specified by subsequent #defines:

```
#define HAL_COEX_DP_ENABLE                 (1)
```

Note: REQUEST will assert on valid BLE preamble/sync. REQUEST will also stay asserted through any follow-up TX/RX/... required for this RX packet.

The following #defines example sets Directional PRIORITY pulse-width to 20 µs. If set to 0, Directional PRIORITY reverts to Static PRIORITY.

```
#define HAL_COEX_DP_PULSE_WIDTH_US         (20U)
```

The following #defines example selects the TIMER used by to generate Directional PRIORITY pulse. HAL_TIMER_TIMER0 is reserved for SDK operation and is unavailable. HAL_TIMER_TIMER1 is typically available on all EFR32 devices. HAL_TIMER_TIMER0 and HAL_TIMER_WTIMER1 are available on some EFR32 devices. See the datasheet and reference manual on EFR32 design for details.

```
#define HAL_COEX_DP_TIMER                  (HAL_TIMER_TIMER1)
```

The following #defines example selects the base PRS channel, REQUEST invert PRS channel, and RACLNAEN invert channel used to create Directional PRIORITY. The Bluetooth SDK stack reserves PRS channel 7.

```
#define BSP_COEX_DP_CHANNEL                (3)
#define BSP_COEX_DP_REQUEST_INV_CHANNEL    (4)
```

```
#define BSP_COEX_DP_RACLNAEN_INV_CHANNEL    (8)
```

The following #defines example selects the pin and port used to drive Directional PRIORITY and the LOC value for PRS channel to drive that pin. Consult the selected EFR32 datasheet and reference manual for the LOC required for PRS channel and GPIO pin. Not all GPIOs can be driven by any PRS channel. The PRS base channel must be selected as a channel capable of driving the desired GPIO.

```
#define BSP_COEX_DP_PIN                  (11U)
#define BSP_COEX_DP_PORT                 (gpioPortD)
```

#define BSP_COEX_DP_LOC (11U)The following #defines example selects the pin and port used to drive Directional PRIORITY TIMER to start pulse. In Shared REQUEST, this pin and port must match PWM_REQUEST pin and port. In REQUEST not shared, this pin and port must match REQUEST pin and port. Consult the selected EFR32 datasheet and reference manual for the LOC required for the GPIO to drive the selected TIMER's CC0 input (valid for Series 1 only):

```
#define BSP_COEX_DP_CC0_PIN              (6U)
#define BSP_COEX_DP_CC0_PORT             (gpioPortC)
#define BSP_COEX_DP_CC0_LOC              (11U)
```

10. Add code to initialize and configure coexistence:
    - Add include file to app.c:

      ```
      #include "coexistence-ble.h"
      ```

    - Add one of following variable definition to app.c:

      ```
      uint8 myCoexConfig[] = { 255, 255, 39, 20 }; // for duty-cycled SCAN and no BT Mesh ADV-
      Bearer
      ```

      or

      ```
      uint8 myCoexConfig[] = { 175, 175, 39, 20 }; // for 100% Passive SCAN or BT Mesh ADV-Bearer
      ```

      which is based on the following definition:

      ```
      typedef struct{
        uint8_t threshold_coex_pri; /** Priority line is toggled if priority is below this*/
        uint8_t threshold_coex_req; /** Coex request is toggled if priority is below this*/
        uint8_t coex_pwm_period;    /** PWM Period in ms, if 0 pwm is disabled*/
        uint8_t coex_pwm_dutycycle; /** PWM dutycycle percentage, if 0 pwm is disabled, if >= 100
                                        pwm line is always enabled*/
      }sl_bt_ll_coex_config;

      //Default coex configuration
      #define SL_BT_COEX_DEFAULT_CONFIG { 175, 255, HAL_COEX_PWM_REQ_PERIOD,
      HAL_COEX_PWM_REQ_DUTYCYCLE
      ```

    - Add one of following variable definition to app.c:

      ```
      // for duty-cycled SCAN and no BT Mesh ADV-Bearer and default link layer priorities
      uint8 myLinkLayerPriorities[] = { 191, 143, 175, 127, 135, 0, 55, 15, 16, 16, 0, 4, 4 }
      ```

      or

      ```
      // for duty-cycled SCAN and no BT Mesh ADV-Bearer
      uint8 myLinkLayerPriorities[] = { 223, 175, 174, 127, 135, 0, 55, 15, 16, 16, 0, 4, 4 };
      ```

      which is based on following definition:

      ```
      typedef struct {
              uint8_t scan_min;
              uint8_t scan_max;
              uint8_t adv_min;
              uint8_t adv_max;
              uint8_t conn_min;
              uint8_t conn_max;
              uint8_t init_min;
              uint8_t init_max;
      ```

```
        uint8_t rail_mapping_offset;
        uint8_t rail_mapping_range;
        uint8_t afh_scan_interval;
        uint8_t adv_step;
        uint8_t scan_step;
}sl_bt_bluetooth_ll_priorities;
//Default priority configuration
#define SL_BT_BLUETOOTH_PRIORITIES_DEFAULT { 191, 143, 175, 127, 135, 0, 55, 15, 16, 16, 0,
4, 4 }
```

- Enable or disable Passive SCAN.

```
#define SCAN_PASSIVE                        (0)
```

or

```
#define SCAN_PASSIVE                        (1)
```

- Add point to custom link layer table in config variable in sl_bluetooth.c (instead of the default stack definition SL_BT_CONFIG_DEFAULT):

```
static const sl_bt_configuration_t config = {
.config_flags = SL_BT_CONFIG_FLAGS,                                        \
.sleep.flags = SL_BT_SLEEP_FLAGS_DEEP_SLEEP_ENABLE,                        \
.bluetooth.max_connections = SL_BT_CONFIG_MAX_CONNECTIONS,                 \
.bluetooth.max_advertisers = SL_BT_CONFIG_MAX_ADVERTISERS,                 \
.bluetooth.max_periodic_sync = SL_BT_CONFIG_MAX_PERIODIC_ADVERTISING_SYNC, \
.bluetooth.mem_pool = sl_bt_default_mem_pool,                              \
.bluetooth.mem_pool_size = sizeof(sl_bt_default_mem_pool),                 \
.bluetooth.sleep_clock_accuracy = SL_BT_CONFIG_SLEEP_CLOCK_ACCURACY,       \  // use modified
Link Layer Priorities
.bluetooth.linklayer_priorities = myLinkLayerPriorities, // default = NULL


.scheduler_callback = SL_BT_CONFIG_LL_CALLBACK,                            \
.stack_schedule_callback = SL_BT_CONFIG_STACK_CALLBACK,                    \
.gattdb = &bg_gattdb_data,                                                 \
.max_timers = SL_BT_CONFIG_MAX_SOFTWARE_TIMERS,                            \
.rf.tx_gain = SL_BT_CONFIG_RF_PATH_GAIN_TX,                                \
.rf.rx_gain = SL_BT_CONFIG_RF_PATH_GAIN_RX,};
```

- Add the coexistence initialization function call and initialize threshold_coex_req and threshold_code_pri within main() in main.c.

```
…
// Initialize stack
sl_bt_init();

// Initialize coexistence
sl_bt_init_coex_hal();

// Initialize threshold_coex_req and threshold_code_pri
sl_bt_coex_set_parameters(myCoexConfig[0],myCoexConfig[1],myCoexConfig[2],myCoexConfig[3]);
```

4.5.2. 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:

```
sl_bt_coex_set_options(SL_COEX_OPTION_ENABLE,0);
```

At runtime, the following code enables the PTA feature:

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_ENABLE, 1);
```

2. REQUEST Window

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

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_REQUEST_WINDOW_MASK, desired_request_window <<
SL_BT_COEX_OPTION_REQUEST_WINDOW_SHIFT);
```

   Where `desired_request_window` is the REQUEST_WINDOW in µs.

3. Abort transmission mid packet if GRANT is lost.

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

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_TX_ABORT, 0);
```

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

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_TX_ABORT, 1);
```

4. PRIORITY Escalation capability

   At runtime, the following code disables PRIORITY assertion:

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_HIGH_PRIORITY, 0);
```

   At runtime, the following code enables PRIORITY assertion:

```
sl_bt_coex_set_options(SL_BT_COEX_OPTION_HIGH_PRIORITY, 1);
```

5. Channel Map Masking

   If an 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 using this function call:

```
sl_status sl_bt_gap_set_data_channel_classification(size_t channel_map_len, const uint8_t*
channel_map)
```

   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.

6. `threshold_coex_req, threshold_code_pri, pwm_period, and pwm_dutycycle`

   It may be required during application execution to change the two coex thresholds and PWM period/duty-cycle. These settings can be changed at run time using this function call:

```
sl_status sl_bt_coex_set_parameters(uint8_t priority, uint8_t request, uint8_t pwm_period,
uint8_t pwm_dutycycle)
```

7. Link layer Priority table.

   It may be required during application execution to change the link layer priority table. This table can be changed at run time using this functional call:

```
sl_status_t sl_bt_system_linklayer_configure (uint8 key,uint8 data_len, const uint8* data)
```

   where `data` is an array containing:

```
typedef struct {
  uint8_t scan_min;
  uint8_t scan_max;
```

```
    uint8_t adv_min;
    uint8_t adv_max;
    uint8_t conn_min;
    uint8_t conn_max;
    uint8_t init_min;
    uint8_t init_max;
    uint8_t rail_mapping_offset;
    uint8_t rail_mapping_range;
    uint8_t afh_scan_interval;
    uint8_t adv_step;
    uint8_t scan_step;
  } sl_bt_bluetooth_ll_priorities;
```

This full array is 17 bytes in length. However, if `data_len` is less than 17, only first `data_len` entries will be modified. For example, if `data_len`=2, only `scan_min` and `scan_max` are updated.

4.5.3. Run-Time PTA Debug Counters

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

```
status_t sl_bt_system_get_counters(uint8_t reset, uint16_t *tx_packuint8_t key, size_t data_len,
const uint8_t* dataets, uint16_t *rx_packets, uint16_t *crc_errors, uint16_t *failures);
```

where:

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

where, since startup or last reset:

- `result` is success (== 0) or failure (!= 0) of `sl_bt_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

4.6. 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 (for Series 1)

- 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 coexistence-hal-config.h are:

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

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

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

#define BSP_COEX_PRI_PIN                                      (12U)
#define BSP_COEX_PRI_PORT                                     (gpioPortD)
#define BSP_COEX_PRI_ASSERT_LEVEL                             (1)
#define HAL_COEX_PRIORITY_DEFAULT                             (1)
#define HAL_COEX_PRI_SHARED                                   (0)

//#define BSP_COEX_RHO_PIN                                      (11U)
//#define BSP_COEX_RHO_PORT                                     (gpioPortC)
//#define BSP_COEX_RHO_ASSERT_LEVEL                             (1)

#define HAL_COEX_PWM_DEFAULT_ENABLED                          (0)
#define HAL_COEX_PWM_REQ_PERIOD                               (39U)
#define HAL_COEX_PWM_REQ_DUTYCYCLE                            (20U)
#define HAL_COEX_PWM_PRIORITY                                 (0)
//#define BSP_COEX_PWM_REQ_PIN                                  (6U)
//#define BSP_COEX_PWM_REQ_PORT                                 (gpioPortC)
//#define BSP_COEX_PWM_REQ_ASSERT_LEVEL                         (1)

#define HAL_COEX_DP_ENABLED                                   (0)
//#define HAL_COEX_DP_PULSE_WIDTH_US                            (20U)
//#define HAL_COEX_DP_TIMER                                     (HAL_TIMER_TIMER1)
//#define BSP_COEX_DP_CHANNEL                                   (3)
//#define BSP_COEX_DP_REQUEST_INV_CHANNEL                       (4)
//#define BSP_COEX_DP_RACLNAEN_INV_CHANNEL                      (8)
//#define BSP_COEX_DP_PIN                                       (12U)
//#define BSP_COEX_DP_PORT                                      (gpioPortD)
//#define BSP_COEX_DP_LOC                                       (11U)
//#define BSP_COEX_DP_CC0_PIN                                   (6U)
//#define BSP_COEX_DP_CC0_PORT                                  (gpioPortC)
//#define BSP_COEX_DP_CC0_LOC                                   (11U)
// [COEX]$
```

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

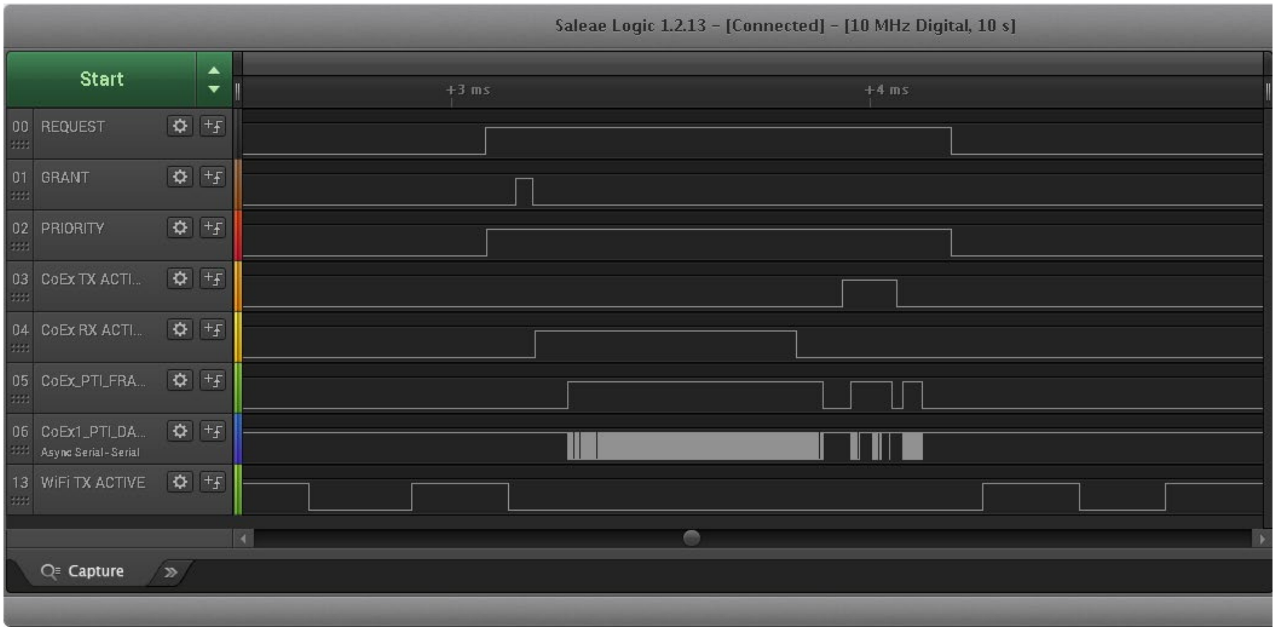


Figure 14. Example CONNECTION event (slave) for Single EFR32 typical 3-Wire Wi-Fi/PTA Logic Analyzer Capture

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 14 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 (for Series 1)

- 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 coexistence-hal-config.h are:

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

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

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

//#define BSP_COEX_PRI_PIN                                   (12U)
//#define BSP_COEX_PRI_PORT                                  (gpioPortD)
//#define BSP_COEX_PRI_ASSERT_LEVEL                          (1)
//#define HAL_COEX_PRIORITY_DEFAULT                          (1)
//#define HAL_COEX_PRI_SHARED                                (0)

//#define BSP_COEX_RHO_PIN                                   (11U)
//#define BSP_COEX_RHO_PORT                                  (gpioPortC)
//#define BSP_COEX_RHO_ASSERT_LEVEL                          (1)

#define HAL_COEX_PWM_DEFAULT_ENABLED                         (0)
#define HAL_COEX_PWM_REQ_PERIOD                              (78U)
#define HAL_COEX_PWM_REQ_DUTYCYCLE                           (20U)
#define HAL_COEX_PWM_PRIORITY                                (0)
//#define BSP_COEX_PWM_REQ_PIN                               (6U)
//#define BSP_COEX_PWM_REQ_PORT                              (gpioPortC)
//#define BSP_COEX_PWM_REQ_ASSERT_LEVEL                      (1)

#define HAL_COEX_DP_ENABLED                                  (0)
//#define HAL_COEX_DP_PULSE_WIDTH_US                         (20U)
//#define HAL_COEX_DP_TIMER                                  (HAL_TIMER_TIMER1)
//#define BSP_COEX_DP_CHANNEL                                (3)
//#define BSP_COEX_DP_REQUEST_INV_CHANNEL                    (4)
//#define BSP_COEX_DP_RACLNAEN_INV_CHANNEL                   (8)
//#define BSP_COEX_DP_PIN                                    (12U)
//#define BSP_COEX_DP_PORT                                   (gpioPortD)
//#define BSP_COEX_DP_LOC                                    (11U)
//#define BSP_COEX_DP_CC0_PIN                                (6U)
//#define BSP_COEX_DP_CC0_PORT                               (gpioPortC)
//#define BSP_COEX_DP_CC0_LOC                                (11U)
// [COEX]$
```

The logic analyzer capture in Figure 15 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:

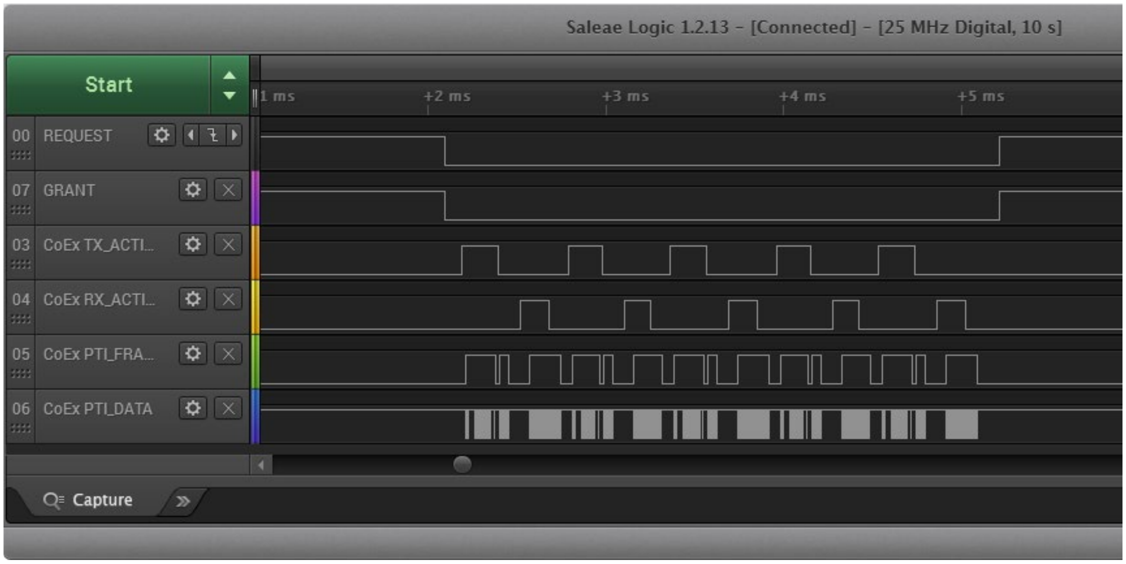


Figure 15. Example CONNECTION event (master) for Multi-EFR32 2-Wire Wi-Fi/PTA Logic Analyzer Capture (first anchor point in CONNECTION, using active-low REQUEST)

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 15 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

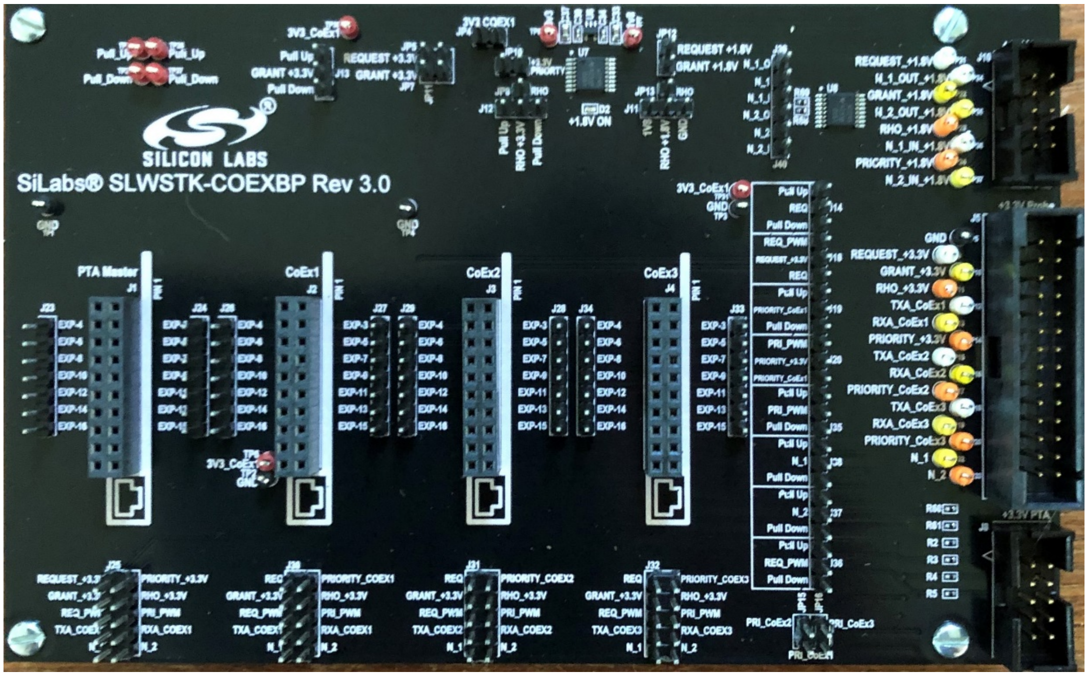
5.1. Example TX_ACTIVE/RX_ACTIVE

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 EFR32MG 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. 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 coexistence-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 4.5.2 Run-Time PTA Re-configuration and section 4.5.3 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.

8. Document Revision History

Revision 1.5

June 2020

Bluetooth version: 2.13.5.0 Bluetooth Mesh version: 1.6.3.0

- Renamed section 4.1 to PTA Support Options; added heading level three to 1-Wire PTA, 2-Wire PTA, 3-Wire PTA, and 4-Wire PTA.
- Added section 4.2 Wi-Fi/PTA Considerations, section 4.3 PWM for High Duty Cycle Wi-Fi, and section 4.4 Directional PRIORITY from *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi*.
- Updated section 4.6 Coexistence Configuration Setup Examples for Different Wi-Fi/PTA Applications due to changes in the Bluetooth SDK 3.0.0.
- Updated figures in section 4.4.1 Single-EFR32 PTA with Directional PRIORITY and section 4.4.2 Multi-EFR32 PTA with Directional PRIORITY. They use the RACPAEN signal. RACLNAEN is no longer in use.
- Corrected the Static PRIORITY signal assignment in section 4.4.1 Single-EFR32 PTA with Directional PRIORITY and section 4.4.2 Multi-EFR32 PTA with Directional PRIORITY.

Revision 1.4

March 2020

Bluetooth version: 2.13.3.0 Bluetooth Mesh version: 1.6.2.0

- Made minor text changes.

Revision 1.3

February 2020

Bluetooth version: 2.13.2.0 Bluetooth Mesh version: 1.6.1.0

- Deleted all text dealing with the implementation of managed coexistence and moved it to *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi* available under non-disclosure from Silicon Labs technical support. In prior revisions, this content resided in *AN1128-NDA: Bluetooth® Coexistence with Wi-Fi* which has been deprecated.

Revision 1.2

January 2020

Bluetooth version: 2.13.1.0 Bluetooth Mesh version: 1.6.1.0

- Updated PTA REQUEST to PRIORITY timing.
- Updated Directional PRIORITY PRS/TIMER implementation and added timing diagrams.
- Added Directional PRIORITY run-time configuration BGAPI command.
- Removed PWM and Directional PRIORITY errata.

Revision 1.1

December 2019

Bluetooth version: 2.13.0.0 Bluetooth Mesh version: 1.6.1.0

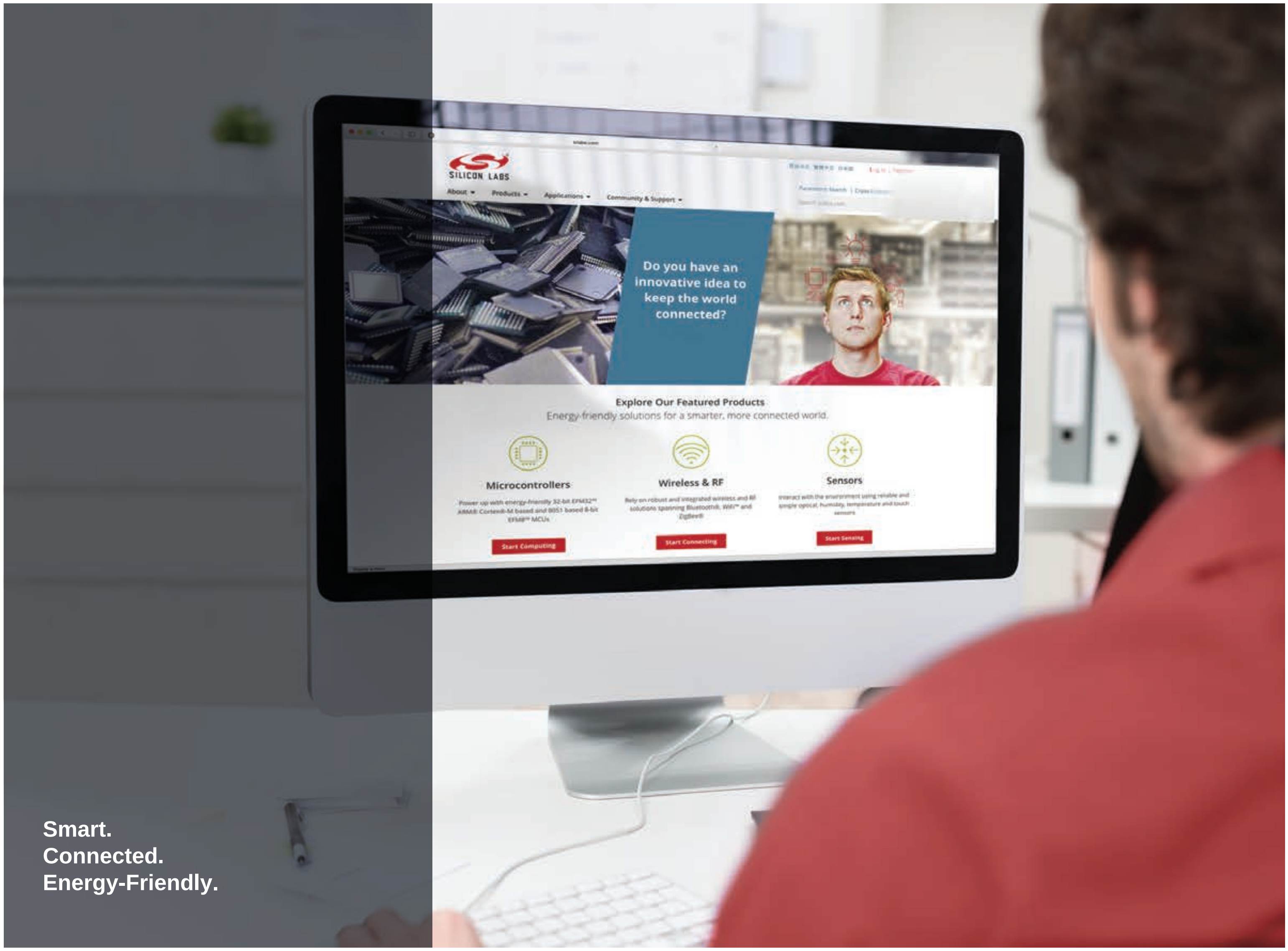
- Added SL Thread notice on first page.
- Added Link Layer PRIORITY support.
- Added 100% Passive SCAN and Bluetooth mesh ADV-Bearer support.
- Added PWM information (not functional in Bluetooth 2.13.0.0).
- Added Directional PRIORITY support.

Revision 1.0

December 2018

Bluetooth version: 2.11.0.0 Bluetooth Mesh version: 2.8.0.0

- Initial release



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information

SILICON LABS

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

http://www.silabs.com