AN1017: Zigbee® and Thread Coexistence with Wi-Fi

This application note describes the impact of Wi-Fi on Zigbee and Thread and methods to improve coexistence with Wi-Fi. First, design considerations to improve coexistence without direct interaction between Zigbee/Thread and Wi-Fi radios are described. These techniques are applicable both to the EM35x/EM358x and Mighty Gecko family (EFR32MGx). Next, Silicon Lab's Packet Traffic Arbitration (PTA) support to coordinate 2.4GHz RF traffic for co-located Zigbee/Thread and Wi-Fi radios is described. This PTA feature set is available for the EFR32MGx family.

Additional details about the implementation of managed coexistence and test results are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

KEY POINTS

- Wi-Fi impact on Zigbee/Thread
- Improving unmanaged coexistence
- Implementing managed coexistence
- Extending the coexistence solution

Contents

1 Introduction

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

- Increased Wi-Fi transmit power level for “extended range”
  - +30dBm Wi-Fi Access Points are now common.
- Increased Wi-Fi throughput
  - Depending on achievable SNR, high throughput requirements for file transfers and/or video streaming may result in high Wi-Fi duty cycle within 2.4GHz ISM band.
- Integrating Wi-Fi, Zigbee, Thread, and Bluetooth low energy into the same device for gateway functionality
  - This is required by Home Automation and Security applications, and provides easier end-node commissioning using Bluetooth low energy.

This application note describes the impact of Wi-Fi on Zigbee and Thread and methods to improve coexistence with Wi-Fi on two Silicon Labs integrated circuits—the EM35x/EM358x and the Mighty Gecko family (EFR32MGx).

- Section 3 Unmanaged Coexistence describes design considerations to improve coexistence without direct interaction between Zigbee/Thread and Wi-Fi radios.
- Section 4 Managed Coexistence describes Silicon Lab’s PTA (Packet Traffic Arbitration) support to coordinate 2.4GHz RF traffic for co-located Zigbee/Thread and Wi-Fi radios. PTA support is available for the EFR32MGx only.
- Section 5 Application Code Coexistence Extensions describes application code available to extend the coexistence solution for particular applications.

Notes:

1. Current Bluetooth and Bluetooth low energy solutions operate at less than +10dBm transmit power level and implement automatic frequency hopping (AFH). Even with Bluetooth at +10dBm, Silicon Labs’ testing indicates minimal impact from co-located Bluetooth on 802.15.4 performance. 802.15.4 device join success remains at 100% and 802.15.4 message loss remains at 0%. 802.15.4 MAC retries increase slightly due to occasional co-channel and adjacent channel collisions between the Bluetooth AFH and fixed 802.15.4 channel. However, the Wi-Fi coexistence discussion and solutions described in this application note can be applied equally well to Bluetooth coexistence. As such, all following solutions presented for “Wi-Fi Coexistence” can be applied to “Wi-Fi/Bluetooth Coexistence”.
2. The revision 1.5 application note describes EFR32 Zigbee and Thread coexistence support for EmberZNet PRO 6.5.0 and Silicon Labs Thread 2.9.0. Not all coexistence support features in EmberZNet PRO 6.5.0 or Silicon Labs Thread 2.9.0 are available in earlier EmberZNet PRO and Thread stack releases supporting coexistence (that is, EmberZNet PRO 5.8.0–6.4.0 and Thread 2.1.0–2.8.0). Section 4.2 PTA Support Software Setup notes the EmberZNet PRO and Silicon Labs Thread stack releases where each coexistence support feature became available.

2 Wi-Fi Impact on Zigbee/Thread

World-wide, Wi-Fi (IEEE 802.11b/g/n) supports up to 14 overlapping 20/22MHz bandwidth channels across the 2.4GHz ISM band with transmit power levels up to +30dBm. Similarly, 2.4GHz Zigbee and Thread (based on IEEE 802.15.4) support 16 non-overlapping 2MHz bandwidth channels at 5MHz spacing with transmit powers up to +20dBm. These Wi-Fi and Zigbee/Thread channel mappings are shown in the following figure.

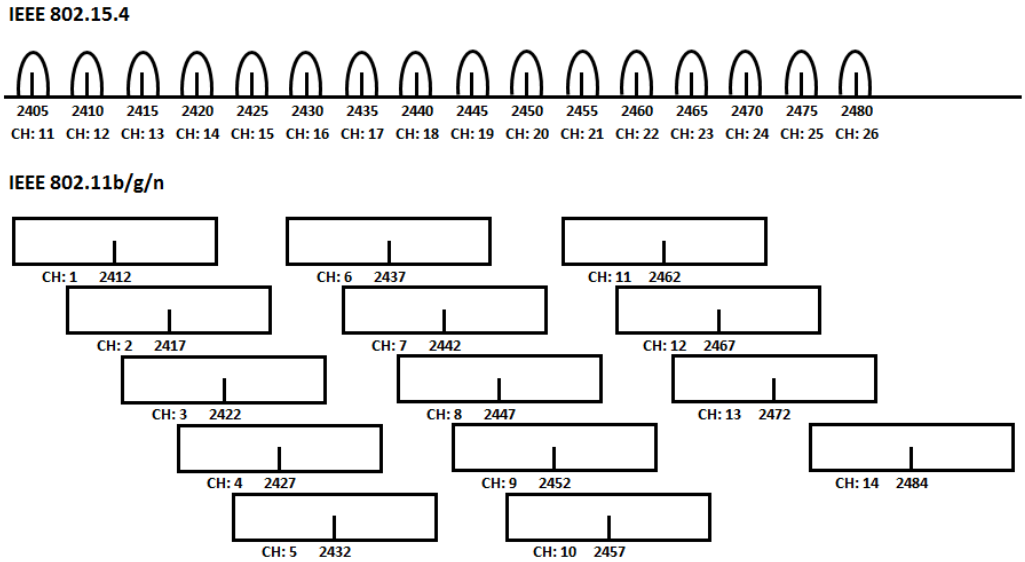


Figure 1. 802.15.4 and 802.11b/g/n Channel Mapping (World-Wide)

Actual channels available vary by country. For example, in the USA, Wi-Fi channels 1 through 11 are available and Zigbee channels 11 through 26 are available, although channels 25 and 26 require reduced transmit power levels to meet FCC requirements (North America only).

To better understand the effects of Wi-Fi on Zigbee/Thread, Silicon Labs measured the impact of a 100% duty-cycled 802.11n (MCS3, 20MHz bandwidth) blocker transmitting at various power levels while receiving an 802.15.4 message transmitted at power level sufficient to achieve 1% PER (receive sensitivity). The results for co-channel, adjacent channel, and “far-away” channel are shown in the following figure. All 802.11n and 802.15.4 power levels are referenced to the Silicon Labs’ Mighty Gecko (EFR32MG1P232F256GM) RF input. The test application was developed using Silicon Labs’ EmberZNet PRO (Zigbee) stack with NodeTest running on the EFR32 DUT (Device Under Test) and a test script to control the DUT and RF test equipment. Because this is an 802.15.4 focused test, results are identical for Wi-Fi blocking Thread.

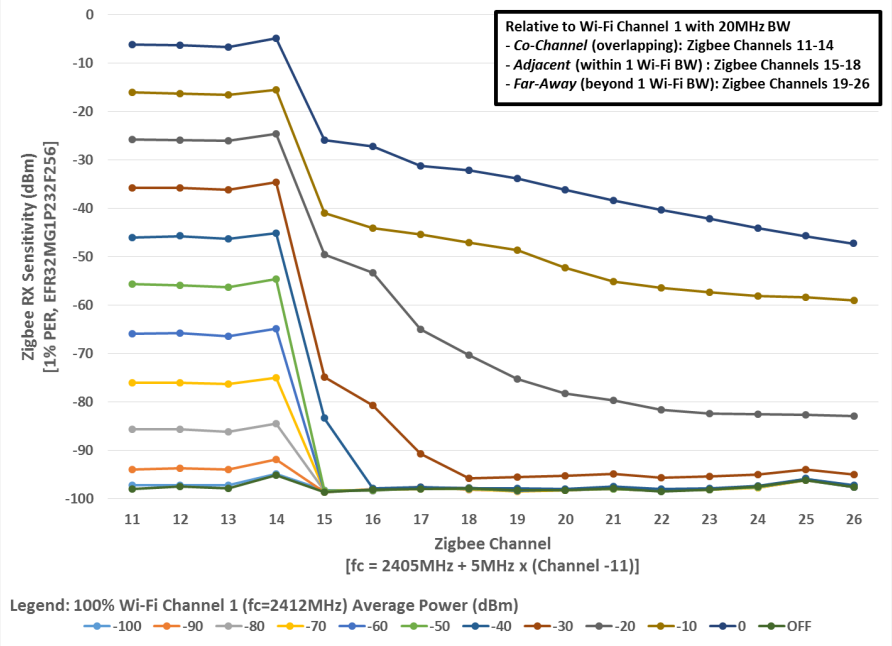


Figure 2. 802.15.4 Receive Sensitivity with 100% Duty Cycled 802.11n (MCS3/20MHz) Wi-Fi Blocker

From the figure above, as well as other measurements using the EM35x/EM358x (not shown), these are the key observations about the impact of Wi-Fi on Zigbee/Thread.

Co-Channel (Zigbee overlapping Wi-Fi):

- At Zigbee channel 12, EFR32MG1P232F256GM can receive an 802.15.4 signal down to 6dB weaker than aggregate Wi-Fi transmit power (100% duty cycle).
  - This receive sensitivity limitation impacts both co-located and remote, not co-located, 802.15.4 radios.
- At Zigbee channel 12, EM35x/EM358x with and without FEM (front-end module) can receive an 802.15.4 signal down to 6dB weaker than aggregate Wi-Fi transmit power (100% duty cycle).
- 802.15.4 transmits can also be blocked by Wi-Fi transmit power tripping the 802.15.4 -75dBm CCA (Clear Channel Assessment) threshold.

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

- At Zigbee channel 15, EFR32MG1P232F256 can receive a -85dBm 802.15.4 signal with -42dBm or weaker Wi-Fi transmit power (100% duty cycle).
  - Maximum receive sensitivity is attained at -45dBm or weaker Wi-Fi transmit power (100% duty cycle).
- At Zigbee channel 15, EM35x/EM358x without FEM can receive a -85dBm 802.15.4 signal with -42dBm or weaker Wi-Fi transmit power (100% duty cycle), -46dBm or weaker with Skyworks SE2432L FEM LNA (Low Noise Amplifier) enabled.

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

- At Zigbee channel 25, EFR32MG1P232F256 can receive a -85dBm 802.15.4 signal with -22dBm or weaker Wi-Fi transmit power (100% duty cycle).
  - Maximum receive sensitivity is attained at -31dBm or weaker Wi-Fi transmit power (100% duty cycle).
- At Zigbee channel 25, EM35x/EM358x without FEM can receive a -85dBm 802.15.4 signal with -25dBm or weaker 100% Wi-Fi transmit power (100% duty cycle), -29dBm or weaker with Skyworks SE2432L FEM LNA enabled.

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 (Signal to Noise Ratio) conditions. As seen in the figure above, the EFR32MGx (or EM35x/EM358x) 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 802.15.4 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.

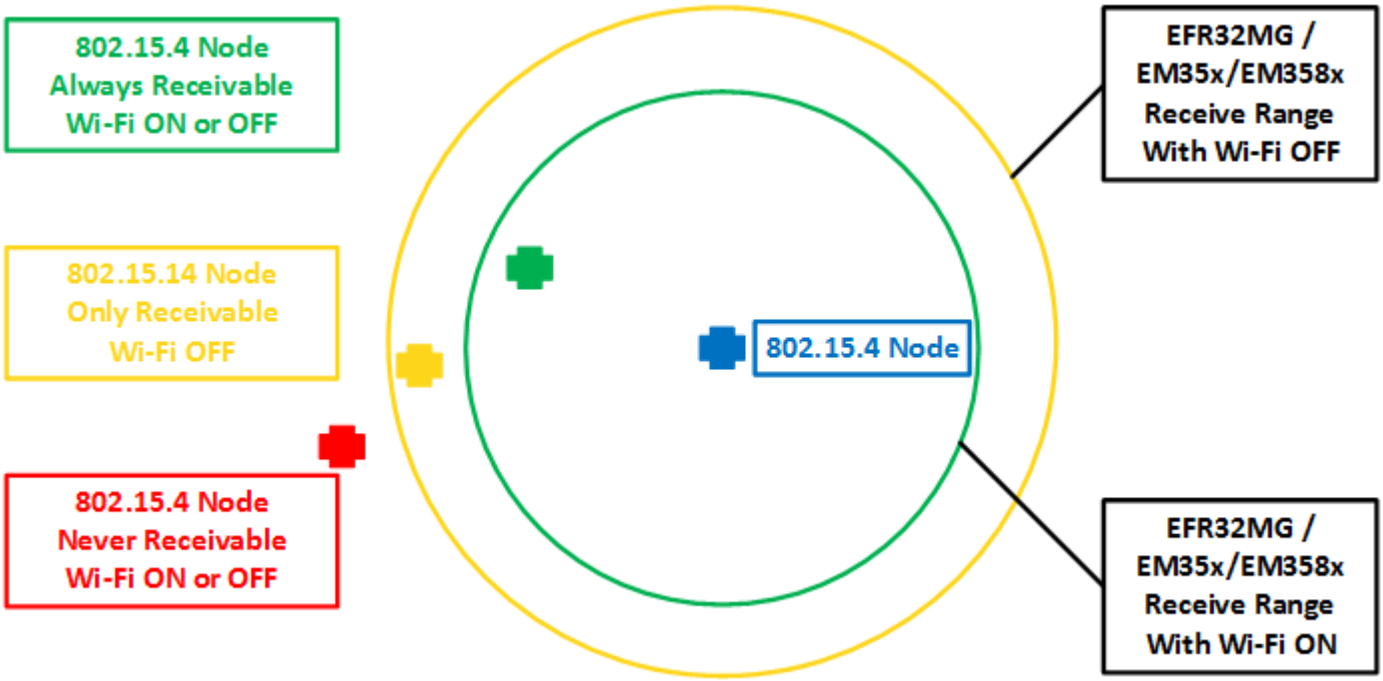


Figure 3. EFR32MGx or EM35x/EM358x Receiver Desensitized when Wi-Fi Transmitting

Depending on each node's type (coordinator, router, or end device) and the Wi-Fi duty cycle, the impact of strong Wi-Fi turning ON/OFF will vary.

In a Zigbee network:

- Coordinator: Tasked with network creation, the control of network parameters, and basic maintenance, in addition to performing an application function, such as aggregating data or serving as a central control point or gateway.
- Router: In addition to running an application function, a Router can receive and retransmit data from other nodes.
- End Device Typically a battery-powered device (Sleepy End Device) running an application function and able to talk to a single parent node (either the Coordinator or a Router). End Devices cannot relay data from other nodes.

In a Thread network:

- Border Router: Provides Thread Network node connectivity to other devices in external networks (for example, Internet access).
- Router: In addition to running an application function, a Router can receive and retransmit data from other nodes and provide join and security capability. When routing function is not needed by network, a Router can downgrade to a Router-Eligible End Device.
- Router-Eligible End Device: In addition to running an application function, a Router-Eligible End Device can receive and retransmit data from other nodes. When additional routers needed by network, a Router-Eligible End Device can upgrade to a Router.
- Sleepy End Device: Typically a battery-powered device running an application function and able to talk to a single parent node (either a Border Router, Router, or Router-Eligible End Device). Sleepy End Devices cannot relay data from other nodes.

Two Zigbee cases are considered below, but many other cases are possible.

Case 1: Zigbee Coordinator near strong Wi-Fi plus three end-nodes

For this case, the figure above is composed of:

- Coordinator: Blue node
- End Devices: Green, Yellow, and Red nodes

In this simple network, each end device attempts to join the network formed by the coordinator. However, the red node is outside of receive range and cannot join. With Wi-Fi OFF, both the green and yellow nodes successfully join the network and have no issues sending messages to the Coordinator. Regardless of Wi-Fi ON/OFF duty cycle, the green node remains successful sending messages to the Coordinator.

With Wi-Fi ON/OFF at low-duty cycle, some messages from the yellow node are periodically blocked, but Zigbee retry mechanisms are effective in getting the messages to the coordinator. However, with Wi-Fi ON/OFF at high-duty cycle, many messages from the yellow node are blocked and Zigbee retry mechanisms may be exhausted. Even when retry mechanisms are successful, the message latency increases. If the yellow node is a battery-powered Sleepy End Device, it must remain active longer to execute retries, reducing battery life.

Case 2: Zigbee Coordinator near strong Wi-Fi, Router within always receive range, plus two end-nodes

For this case, the figure above is composed of:

- Coordinator: Blue node
- Router: Green node
- End Devices: Yellow and Red nodes

In this simple network, the green Router forms a route directly to the Coordinator, maintained regardless of Wi-Fi ON/OFF duty cycle. With Wi-Fi OFF, the yellow node forms a route directly to the blue Coordinator at a lower route cost than a route via the green Router. The red node cannot be received by the Coordinator and its messages are also routed through the Router to the Coordinator.

With Wi-Fi OFF, the green Router, the yellow node, and the red node (via the green Router) have no issues sending messages to the Coordinator. Regardless of Wi-Fi ON/OFF duty cycle, the green Router and the red node (via the green Router) remain successful sending messages to the Coordinator. With Wi-Fi ON/OFF at low-duty cycle, some messages from the yellow node are periodically blocked, but Zigbee retry mechanisms are effective in getting the messages to the Coordinator.

With Wi-Fi ON/OFF at high-duty cycle, many messages from the yellow node are blocked and Zigbee retry mechanisms may be exhausted. If Wi-Fi ON/OFF stays at high-duty cycle for sufficient time, the network responds by restructuring the yellow node to route messages to the Coordinator via the Router. However, this route rediscover takes time and messages may be lost. As long as Wi-Fi ON/OFF remains high-duty cycle, the yellow node messages will continue to go through the Router, which forwards messages to the Coordinator.

However, when Wi-Fi ON/OFF returns to low-duty cycle, the network will, due to lower route cost, return to the original structure with the yellow node sending messages directly to the Coordinator.

Under conditions with Wi-Fi ON/OFF switching between low and high duty cycles, the network may switch back and forth between these two route states. During these switching events, messages from the yellow end-node to the Coordinator are lost.

3 Unmanaged Coexistence

The unmanaged coexistence recommendations that follow provide guidance on how to maximize the EFR32MGx or EM35x/EM358x message success with strong nearby Wi-Fi.

3.1 Implement Frequency Separation

From the observations in the previous section, co-channel operation of 802.15.4 with 100% duty cycle Wi-Fi blocks most of the 802.15.4 messages and must be avoided. Also, EFR32MGx tolerates up to 20dB stronger Wi-Fi signal in “far-away” channel case than in adjacent channel case. The 802.15.4 network performance is improved by maximizing the frequency separation between the Wi-Fi network and the 802.15.4 network.

If the Wi-Fi and 802.15.4 radios are implemented with a common host (MCU controlling both radios), then the host should attempt to maximize the frequency separation. For Wi-Fi networks, the Access Point (AP) establishes the initial channel and, in auto channel configuration, is free to move the network to another channel using the Channel Switch Announcement (CSA), introduced in 802.11h, to schedule the channel change.

For Thread networks, frequency separation implementation depends on the application layer. For Zigbee networks, the Coordinator establishes the initial channel. However, when implemented by the product designer, a Network Manager function, which can be on the Coordinator or on a Router, can solicit energy scans from the mesh network nodes and initiate a network channel change as necessary to a quieter channel.

Note: The Network Manager function is not a mandatory feature, but rather must be implemented using tools/functions provided by the EmberZNet PRO stack.

Details on mesh network channel frequency agility can be found in:

- https://www.silabs.com/community/wireless/zigbee-and-thread/knowledge-base.entry.html/2012/06/29/how_does_frequencya-x5lU
- https://www.silabs.com/community/wireless/zigbee-and-thread/knowledge-base.entry.html/2012/06/29/how_can_i_test_frequ-vrgk

3.2 Operate Wi-Fi with 20MHz Bandwidth

Since Wi-Fi 802.11n uses OFDM (Orthogonal Frequency-Division Multiplexing) sub-carriers, third-order distortion products from these sub-carriers extend one bandwidth on each side of the Wi-Fi channel. 802.11n can operate in 20MHz or 40MHz modes. If operated in 40MHz mode, 40MHz of the 80MHz ISM band is consumed by the Wi-Fi channel. However, an additional 40MHz on each side can be affected by third-order distortion products. These third-order products can block the 802.15.4 receiver and is the primary reason adjacent channel performance is up to 20dB worse than “far-away” channel performance.

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

If the Wi-Fi and 802.15.4 radios are implemented with a common host, then the host should have the Wi-Fi radio set the **FortyMHz Intolerant** bit during association to force the Wi-Fi to 20MHz mode, improving the 802.15.4 performance.

If the application requires Wi-Fi to operate in 40MHz mode, frequency separation must be maximized by placing Wi-Fi channels and 802.15.4 channel at opposite ends of the 2.4GHz ISM band.

From Silicon Labs’ managed coexistence testing, 802.15.4 performance with 40MHz Wi-Fi, *for the same Wi-Fi RF duty cycle*, is comparable to 802.15.4 performance with 20MHz Wi-Fi. While 802.15.4 performance with 100% Wi-Fi RF duty cycle is inherently impaired, 40MHz Wi-Fi, *for the same target Wi-Fi data rate*, has a lower RF duty cycle than 20MHz Wi-Fi, providing the 802.15.4 radio more frequent and longer time gaps for successful transmits and receives.

3.3 Increase Antenna Isolation

From the observations in Section 2 Wi-Fi Impact on Zigbee/Thread, minimizing the Wi-Fi energy seen by the 802.15.4 RF input improves the 802.15.4 receive range. For example, in the “far-away” channel case (Wi-Fi channel 1 and Zigbee channel 25) with 100% Wi-Fi duty cycle, a -85dBm 802.15.4 message can be received when the average Wi-Fi energy at EFR32MGx input is -22dBm or less. If the Wi-Fi average transmit power level is +10dBm, 32dB or more antenna isolation between the Wi-Fi transmitter and 802.15.4 RF input is required to always receive a -85dBm 802.15.4 signal, Wi-Fi ON or OFF.

Increased antenna isolation can be achieved by:

- Increasing the distance between antennas. In open-space, far-field, power received is proportional to $1/R^2$, where R is the distance between antennas.
- Taking advantage of antenna directionality. A monopole antenna provides a null along the axis of the antenna, which can be directed toward the Wi-Fi antenna(s).

3.4 Use Zigbee/Thread Retry Mechanisms

The 802.15.4 specification requires retries at the MAC (Media Access Control) layer, which are implemented in Silicon Labs' EmberZNet PRO stack. To further improve message delivery robustness, Silicon Labs EmberZNet PRO stacks also implements NWK (Zigbee network layer) retries, wrapping the MAC retries. The user application can also take advantage of APS (Zigbee Application Support (APS) Sub-Layer) retries, wrapping the NWK retries. More information on the retry mechanisms can be found at:

- https://www.silabs.com/documents/public/user-guides/ug103-02-appdevfundamentals-zigbee.pdf (Section 4.4)
- https://www.silabs.com/community/wireless/zigbee-and-thread/knowledge-base.entry.html/2012/06/29/how_does_the_emberzn-po1M

These retry mechanisms are effective at improving message delivery. However, under high interference conditions, message latency increases.

For Thread networks, 802.15.4 retries at the MAC layer still apply. However, other message retry mechanisms depend on the application layer.

3.5 Remove FEM (or Operate FEM LNA in Bypass)

EFR32MGx can deliver nearly +20dBm transmit power and has excellent receiver sensitivity without an external FEM (Front End Module). However, many EM35x/EM358x applications utilize an external FEM to increase transmit power to +20dBm for increased range (in regions where this is permitted, for example, the Americas). The additional FEM LNA receive gain also improves sensitivity. However, this additional gain also degrades the EM35x/EM358x linearity performance in the presence of strong Wi-Fi.

As an example, a Skyworks' SE2432L FEM combined with an EM35x/EM358x provides increased transmit power and, when no blockers are present, increased receiver sensitivity. However, in the presence of strong 100% duty cycle Wi-Fi, EM57x/EM358x with SE2432L shows 4dB degradation in receiver sensitivity when compared to EM57x/EM358x only, for both adjacent and "far-away" channel cases.

For best receive sensitivity in the presence of strong Wi-Fi blockers, either eliminate the FEM or operate the FEM LNA in bypass mode. This recommendation is a trade-off as receive sensitivity without Wi-Fi blockers is improved with FEM LNA gain enabled.

4 Managed Coexistence

The market trends of higher Wi-Fi transmit power, higher Wi-Fi throughput, and integration of Wi-Fi and 802.15.4 radios into the same device have the following impacts.

- Advantages:
  - Host can implement frequency separation between Wi-Fi and 802.15.4.
  - Co-located Wi-Fi radio can force the Wi-Fi network to operate with 20MHz bandwidth.
  - Co-located Wi-Fi and 802.15.4 radios can communicate pending and/or in-progress activity on 2.4GHz ISM transmits and receives.
- Disadvantages:
  - Higher Wi-Fi transmit power requires greater antenna isolation.
  - Higher Wi-Fi throughput results in higher Wi-Fi duty cycle.
  - Antenna isolation is usually limited by the size of the product (only 15-20dB isolation is not unusual).

Assuming frequency separation achieves the “far-away” channel case and Wi-Fi only uses 20MHz bandwidth, a +30dBm Wi-Fi transmit power level at 100% duty cycle requires 45dB antenna isolation to receive -80dBm 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 802.15.4 radios to coordinate each radio’s access to the 2.4GHz ISM band for transmit and receive. For the EFR32, Silicon Labs has implemented a coordination scheme compatible with Wi-Fi devices supporting PTA. This PTA-based coordination allows the EFR32 to signal the Wi-Fi device when receiving a message or wanting to transmit a message. When the Wi-Fi device is made aware of the EFR32 requiring the 2.4GHz ISM band, any Wi-Fi transmit can be delayed, improving Zigbee/Thread message reliability.

Section 4.1 discusses PTA support hardware options, Section 4.2 discusses PTA support software setup, and Section 4.3 provides test results for EFR32 PTA implementation under various Wi-Fi operating conditions.

Notes:

1. PTA support software is only available for EFR32MGx under EmberZNet PRO 5.8.0 or later (Zigbee) and in Silicon Labs Thread 2.1.0 or later (Thread). Section 4.2 PTA Support Software Setup notes the EmberZNet PRO and Silicon Labs Thread stack releases where each coexistence support feature became available.
2. While PTA support software is not available for EM3x, EM3x does support the TXA (TX_ACTIVE) and legacy RHO (Radio Hold OFF) features. The following links describe existing EM3x coexistence support features:
   - http://community.silabs.com/t5/Mesh/WiFi-and-Zigbee-coordination/td-p/146392
   - http://community.silabs.com/t5/Mesh-Knowledge-Base/EM3xx-Designing-for-radio-network-coexistence/ta-p/158864

4.1 PTA Support Hardware Options

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

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

In considering radio state, transmit/receive, and frequencies, 802.15.2 describes the following.

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

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

From the above table, 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 802.15.4.
- 802.15.2 “Out-of-Band” covers both Adjacent and “Far-Away” Channel operation, which showed ~20dB improvement in “Far-Away” Channel vs. Adjacent Channel.

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

- Frequency Separation
- Operate Wi-Fi in 20MHz Bandwidth
- Antenna Isolation
- Zigbee/Thread 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.

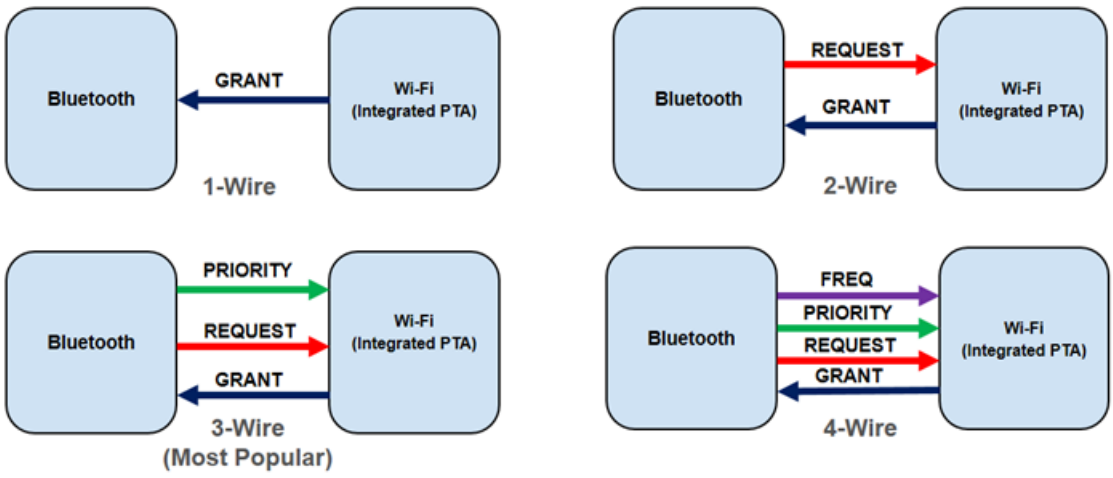


Figure 4. Typical Wi-Fi/Bluetooth PTA Implementations

1-Wire PTA

In 1-Wire PTA, 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 external radio to request the 2.4GHz ISM and is not recommended.

2-Wire PTA

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

3-Wire PTA

In 3-Wire PTA, 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.

Note: For **EmberZNet PRO 6.5.0** or later and **Silicon Labs Thread 2.9.0** or later, PRIORITY can be implemented as either static or directional (enhanced) priority. Silicon Labs' EFR32 supports only static priority for **EmberZNet Pro 6.4.0** or **Silicon Labs Thread 2.8.0** or earlier.

- 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 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 run-time, to address various product optimization requirements. However, given the relatively low RF duty cycle of 802.15.4, static PRIORITY may not be necessary for platforms that do not experience high Wi-Fi duty cycles nor support Bluetooth audio streaming, freeing a GPIO pin on the EFR32 and SoC.

4-Wire PTA

In 4-Wire PTA, 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. For this reason, Silicon Labs' EFR32 does not support the FREQ signal. Silicon Labs therefore recommends asserting the FREQ input to the Wi-Fi/PTA.

4.1.1 Single EFR32 Connected to Wi-Fi/PTA

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.1.2 Multiple EFR32s Connected to Wi-Fi/PTA

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.1.3 Wi-Fi/PTA Considerations

Additional details about the implementation of managed coexistence are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

4.2 PTA Support Software Setup

EmberZNet PRO 5.8.0 and later, and Silicon Labs Thread 2.1.0 and later contain EFR32 PTA support, enabling customers to implement EFR32 PTA configured for target Wi-Fi/PTA platform. **However, not all PTA support features are available in all revisions.**

Note: While coexistence PTA support is also available in EmberZNet PRO 6.0.0 and Silicon Labs Thread 2.4.0, **use in those stacks is not recommended**. In those stacks, the PTA signals can be incorrect, resulting in EFR32 not gaining protected TX or RX and/or incorrectly holding 2.4GHz band from Wi-Fi.

Refer to the following table for PTA support features available in a particular release.

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

PTA Feature	Configure		EmberZNet PRO							SL Thread						
	AppBuilder	Run-Time API [3]	5.8.0, 5.8.1	5.9.0	5.9.1	5.9.2, 5.10.0, 5.10.1	6.1.0, 6.2.0	6.3.0, 6.3.1, 6.4.0	6.5.0, 6.5.1	2.1.0, 2.1.1	2.2.0	2.2.1	2.3.0, 2.3.1	2.5.0, 2.6.0	2.7.0, 2.7.1, 2.8.0	2.9.0, 2.9.1
RHO pin settings: enable/disable, polarity, port and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REQUEST pin settings: Enable/disable, polarity, port and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REQUEST signal is shared	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REQUEST signal max backoff mask	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
REQUEST PWM enable/disable, period, and duty-cycle	✓	✓	[5]	[5]	[5]	[5]	[5]	✓	✓	[5]	[5]	[5]	[5]	[5]	✓	✓
GRANT pin settings: Enable/disable, polarity, port and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Abort transmission mid packet if GRANT is lost	✓	✓				✓	✓	✓	✓				✓	✓	✓	✓
PRIORITY pin settings: Enable/disable, polarity, port and pin	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PRIORITY signal is shared	✓		[4]	[4]	[4]	[4]	✓	✓	✓	[4]	[4]	[4]	[4]	✓	✓	✓
PRIORITY assertion based on TX/RX traffic	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PRIORITY PWM high or low	✓	✓	[5]	[5]	[5]	[5]	[5]	✓	✓	[5]	[5]	[5]	[5]	[5]	✓	✓
Receive retry REQUEST enabled **[1]**	✓	✓				✓	✓	✓	✓				✓	✓	✓	✓
Receive retry timeout mask	✓	✓				✓	✓	✓	✓				✓	✓	✓	✓
REQUEST high PRIORITY on receive retry	✓	✓				✓	✓	✓	✓				✓	✓	✓	✓
Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Disable REQUEST (force holdoff)		✓		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Synch MAC to GRANT (MAC holdoff)		✓			✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)		✓		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
• TX PRIORITY Escalation capability **[2]**	✓ [2]					[2]	[2]	✓	✓				[2]	[2]	✓	✓
• CCA/GRANT TX PRIORITY Escalation Threshold		✓				✓	✓	✓	✓				✓	✓	✓	✓
• MAC Fail TX PRIORITY Escalation Threshold		✓				✓	✓	✓	✓				✓	✓	✓	✓
• Pulse Directional PRIORITY settings: • Enable/disable, Directional PRIORITY timer, RX priority pulse hold time, Directional PRIORITY PRS	✓								✓							✓

PTA Feature	Configure		EmberZNet PRO							SL Thread						
	AppBuilder	Run-Time API [3]	5.8.0, 5.8.1	5.9.0	5.9.1	5.9.2, 5.10.0, 5.10.1	6.1.0, 6.2.0	6.3.0, 6.3.1, 6.4.0	6.5.0, 6.5.1	2.1.0, 2.1.1	2.2.0	2.2.1	2.3.0, 2.3.1	2.5.0, 2.6.0	2.7.0, 2.7.1, 2.8.0	2.9.0, 2.9.1
channel, inverted REQUEST PRS channel, inverted RACLNAEN PRS channel.																
• Passive Configuration	[6]								[6]							[6]

- **[1] For EmberZNet PRO 6.1.0 or earlier and Silicon Labs Thread 2.5.0 or earlier,** SoC and xNCP/SPI applications require an event to disable sleep/idle. See Section 5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications for details.
- **[2] For EmberZNet PRO 6.2.0 or earlier and Silicon Labs Thread 2.6.0 or earlier,** additional application code is required. See Section 5.3 TX PRIORITY Escalation for details.
- **[3]** For Run-Time API options not supported by selected EmberZNet PRO or Thread release, the corresponding ptaOptions bit fields are RESERVED and must be written to 0.
- **[4]** Requires additional application code. See Section 5.4 Multi-EFR32 Shared "PRIORITY" Signal for details.
- **[5] For EmberZNet PRO 6.2.0 or earlier and Silicon Labs Thread 2.6.0 or earlier,** additional sample code is required. See Section 5.5.4 PWM Run-Time Control for details.
- **[6]** Radio configuration optimization available only for **EFR32xG12**, **EFR32xG13** and **EFR32xG14**. Not available for all other EFR32xG variants.

Note: In stacks prior to **EmberZNet PRO 6.5.0** or **Silicon Labs Thread 2.9.0** or later, 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 or updating to the newest stack; however, manual macros can be added with assistance from Silicon Labs Support if necessary.

4.2.1 AppBuilder Configuration (PTA defaults after reset)

To enable EFR32 PTA coexistence support with **EmberZNet PRO 5.8.0–5.9.2 or Thread 2.1.0–2.2.1**:

1. Open AppBuilder from within Simplicity Studio V4.
2. Open the desired EFR32 application.
3. Select the **Plugins** tab.
4. Under **HAL** plugins, enable the **Coexistence Configuration** plugin, showing the following coexistence configuration options.

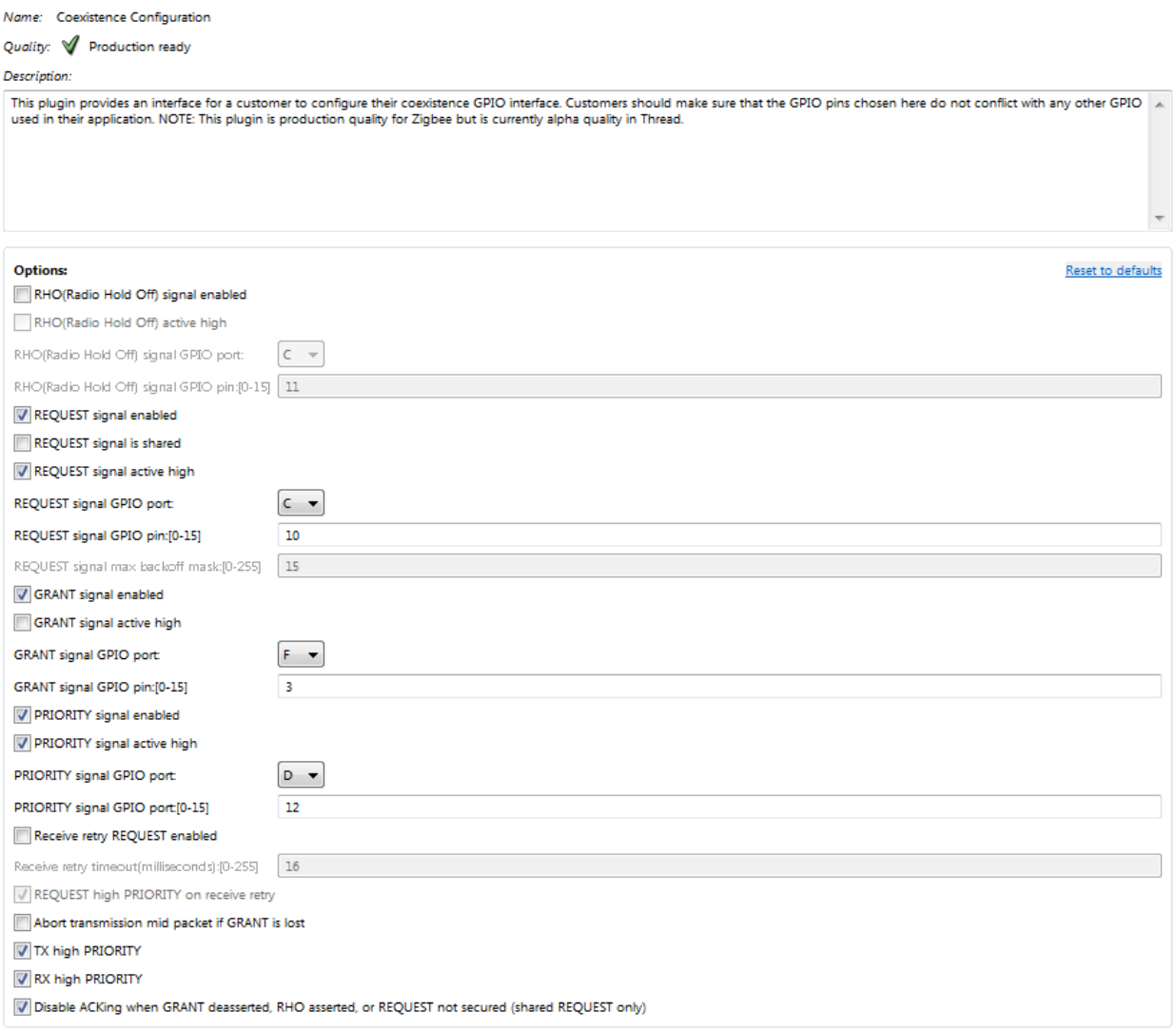
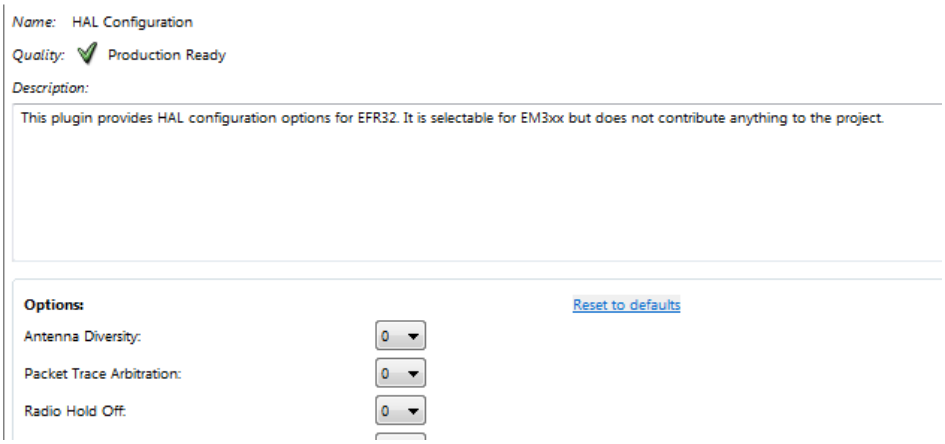


Figure 5. Coexistence Configuration Plugin Settings

In **EmberZNet PRO 5.8.0–5.9.2 and Silicon Labs Thread 2.1.0–2.2.1**, the EFR32 PTA coexistence feature is enabled at firmware startup.

In addition to the above steps to enable EFR32 PTA coexistence support, **EmberZNet PRO 5.10.0–5.10.1 or Silicon Labs Thread 2.3.0–2.3.1** require:

1. Set **Packet Trace Arbitration** in HAL Configuration plugin to **0** (disabled).



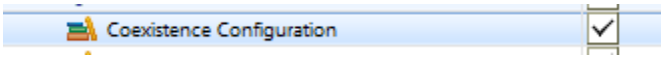
2. Enable the EFR32 PTA coexistence support.
   - For SoC applications:
     - Insert a “halPtaSetEnable(1);” in emberAfMainInitCallback() (see Section 4.2.2.1 SoC Application).
   - For xNCP applications, either:
     - Enable the PTA during xNCP firmware startup:
     - Insert a “halPtaSetEnable(1);” in emberAfMainInitCallback() (see Section 4.2.2.1 SoC Application).
       OR
     - Enable the PTA from Host application.
     - Define the PTA enable value and send an EZSP PTA enable command (see Section Zigbee Network Coprocessor Application using EZSP API):

```
uint8_t ENABLE_PTA_VALUE[1] = {1};
ezspSetValue(EZSP_VALUE_ENABLE_PTA, 1, ENABLE_PTA_VALUE);
```

Note: While coexistence PTA support is also available in EmberZNet PRO 6.0.0 and Silicon Labs Thread 2.4.0, **use in those stacks is not recommended**. In those stacks, the PTA signals can be incorrect, resulting in EFR32 not gaining protected TX or RX and/or incorrectly holding 2.4GHz band from Wi-Fi.

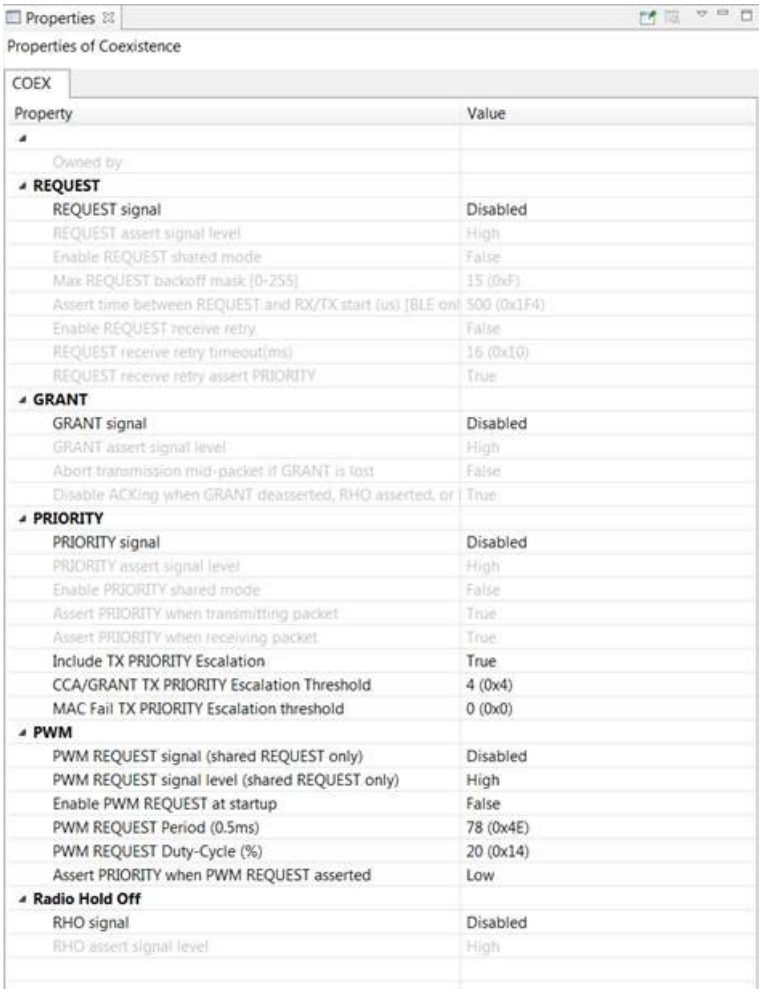
In **EmberZNet PRO 6.1.0 or later** and **Silicon Labs Thread 2.5.0 or later**, the coexistence feature setup moved from the Coexistence Configuration plugin to the Hardware Configurator. To configure the coexistence feature in **EmberZNet PRO 6.1.0 or later** and **Silicon Labs Thread 2.5.0 or later**:

1. Enable the **Coexistence Configuration** plugin under the **HAL** section of the **Plugins** tab.



2. Open the project’s .hwconf file in Hardware Configurator and select **Default Mode Peripherals** view.
3. Ensure **Coexistence** in the **Radio** section is enabled.

4. Double-click **Coexistence** to open coexistence properties.



The **EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later** coexistence properties map to **EmberZNet PRO 5.8.0-5.10.1 and Silicon Labs Thread 2.1.0-2.3.1** Coexistence Configuration plugin as follows.

EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later Hardware Configurator		EmberZNet PRO 5.8.0-5.10.1 and Silicon Labs Thread 2.1.0-2.3.1 Coexistence Configuration
Section	**Property**	**Plugin Field**
REQUEST	REQUEST signal	REQUEST signal enabled REQUEST signal GPIO port REQUEST signal GPIO pin
	REQUEST assert signal level	REQUEST signal active high
	Enable REQUEST shared mode	REQUEST signal is shared
	Max REQUEST backoff mask [0-255]	REQUEST signal max backoff mask[0-255]
	Assert time between REQUEST and RX/TX start (us) [BLE only] ***Note: EFR32MGx BLE coexistence only, no effect on Zigbee or Thread coexistence.***	-
	Enable REQUEST receive retry	Receive retry REQUEST enabled
	REQUEST receive retry timeout(ms)	Receive retry timeout (milliseconds) [0-255]
	REQUEST receive retry assert PRIORITY	REQUEST high PRIORITY on receive retry
GRANT	GRANT signal	GRANT signal enabled

EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later Hardware Configurator		EmberZNet PRO 5.8.0-5.10.1 and Silicon Labs Thread 2.1.0-2.3.1 Coexistence Configuration
Section	**Property**	**Plugin Field**
		GRANT signal GPIO port GRANT signal GPIO pin
	Grant assert signal level	GRANT signal active high
	Abort transmission mid-packet if GRANT is lost	Abort transmission mid packet if GRANT is lost
	Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)	Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)
PRIORITY	PRIORITY signal	PRIORITY signal enabled PRIORITY signal GPIO port PRIORITY signal GPIO pin
	PRIORITY assert signal level	PRIORITY signal active high
	Enable PRIORITY shared mode	See Section 5.4 Multi-EFR32 Shared "PRIORITY" Signal.
	Assert PRIORITY when transmitting packet	TX high PRIORITY
	Assert PRIORITY when receiving packet	RX high PRIORITY
	Include TX PRIORITY Escalation[2]	See Section 5.3 TX PRIORITY Escalation
	CCA/GRANT TX PRIORITY Escalation Threshold[2]	See Section 5.3 TX PRIORITY Escalation
	MAC Fail TX PRIORITY Escalation Threshold[2]	See Section 5.3 TX PRIORITY Escalation
PWM	PWM REQUEST signal (shared REQUEST only)[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
	PWM REQUEST signal level (shared REQUEST only)[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
	Enable PWM REQUEST at startup[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
	PWM REQUEST Period (0.5ms)[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
	PWM REQUEST Duty-Cycle (%)[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
	Assert PRIORITY when PWM REQUEST asserted[3]	See Section 5.5 PWM for High Duty Cycle Wi-Fi
Radio Hold Off	RHO signal	RHO (Radio Hold Off) signal enabled RHO (Radio Hold Off) signal GPIO port RHO (Radio Hold Off) signal GPIO pin
	RHO assert signal level	RHO (Radio Hold Off) active high
Directional Priority	Enable Directional Priority[4]	See Directional PRIORITY in this section.
	Directional Priority Timer[4]	See Directional PRIORITY in this section.
	Microseconds to hold directional priority RX priority pulse[4]	See Directional PRIORITY in this section.
	Directional Priority PRS channel[4]	See Directional PRIORITY in this section.
	PRS channel 3 output pin[4]	See Directional PRIORITY in this section.
	Inverted REQUEST PRS channel[4]	See Directional PRIORITY in this section.
	Inverted RACLNAEN PRS channel[4]	See Directional PRIORITY in this section.
Passive Configuration	Enable radio configuration optimized for radio coexistence[4]	See Passive Configuration.

Notes:

1. The revision 1.1 application note uses the **EmberZNet PRO 5.8.0-5.10.1 and Silicon Labs Thread 2.1.0-2.3.1** Coexistence Configuration Plugin field names to describe the AppBuilder configurable options. Consult the table above for mapping to equivalent **EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later** coexistence proprieties.
2. Feature integrated into **EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later.** For earlier revisions, see Section 5.3 TX PRIORITY Escalation.

3. Feature integrated into **EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later.** For earlier revisions, see Section 5.5 PWM for High Duty Cycle Wi-Fi**.**
4. Feature integrated into **EmberZNet PRO 6.5.0 or later and Silicon Labs Thread 2.9.0 or later.**

Note: In stacks prior to **EmberZNet PRO 6.5.0** or **Silicon Labs Thread 2.9.0** or later, 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 or updating to the newest stack; however, manual macros can be added with assistance from Silicon Labs Support if necessary.

The coexistence configuration AppBuilder configurable options include the following.

RHO (Radio Hold Off)

- **RHO (Radio Hold Off) signal enabled**
  - If selected, RHO is mapped to GPIO pin and is used by PTA implementation.
  - If not selected, RHO is not mapped to GPIO pin and RHO is always deasserted.
- **RHO (Radio Hold Off) active high**
  - If selected, RHO is asserted when RHO GPIO pin is high (> Vih).
  - If not selected, RHO is asserted when RHO GPIO pin is low (< Vil).
- **RHO (Radio Hold Off) signal GPIO port and RHO (Radio Hold Off) signal GPIO pin**
  - Select RHO port and pin matching circuit board configuration.
  - To minimize PTA impact to other EFR32 peripherals, recommended RHO port and pin are:

EFR32 Package	RHO Port/Pin
QFN48	PC11
QFN32	PC10

REQUEST

- **REQUEST signal enabled**
  - If selected, REQUEST is mapped to GPIO pin and is used by PTA implementation.
  - If not selected, REQUEST is not mapped to GPIO pin.
- **REQUEST signal is shared**
  - In **EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0** REQUEST GPIO pin is always configured as push-pull regardless of the shared setting and should not be used in multi-EFR32 configurations.
  - If selected. REQUEST is shared and implements open-drain or open-source I/O for multi-EFR32 radio applications.
  - If active low, REQUEST is open-drain and an external 1 kΩ ±5% pull-up is required.
  - If active high, REQUEST is open-source and an external 1 kΩ ±5% pull-down is required.
  - If not selected, REQUEST is not shared and implements a push-pull output for single EFR32 radio applications.
- **REQUEST signal active high**
  - If selected, REQUEST GPIO pin is driven high (> Voh) when REQUEST is asserted.
  - If not selected, REQUEST GPIO pin is driven low (< Vol) when REQUEST is asserted.
- **REQUEST signal GPIO port and REQUEST signal GPIO pin**
  - Select REQUEST port and pin matching circuit board configuration.
  - To minimize PTA impact to other EFR32 peripherals, recommended REQUEST port and pin are:

EFR32 Package	REQUEST Port/Pin
QFN48	PC10
QFN32	PC11

REQUEST signal max backoff mask[0-255]

- REQUEST signal max backoff determines the random REQUEST delay mask (only valid if REQUEST signal is shared).
- 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 2^n-1 to insure a continuous random delay range.

GRANT

- **GRANT signal enabled**
  - If selected, GRANT is mapped to GPIO pin and is used by PTA implementation.
  - If not selected, GRANT is not mapped to GPIO pin and GRANT is always asserted.
- **GRANT signal active high**
  - If selected, GRANT is asserted when GRANT GPIO pin is high (> Vih).
  - If not selected, GRANT is asserted when GRANT GPIO pin is low (< Vil).
- **GRANT signal GPIO port and GRANT signal GPIO pin**
  - Select GRANT port and pin matching circuit board configuration.
  - To minimize PTA impact to other EFR32 peripherals, recommended GRANT port and pin are:

EFR32 Package	GRANT Port/Pin
QFN48	PF3
QFN32	PB15

- **Abort transmission mid packet if GRANT is lost**
  - **Do not attempt to use in EmberZNet PRO 5.8.0–5.9.1 or Silicon Labs Thread 2.1.0–2.2.1 (firmware crash possible).**
  - If selected, losing GRANT during an 802.15.4 TX will abort the 802.15.4 TX.
  - If not selected, losing GRANT after the initial evaluation at end of CCA will not abort the 802.15.4 TX.

PRIORITY

- **PRIORITY signal enabled**
  - If selected, PRIORITY is mapped to GPIO pin and is used by PTA implementation.
  - If not selected, PRIORITY is not mapped to GPIO pin.
- **PRIORITY signal active high**
  - If selected, PRIORITY GPIO pin is driven high (> Voh) when PRIORITY is asserted.
  - If not selected, PRIORITY GPIO pin is driven low (< Vol) when PRIORITY is asserted.
- **PRIORITY signal GPIO port** and **PRIORITY signal GPIO pin**
  - Select REQUEST port and pin matching circuit board configuration.
  - To minimize PTA impact to other EFR32 peripherals, recommended PRIORITY port and pin are:

EFR32 Package	PRIORITY Port/Pin
QFN48	PD12
QFN32	PB14

- **Enable PRIORITY shared mode**

  Note: Shared PRIORITY is directly selectable in **EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later**. For earlier stacks, see Section 5.4 Multi-EFR32 Shared “PRIORITY” Signal.

  - In **Em**berZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0, Priority GPIO pin is always configured as push-pull regardless of shared setting and should not be used in multi-EFR32 configurations.
  - If enabled. PRIORITY is shared and implements open-drain or open-source I/O for multi-EFR32 radio applications.
  - If active low, PRIORITY is open-drain and an external 1 kΩ ±5% pull-up is required.
  - If active high, PRIORITY is open-source and an external 1 kΩ ±5% pull-down is required.
  - If not enabled, PRIORITY is not shared and implements a push-pull output for single EFR32 radio applications.
- **TX high PRIORITY**
  - If selected, PRIORITY is asserted during 802.15.4 TX.
  - If not selected, PRIORITY is deasserted during 802.15.4 TX.

- **RX high PRIORITY**
  - If selected, PRIORITY is asserted during 802.15.4 RX.
  - If not selected, PRIORITY is deasserted during 802.15.4 RX.
- **Include TX PRIORITY Escalation**

  Note: TX PRIORITY Escalation is directly selectable in EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later. For earlier SDKs, see Section 5.3 TX PRIORITY Escalation**.**

  - If enabled. TX PRIORITY Escalation feature is compiled into firmware.
  - If not enabled, TX PRIORITY Escalation feature is not compiled into firmware and CCA/GRANT TX PRIORITY Escalation Threshold and MAC Fail TX PRIORITY Escalation Threshold must be set to 0 when writing to ptaOptions via run-time API.
- **CCA/GRANT TX PRIORITY Escalation Threshold**
  - If set to 0 (000b, default):
    - CCA/GRANT TX PRIORITY Escalation is disabled.
    - PRIORITY during TX is asserted as per “TX high PRIORITY” setting.
  - If set between n=1 (001b) to 7 (111b) [requires “TX high PRIORITY” set to low priority (0)]:
    - CCA/GRANT TX PRIORITY Escalation is enabled.
    - PRIORITY during TX becomes asserted high after n MAC failures due to four CCA and/or GRANT denial failures.
    - PRIORITY during TX remains asserted high until a successful MAC TX and RX ACK.
- **MAC Fail TX PRIORITY Escalation Threshold**
  - If set to 0 (00b, default):
    - CCA/GRANT TX PRIORITY Escalation is disabled.
    - PRIORITY during TX is asserted as per “TX high PRIORITY” setting.
  - If set to n=1 (01b) to 3 (11b) [requires “TX high PRIORITY” set to low priority (0)]:
    - CCA/GRANT TX PRIORITY Escalation is enabled.
    - PRIORITY during TX is asserted high after n MAC failures due to CCA (four CCA failures) or MAC ACK fails (four MAC TX and RX ACK failures).
    - PRIORITY during TX remains asserted high until a successful MAC TX and RX ACK.

PWM

Note: PWM is directly selectable in EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later. For earlier SDKs, see Section 5.5 PWM for High Duty Cycle Wi-Fi**.**

- **PWM REQUEST signal (shared REQUEST only)**
  - In **EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0,** if REQUEST PWM feature is enabled and shared REQUEST is enabled, the PWM signal is driven out both REQUEST and PWM_REQUEST, when it is only expected at PWM_REQUEST. **EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0** should not be used in multi-EFR32 configurations.
  - If REQUEST signal is NOT shared, PWM REQUEST signal must be set to “Disabled”.
  - If REQUEST signal is shared, the REQUEST GPIO pin is used to arbitrate REQUEST between multiple EFR32 radios and a second GPIO is required to drive REQUEST||PWM. PWM REQUEST signal specifies the REQUEST||PWM GPIO.
- **PWM REQUEST signal level (shared REQUEST only)**
  - If PWM REQUEST signal is “Disabled”, then PWM REQUEST signal level selection is ignored. Else, PWM REQUEST signal is shared and implements open-drain or open-source I/O for multi-EFR32 radio applications.
  - If active low, PWM REQUEST is open-drain, an external 1 kΩ ±5% pull-up is required, and PWM REQUEST GPIO pin is driven low (< Vol) when REQUEST||PWM is asserted.
  - If active high, PWM REQUEST is open-source, an external 1 kΩ ±5% pull-down is required, and PWM REQUEST GPIO pin is driven high (> Voh) when REQUEST||PWM is asserted.
- **Enable PWM REQUEST at startup**
  - If enabled. PWM REQUEST is enabled at firmware startup as per specified period, duty-cycle, and priority.
  - If not enabled, PWM REQUEST is enabled at firmware startup, but can be enabled via run-time API.

- **PWM REQUEST Period (0.5ms)**

  Note: PWM REQUEST Period selection cannot be an integer sub-multiple of the Wi-Fi beacon or a significant number of consecutive Wi-Fi beacons may be missed, causing AP to collapse Wi-Fi network or STA to disassociate. Silicon Labs achieves <1% 802.15.4 message receive loss with PWM REQUEST set to 39ms (or 78 half-ms) period, 20% duty-cycle, and high priority, which results in ~30% reduction in Wi-Fi TCP throughput over MCS0 to MCS7 and 20 or 40MHz bandwidth.

  - Sets PWM REQUEST Period from 5ms (10) to 109ms (218) in 0.5ms steps.
- **PWM REQUEST Duty-Cycle (%)**

  Note: Large PWM REQUEST Duty-Cycle selection will substantially impact the Wi-Fi throughput as it reserved more time for 802.15.4 listening. Silicon Labs achieves <1% 802.15.4 message receive loss with PWM REQUEST set to 39ms (or 78 half-ms) period, 20% duty-cycle, and high priority, which results in ~30% reduction in Wi-Fi TCP throughput over MCS0 to MCS7 and 20 or 40MHz bandwidth.

  - Sets PWM REQUEST Duty-Cycle from 1% to 95% in 1% steps.
- **Assert PRIORITY when PWM REQUEST asserted**
  - Sets PWM REQUEST PRIORITY to assert or not when PWM REQUEST asserts.
- **Receive Retry**
  - **Receive retry REQUEST enabled**
    - While also available in **EmberZNet PRO 5.8.0–5.9.1 and Silicon Labs Thread 2.1.0–2.2.1, use in those stacks is not recommended**. If radio receives a unicast packet destined for a different radio, the “Receive retry REQUEST” feature incorrectly triggers and holds the 2.4GHz band.
    - In SoC and xNCP/SPI applications using **EmberZNet PRO 6.1.0 or earlier and Silicon Labs Thread 2.5.0 or earlier** REQUEST can be held far longer than the programmed time-out, due to device sleep/idle triggering during REQUEST hold. This long REQUEST hold can be avoided by creating an application event that prevents sleep/idle. See Section 5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications for details.
    - If selected, REQUEST is held after a corrupted receive packet or, in **EmberZNet PRO 6.2.0 or later and Silicon Labs Thread 2.6.0 or later**, after a successful receive packet with GRANT denied until time-out expires or another packet is received.
      Note: This feature is useful to hold 2.4GHz band clear while remote device re-transmits a packet, maximizing the opportunity to receive an uncorrupted retry packet from remote device, reducing 2.4GHz RF traffic and improving battery life.
    - If not selected, REQUEST is not held after a corrupted receive packet or, in **EmberZNet PRO 6.2.0 or later and Silicon Labs Thread 2.6.0 or later**, after a successful receive packet with GRANT denied.
  - **Receive retry timeout (milliseconds) [0-255]**
    - Selects the timeout for REQUEST hold after a corrupted receive packet.
      Note: 16ms is recommended to allow for maximum 802.15.4 packet duration and MAC retry random delay.
      Note: Many Wi-Fi/PTA implementations have a maximum GRANT timeout, which should be set to received retry timeout plus 6ms to allow for maximum size corrupted packet, maximum random delay, and maximum size retry packet.
  - **REQUEST high PRIORITY on receive retry**
    - If selected, PRIORITY is asserted during REQUEST hold after a corrupted receive packet or, in **EmberZNet PRO 6.2.0 or later and Silicon Labs Thread 2.6.0 or later**, after a successful receive packet with GRANT denied.
    - If not selected, PRIORITY is deasserted during REQUEST hold after a corrupted receive packet or, in **EmberZNet PRO 6.2.0 or later and Silicon Labs Thread 2.6.0 or later**, after a successful receive packet with GRANT denied.

Other

- **Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)**
  - If selected, the ACK to a valid RX packet, requiring an ACK, is **not** transmitted if GRANT is deasserted, RHO is asserted, or REQUEST is not secured (shared REQUEST only).
    Note: This feature allows completing an 802.15.4 message, regardless of PTA signals, in order to minimize additional retries from remote device, reducing 2.4GHz RF traffic and improving battery life.
  - If not selected, the ACK to a valid RX packet requiring an ACK is transmitted regardless of GRANT, RHO, or REQUEST state.

Directional PRIORITY

Notes

1. Directional PRIORITY is available for applications using **EmberZNet PRO 6.5.0 or later** and **Silicon Labs Thread 2.9.0** or later.
2. **PRIORITY** signal must be enabled for implementing **Directional PRIORITY**.
3. **Directional PRIORITY** requires the use of an additional GPIO, one Timer and 6 PRS channels. 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.

The following two figures show the block diagrams for Single-EFR32 PTA with Directional PRIORITY.

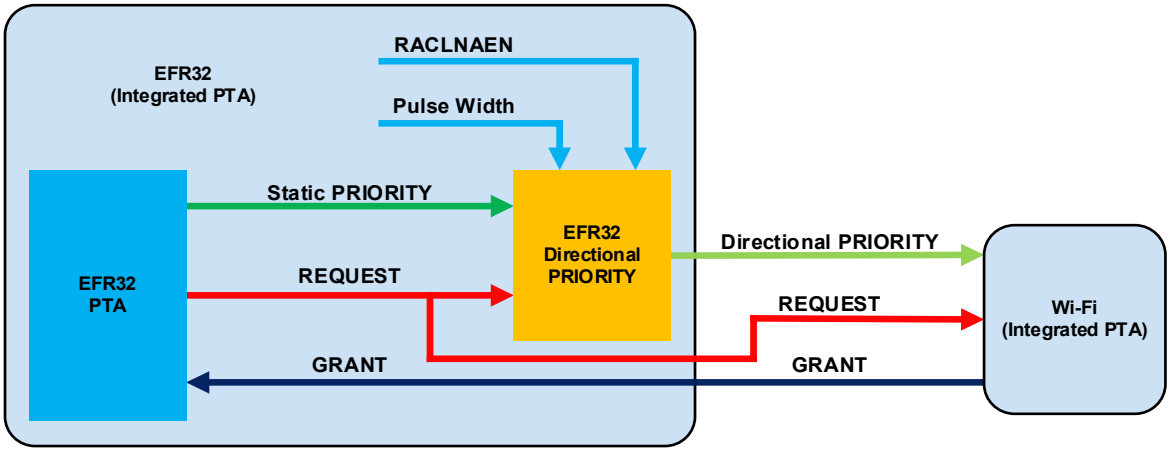


Figure 6. Single-EFR32 PTA without SDK PWM, Directional PRIORITY

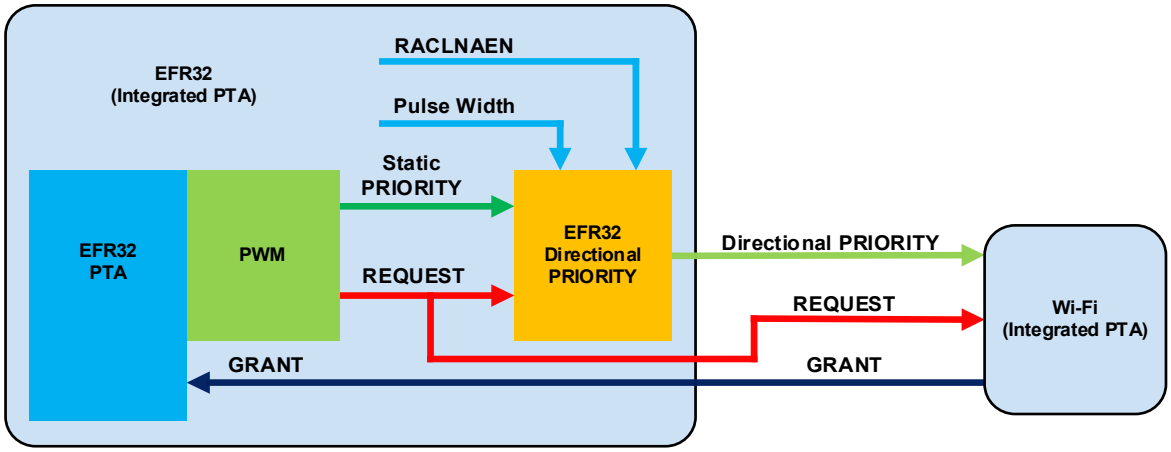


Figure 7. Single-EFR32 PTA with SDK PWM, Directional PRIORITY

GPIO, Radio and Timer signals for Single-EFR32 PTA with or without SDK PWM, Directional PRIORITY.

Directional PRIORITY

- EFR32 Directional Priority output.
- GPIO connects to Wi-Fi PTA.

Static PRIORITY

- EFR32 PTA output.
- Directional PRIORITY input.
- Assign to any unused 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.

RACLNAEN

- EFR32 radio receives LNA enable output.
- Directional PRIORITY input.

Pulse Width

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

The following figure shows the block diagram for Multi-EFR32 PTA with Directional PRIORITY.

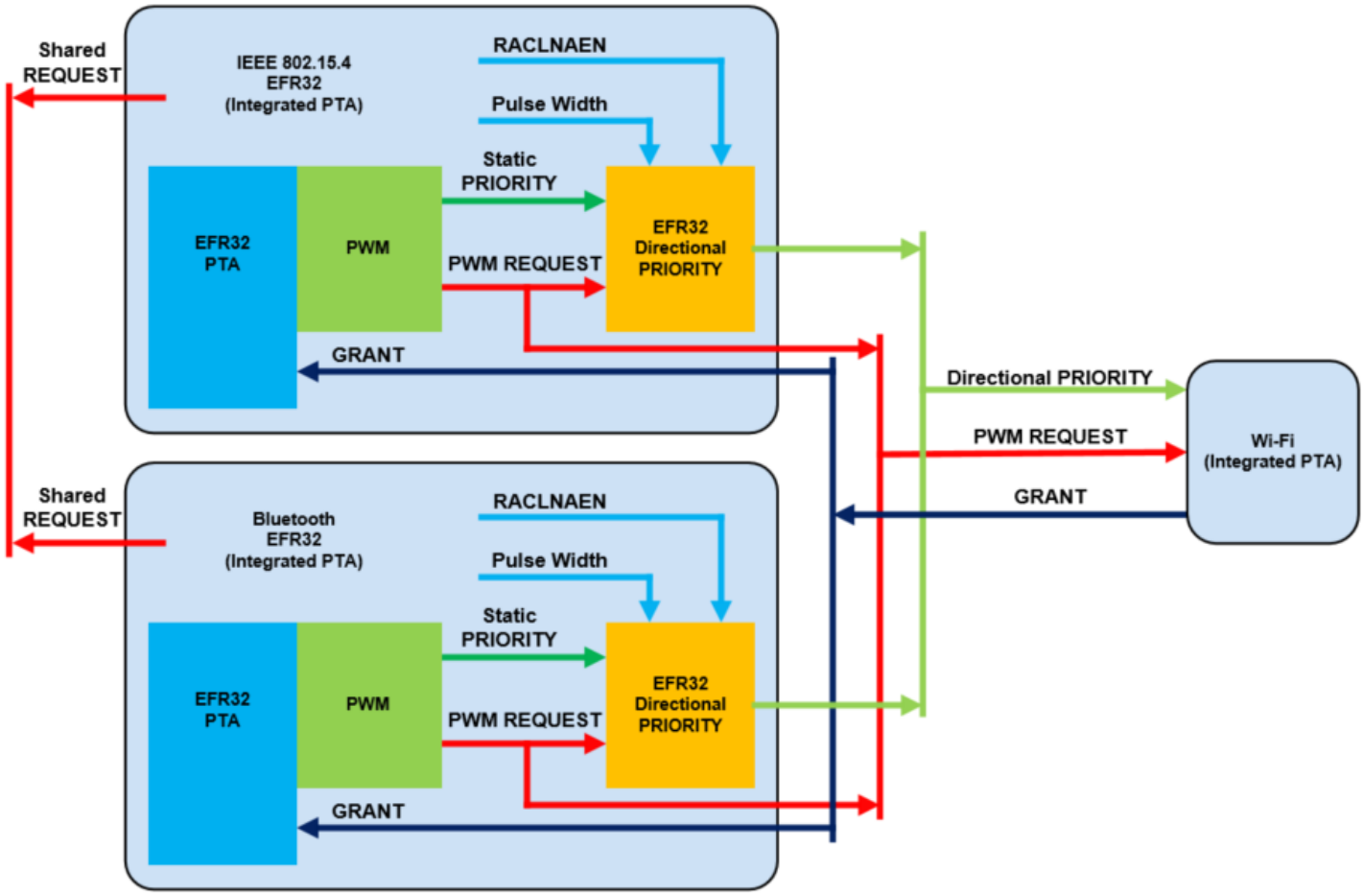


Figure 8. Multi-EFR32 PTA with SDK PWM, Directional PRIORITY (1.5K pull-up / down resistors not shown)

GPIO, Radio and Timer signals for Multi-EFR32 PTA with SDK PWM, Directional PRIORITY.

Shared REQUEST

- EFR32 PTA input / output.
- GPIO connects to all EFR32s for PTA bus arbitration between EFR32s
  - Configured as open source / drain.
  - If active low, Shared REQUEST is open-drain and an external 1 kΩ ±5% pull-up is required.
  - If 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.
    - If active low, Directional PRIORITY is open-drain and an external 1 kΩ ±5% pull-up is required.
    - If 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.
- Assign to any unused 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 low, PWM REQUEST is open-drain and an external 1 kΩ ±5% pull-up is required.
    - 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.

RACLNAEN

- EFR32 Radio Receives LNA 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.

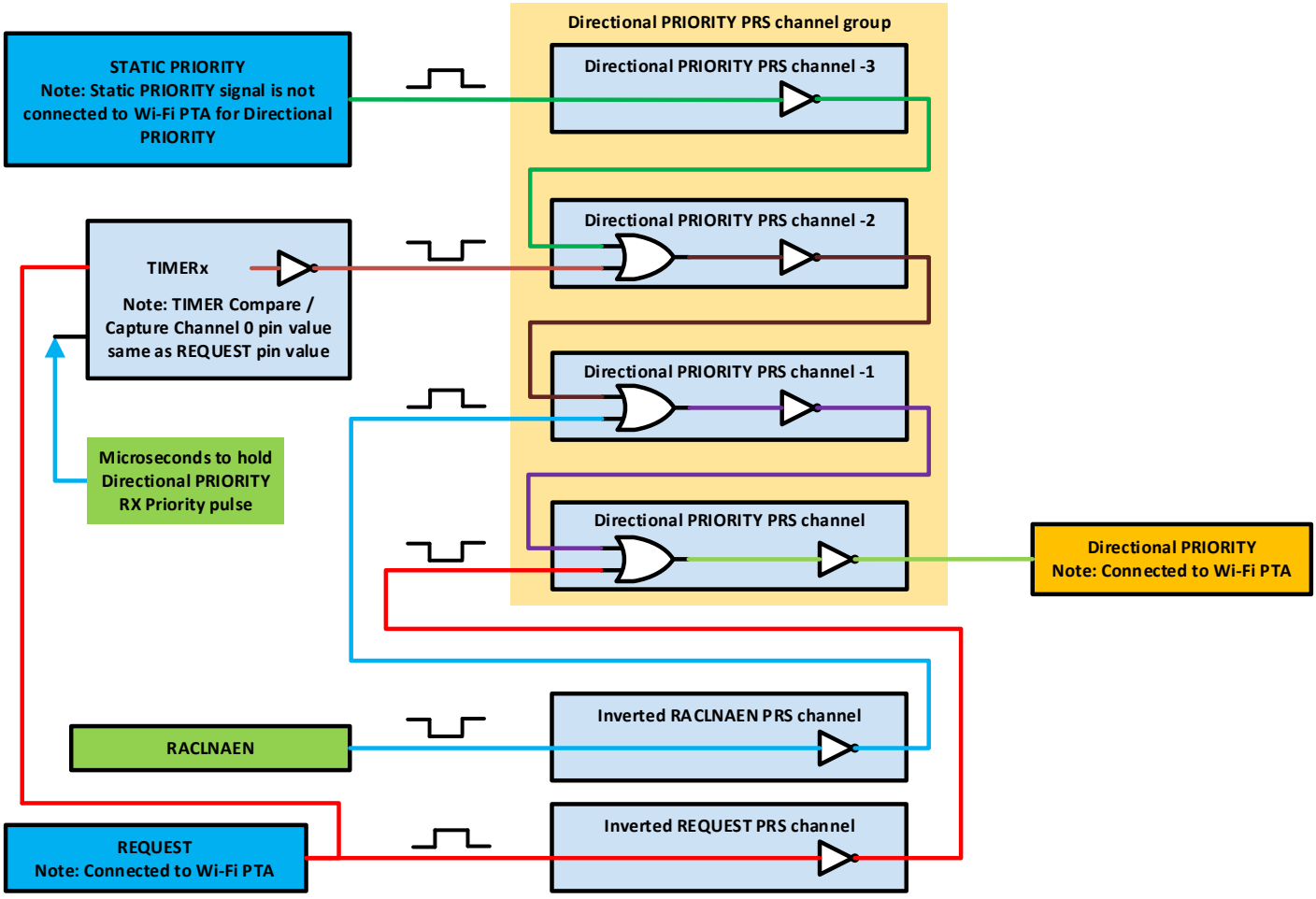


Figure 9. EFR32 Directional PRIORITY Logic Diagram

Directional PRIORITY enabled

- If selected:
  - Directional PRIORITY signal is connected to the Wi-Fi PTA and multiplexes priority state and radio state information.
  - Allows the Wi-Fi PTA master to obtain radio state information from the EFR32 using the Directional PRIORITY signal.
  - When requesting network air time, the EFR32 will assert a pulse on the Directional PRIORITY line depending on the requirement of that transaction and then switch to communicating the state of the radio on the same Directional PRIORITY line.
  - The Directional PRIORITY line is held low when the radio is in receive mode and is held high when the radio is in transmit mode.
  - The PRIORITY signal is not connected to the Wi-Fi PTA and is used as a Static PRIORITY input to the Directional PRIORITY logic block.
- If not selected:
  - The Directional PRIORITY signal is not used or connected to the Wi-Fi PTA.
  - The PRIORITY signal is connected to the Wi-Fi PATA and operates as Static PRIORITY and is either high or low during REQUEST asserted for the transmit or receive operation.

Directional PRIORITY Timer

To configure the Directional PRIORITY Timer:

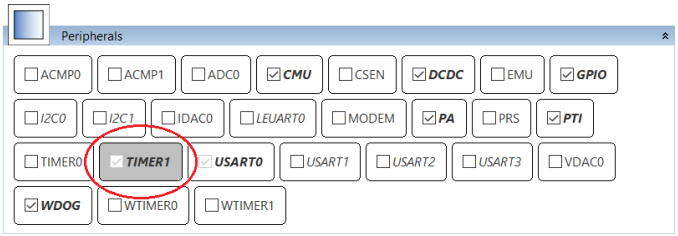
1. Choose an unused Timer.

   TIMER1 is recommended for most EmberZNet PRO or Silicon Labs Thread applications due to TIMER0 is used by the IEEE 802.15.4 software stacks.

2. Select the **Timer Compare / Capture Channel** pin.
3. Open Hardware Configurator.

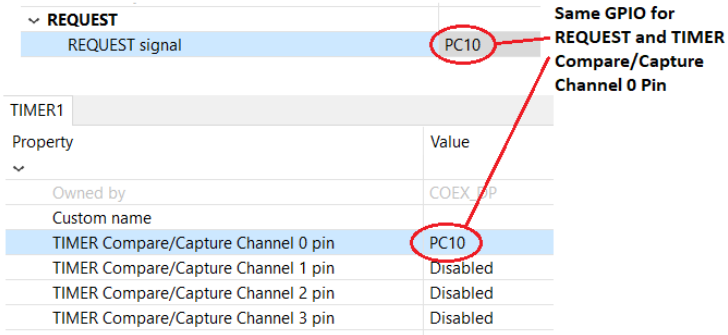
4. Open the corresponding .hwconf file for the application being built.
5. On the **Configurator** tab, select the **Default Mode Peripherals** view.
6. Select the same TIMERx as selected in the Directional PRIORITY section of the coexistence plugin.

   TIMER1 is shown in this example but any available Timer can be used.



7. Change the **TIMER Compare/Capture Channel 0 pin** to match the same GPIO pin used for REQUEST.

   PC10 is shown in this example but other GPIOs can be selected. Refer to the EFR32 datasheet or reference manual to confirm GPIO availability for the TIMERx Compare/Capture Channel 0 pin.



Directional PRIORITY RX Priority pulse [0-255]

- Set to 20 (0x14, default).
- Selects the hold time of the Directional PRIORITY RX Priority pulse in microseconds for a range of 1 to 254 depending on the requirement of the Wi-Fi PTA. Silicon Labs recommends the default of 20 microseconds for typical Wi-Fi PTA implementations.

Directional Priority PRS Channel

To configure the Directional PRIORITY PRS Channel:

1. Choose any group of four PRS channels not currently used by the SDK stack software, other plugins, or custom code.
2. Assign the highest PRS channel number from this group to the Directional Priority PRS Channel value.

   Note: The External FEM plugin is recommended to be enabled for monitoring the EFR32 radio TX activity and radio RX activity for the custom coexistence application test and development purposes and can use up to two additional PRS channels.

3. The SDK stack software automatically selects the preceding three PRS channels from the group.

   Example: Designer selects PRS channel 2 as the Directional Priority PRS Channel value. From the designer's choice, the SDK stack software automatically selects PRS channel 1, PRS channel 0 and PRS channel 11 for use in the Directional Priority PRS Channel group. In this example, the SDK stack software automatically wraps around from the lowest PRS channel number to the highest PRS channel number until all three additional required PRS channels are assigned.

PRS Channel Output Pin

- Directional PRIORITY output GPIO pin
- Connects to the Wi-Fi PTA.

Inverted Request PRS Channel

- Choose any PRS channel not used by the SDK stack software, other plugins, custom code or the Directional Priority PRS Channel option.

Inverted RACLNAEN PRS Channel

- Choose any PRS channel not used by the SDK stack software, other plugins, custom code, the Directional Priority PRS Channel option or the Inverted Request PRS Channel option.

Passive Configuration

Notes

1. Radio configuration optimization is available for applications using **EmberZNet PRO 6.5.0 or later** and **Silicon Labs Thread 2.9.0** or later and is available only for **EFR32xG12**, **EFR32xG13** and **EFR32xG14** EFR32 devices. Radio configuration optimization is not available for all other EFR32xG variants.
2. It is highly recommended to enable Passive Configuration on **EFR32xG12**, **EFR32xG13** and **EFR32xG14** devices when co-located with a Wi-Fi device such as in a Gateway.

Enable Radio Configuration Optimized for Coexistence

- If selected:
  - IEEE802.15.4 PHY performs better in the presence of Wi-Fi interference through improved adjacent channel rejection and improved far channel rejection.
  - Receive sensitivity is slightly reduced.
- If not selected, IEEE 802.15.4 PHY is optimized for Receive sensitivity and will result in reduced adjacent channel rejection and far channel rejection performance in the presence of Wi-Fi interference.

This completes the Coexistence Plugin / Configurator setup. Complete other AppBuilder application setups and generate. The coexistence configuration is saved in the application's .h file.

4.2.2 Run-Time PTA Re-configuration

The following PTA options, which can be configured at compile time via AppBuilder, can also be re-configured at run-time:

- Receive retry timeout (milliseconds) [0-255]
- Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)
- Abort transmission mid packet if GRANT is lost
- TX high PRIORITY
- RX high PRIORITY
- REQUEST high PRIORITY on receive retry
- Receive retry REQUEST enabled
- RHO(Radio Hold Off) signal enabled
- CCA/GRANT TX PRIORITY Escalation Threshold
- MAC Fail TX PRIORITY Escalation Threshold
- PWM REQUEST

Note: For descriptions of the above PTA options fields, see Section 4.2.1 AppBuilder Configuration (PTA defaults after reset).

The following PTA options cannot be configured via AppBuilder and can only be configured at run-time:

- Enable or disable PTA
- Disable REQUEST (force holdoff)
- Synch MAC to GRANT (MAC holdoff)
- REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)

The descriptions of the above PTA options fields follow.

Disable REQUEST (force holdoff)

- If not set (default), REQUEST operates as per descriptions in Section 4.1.1 Single EFR32 Connected to Wi-Fi/PTA and Section 4.1.2 Multiple EFR32s Connected to Wi-Fi/PTA.
- If set, REQUEST stays disabled, effectively halting all radio TX/RX functions.

Synch MAC to GRANT (MAC holdoff)

- **Do not attempt use in EmberZNet PRO 5.9.0 and Silicon Labs Thread 2.2.0 (firmware crash possible).**
- If not set (default), Synch MAC to GRANT is disabled for 802.15.4-compliant random MAC delays.
- If set, MAC CCA/TX is delayed until GRANT is asserted, synching all Zigbee TX operations with GRANT.
  - Synch MAC to GRANT is not strictly 802.15.4 compliant as it prevents random MAC delay execution.
  - Synch MAC to GRANT should only be enabled during known, higher priority, Wi-Fi or BT interfering activity and disabled as soon as such activity completes.

REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)

- If set to 0 (00b, default and recommended):
  - REQUEST during RX is asserted at Preamble/Synch.
  - PRIORITY during RX is asserted at Preamble/Synch, as per “RX high PRIORITY” setting.
- If set to 1 (01b) or 3 (11b) [requires “RX high PRIORITY” set to high priority (1)]:
  - REQUEST during RX is asserted at Address Detection (for this radio).
  - PRIORITY during RX is asserted at Address Detection (for this radio).
- If set to 2 (10b) [requires “RX high PRIORITY” set to low priority (0)]:
  - REQUEST during RX is asserted at Preamble/Synch.
  - PRIORITY during RX is asserted at Address Detection (for this radio).

The API function calls for re-configuring coexistence PTA vary based on SoC, EZSP, or TMSP application.

Note: **For Run-Time API options not supported by selected EmberZNet PRO or Silicon Labs Thread release, the corresponding ptaOptions bit fields are RESERVED and must be written to 0**. See Table 2 in PTA Support Software Setup for details on options supported relative to releases.

4.2.2.1 SoC Application

Note: **For EmberZNet PRO 6.5.0 and Silicon Labs Thread 2.9.0** or later, add the following # include into the application's <xxx>-callbacks.c file to avoid warnings and errors at build time associated with API function calls described in this section:

```
#include "platform/radio/rail_lib/plugin/coexistence/protocol/ieee802154/coexistence-802154.h";
```

The following two SoC API function calls enable and disable the PTA at run-time:

```
bool halPtaIsEnabled(void);
EmberStatus halPtaSetEnable(bool enabled);
```

The following two SoC API function calls re-configure the PTA at run-time:

```
HalPtaOptions halPtaGetOptions(void);
EmberStatus halPtaSetOptions(HalPtaOptions options);
```

Where `HalPtaOptions` is a uint32_t with the following bitmap definition:

PTA Feature	Bit Position	Size (bits)
Receive retry timeout (milliseconds) [0-255]	0	8
Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)	8	1
Abort transmission mid packet if GRANT is lost	9	1
TX high PRIORITY	10	1
RX high PRIORITY	11	1
REQUEST high PRIORITY on receive retry	12	1
Receive retry REQUEST enabled • For **EmberZNet PRO 6.1.0 or earlier and Silicon Labs Thread 2.5.0 or earlier**, SoC and xNCP/SPI applications require an event to disable sleep/idle. • See Section 5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications for details.	13	1

PTA Feature	Bit Position	Size (bits)
RHO(Radio Hold Off) signal enabled	14	1
Reserved (Reserved bits MUST be written 0)	15	1
Disable REQUEST (force holdoff)	16	1
Synch MAC to GRANT (MAC holdoff)	17	1
REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)	18	2
• CCA/GRANT TX PRIORITY Escalation Threshold • SDKs **earlier than EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0** require additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	20	3
Reserved (Reserved bits MUST be written 0)	23	2
• MAC Fail TX PRIORITY Escalation Threshold • SDKs **earlier than EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0** require additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	25	2
Reserved (Reserved bits MUST be written 0)	27	5

The following two SoC API function calls re-configure the PWM REQUEST at run-time:

```
const HalPtaPwmArgs_t *halPtaGetRequestPwmArgs(void);
EmberStatus halPtaSetRequestPwm(halPtaReq_t ptaReq, halPtaCb_t ptaCb, uint8_t dutyCycle, uint8_t
periodHalfMs);
```

Where:

```
typedef struct HalPtaPwmArgs {
  halPtaReq_t req;
  uint8_t dutyCycle;
  uint8_t periodHalfMs;
} HalPtaPwmArgs_t;
```

and

```
ptaReq/req:   0x00 => PWM REQUEST disabled
              0x80 => PWM REQUEST enabled at low priority
              0x82 => PWM REQUEST enabled at high priority
ptaCb: NULL
```

Where:

`dutyCycle`: PWM REQUEST duty-cycle from 5% to 95% in 1% steps

`periodHalfMs`: PWM REQUEST Period from 5ms (10) to 109ms (218) in 0.5ms steps

The following two SoC API function calls re-configure the Directional PRIORITY at run-time:

```
dp_pulse = halPtaGetDirectionalPriorityPulseWidth();
halPtaSetDirectionalPriorityPulseWidth(dp_pulse);
```

Where:

`uint8_t dp_pulse`: Pulse width (0 to disable, 1-255µs)

4.2.2.2 Zigbee Network Coprocessor Application using EZSP API

The following two EZSP (EmberZNet Serial Protocol) API function calls enable and disable the PTA, re-configure the PTA, and reconfigure the PWM REQUEST at run-time:

```
EzspStatus ezspGetValue(EzspValueId valueId, uint8_t *valueLength, uint8_t *value);
EzspStatus ezspSetValue(EzspValueId valueId, uint8_t valueLength, uint8_t *value);
```

Where `valueId` and `valueLength` have the following PTA related options:

EZSP Value ID	Value	Length (bytes)	Description
EZSP_VALUE_ENABLE_PTA	0x31	1	Enable (1) or disable (0) packet traffic arbitration.
EZSP_VALUE_PTA_OPTIONS	0x32	4	Set packet traffic arbitration (PTA) configuration options.
EZSP_VALUE_PTA_PWM_OPTIONS	0x35	3	Configure PWM REQUEST options.
EZSP_VALUE_PTA_DIRECTIONAL_PRIORITY_PULSE_WIDTH	0x36	1	Pulse width (0 to disable, 1-255µs)

Where PTA configuration options are a uint32_t with the following bitmap definition:

PTA Feature	Bit Position	Size (bits)
Receive retry timeout (milliseconds) [0-255]	0	8
Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)	8	1
Abort transmission mid packet if GRANT is lost	9	1
TX high PRIORITY	10	1
RX high PRIORITY	11	1
REQUEST high PRIORITY on receive retry	12	1
Receive retry REQUEST enabled • For **EmberZNet PRO 6.1.0 or earlier**, SoC and xNCP/SPI applications require an event to disable sleep/idle. • See Section 5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications for details.	13	1
RHO(Radio Hold Off) signal enabled	14	1
Reserved (Reserved bits MUST be written 0)	15	1
Disable REQUEST (force holdoff)	16	1
Synch MAC to GRANT (MAC holdoff)	17	1
REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)	18	2
• CCA/GRANT TX PRIORITY Escalation Threshold • Requires additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	20	3
Reserved (Reserved bits MUST be written 0)	23	2
• MAC Fail TX PRIORITY Escalation Threshold • Requires additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	25	2
Reserved (Reserved bits MUST be written 0)	27	5

PWM REQUEST configuration options is a three-byte uint8_t array:

```
{uint8_t ptaReq, uint8_t dutyCycle, uint8_t periodHalfMs}
```

Where:

`ptaReq`: 0x00 => PWMp REQUEST disabled

0x80 => PWM REQUEST enabled at low priority

0x82 => PWM REQUEST enabled at high priority

`dutyCycle`: PWM REQUEST duty-cycle from 5% to 95% in 1% steps

`periodHalfMs`: PWM REQUEST Period from 5ms (10) to 109ms (218) in 0.5ms steps

And the Directional PRIORITY parameter is:

`uint8_t dp_pulse`: Pulse width (0 to disable, 1-255µs)

4.2.2.3 Thread Network Coprocessor Application using TMSP API

The following two TMSP (Thread Management Serial Protocol) API function calls enable and disable the PTA at run-time:

```
bool halPtaIsEnabled(void);
EmberStatus halPtaSetEnable(bool enabled);
```

The following two TMSP API function calls re-configure the PTA at run-time:

```
HalPtaOptions halPtaGetOptions(void);
EmberStatus halPtaSetOptions(HalPtaOptions options);
```

Where `HalPtaOptions` is a uint32_t with the following bitmap definition:

PTA Feature	Bit Position	Size (bits)
Receive retry timeout (milliseconds) [0-255]	0	8
Disable ACKing when GRANT deasserted, RHO asserted, or REQUEST not secured (shared REQUEST only)	8	1
Abort transmission mid packet if GRANT is lost	9	1
TX high PRIORITY	10	1
RX high PRIORITY	11	1
REQUEST high PRIORITY on receive retry	12	1
Receive retry REQUEST enabled • For Silicon Labs Thread 2.5.0 or earlier, SoC and xNCP/SPI applications require an event to disable sleep/idle. • See Section 5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications for details.	13	1
RHO(Radio Hold Off) signal enabled	14	1
Reserved (Reserved bits MUST be written 0)	15	1
Disable REQUEST (force holdoff)	16	1
Synch MAC to GRANT (MAC holdoff)	17	1
REQUEST/PRIORITY Assert (Preamble/Synch or Address Detection)	18	2
• CCA/GRANT TX PRIORITY Escalation Threshold • Requires additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	20	3
Reserved (Reserved bits MUST be written 0)	23	2
• MAC Fail TX PRIORITY Escalation Threshold • Requires additional application code. • See Section 5.3 TX PRIORITY Escalation for details.	25	2
Reserved (Reserved bits MUST be written 0)	27	5

The following two TMSP API function calls re-configure the PWM REQUEST at run-time:

```
const HalPtaPwmArgs_t *getPtaPwmOptions(void);
EmberStatus setPtaPwmOptions(uint8_t request, uint8_t dutyCycle, uint8_t periodHalfMs);
```

Where:

```
typedef struct HalPtaPwmArgs {
  halPtaReq_t req;
  uint8_t dutyCycle;
  uint8_t periodHalfMs;
} HalPtaPwmArgs_t;
```

and

`req/request`: 0x00 => PWM REQUEST disabled

0x80 => PWM REQUEST enabled at low priority

0x82 => PWM REQUEST enabled at high priority

`dutyCycle`: PWM REQUEST duty-cycle from 5% to 95% in 1% steps

`periodHalfMs`: PWM REQUEST Period from 5ms (10) to 109ms (218) in 0.5ms steps

The following two TMSP API function calls re-configure the Directional PRIORITY at run-time:

```
dp_pulse = halPtaGetDirectionalPriorityPulseWidth();
halPtaSetDirectionalPriorityPulseWidth(dp_pulse);
```

Where:

`uint8_t dp_pulse`: Pulse width (0 to disable, 1-255µs)

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

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

- Single EFR32 radio
- RHO unused
- 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 RX and TX (escalation disabled), PD12
  - Compatible 3-Wire W-Fi PTA devices sometimes refer to this signal as RF_STATUS or BT_STATUS (active high).

  Note: PRIORITY is static, not directional. If operated with a 3-Wire 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.
- PWM REQUEST disabled.
- Other options enabled to maximize 802.15.4 performance:
  - 802.15.4 RX and TX both at high priority
  - Receive retry REQUEST enabled with 16ms time-out and high priority.
  - Enabled ACKing when GRANT deasserted.

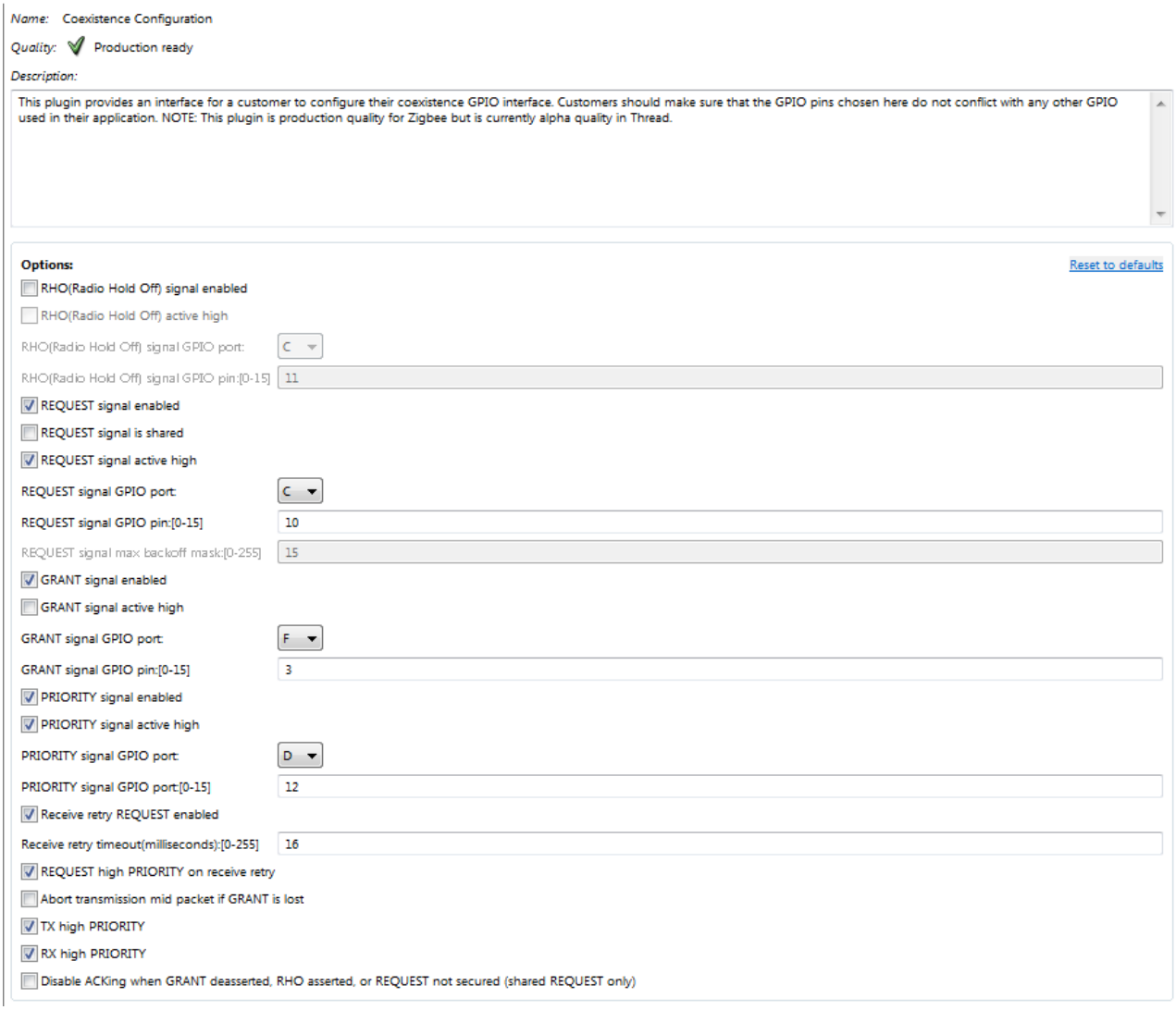


Figure 10. EFR32 PTA Support Configured to Operate as Single EFR32 with Typical 3-Wire W-Fi PTA

The logic analyzer capture in the following figure shows the PTA interface, Wi-Fi radio state, and EFR32 radio state for an EFR32 radio configured for typical 3-Wire Wi-Fi/PTA.

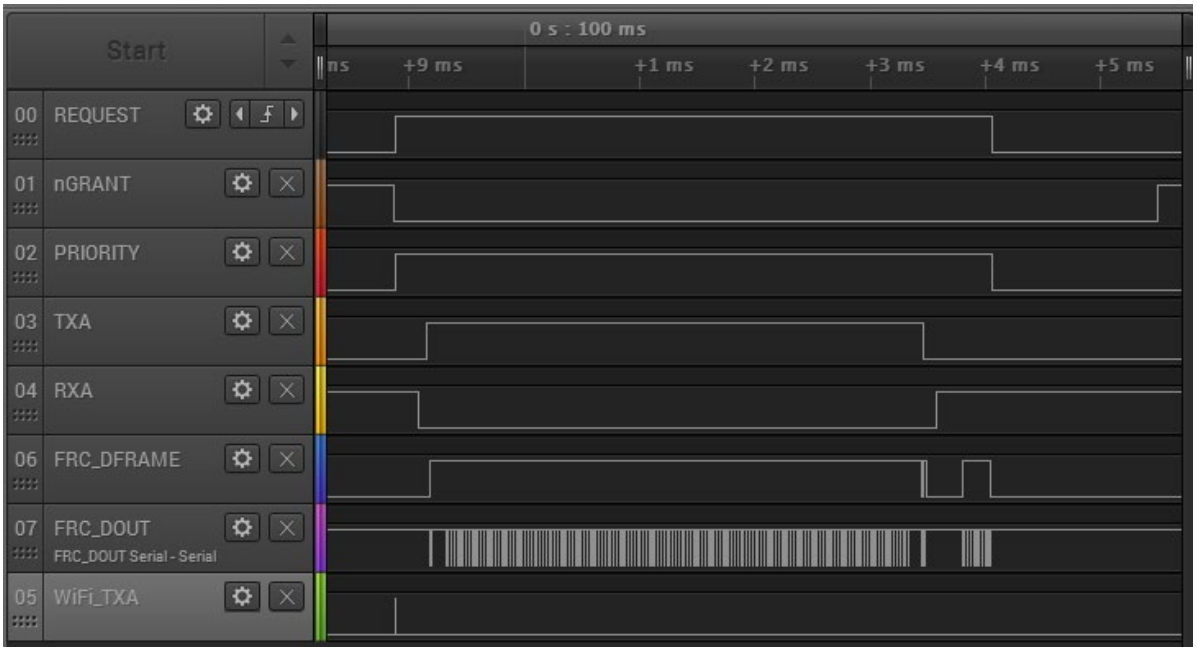


Figure 11. Example 802.15.4 TX for Single EFR32 typical 3-Wire Wi-Fi/PTA Logic Analyzer Capture

Where:

- **REQUEST**: active high, push-pull REQUEST output
- **nGRANT**: active low GRANT input
- **PRIORITY**: active high PRIORITY output
- **TXA**: EFR32 FEM TX Active control signal (configured via FEM Control plugin)
- **RXA**: EFR32 FEM RX Active control signal (configured via FEM Control plugin)
- **FRC_DFRAME**: EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **FRC_DOUT**: EFR32 Frame Control Data Out signal (packet trace data)
- **WiFi_TXA**: Wi-Fi TX Active signal

This logic analyzer sequence shows:

1. Wi-Fi starts a transmit, but is immediately pre-empted (WiFi_TXA pulse) by higher priority 802.15.4 transmit asserting REQUEST and PRIORITY.
2. GRANT is asserted by Wi-Fi/PTA.
3. EFR32 radio completes CCA and CCA passes and GRANT is asserted.
4. EFR32 radio proceeds with transmit (RXA deasserts, followed by TXA assert).
5. After transmit, EFR32 waits for ACK (TXA deasserts, followed by RXA assert).
6. EFR32 receives ACK (second FRC_DFRAME pulse). <= *802.15.4 TX message successfully completed*
7. EFR32 deasserts PRIORITY and REQUEST.
8. Wi-Fi/PTA deasserts GRANT.

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

- Multiple EFR32 radios (external 1 kΩ ±5% pull-up required on REQUEST)
- RHO unused
- REQUEST shared, active low, PC10
- GRANT, active low, PF3
- PRIORITY unused
- PWM REQUEST disabled
- Other options enabled to maximize 802.15.4 performance:
  - Receive retry REQUEST enabled with 16ms time-out.
  - Enabled ACKing when GRANT is deasserted.

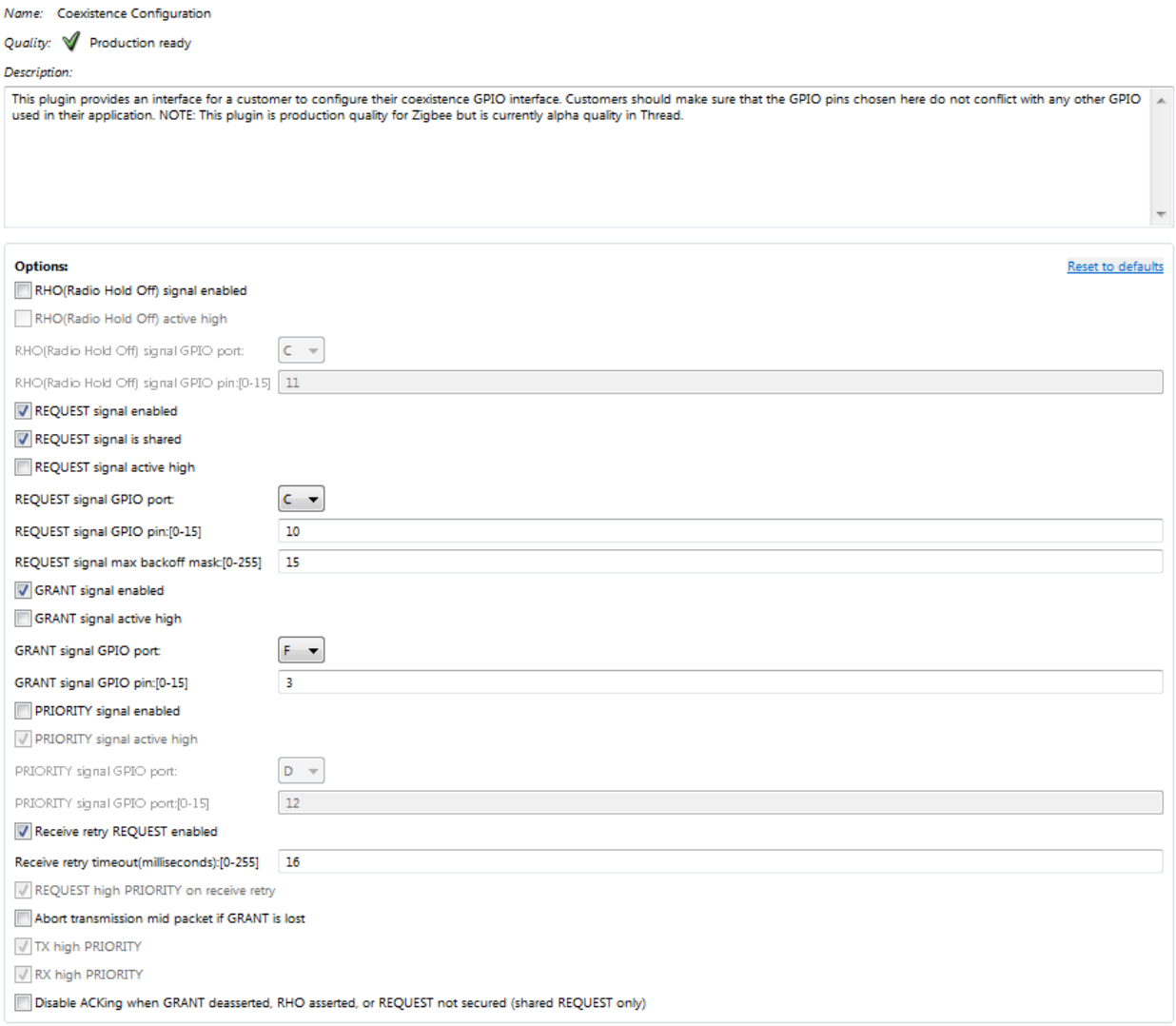


Figure 12. EFR32 PTA Support Configures to Operate with Multi-radio 2-Wire Wi-Fi/PTA with active-low REQUEST

The logic analyzer capture in the following figure 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:

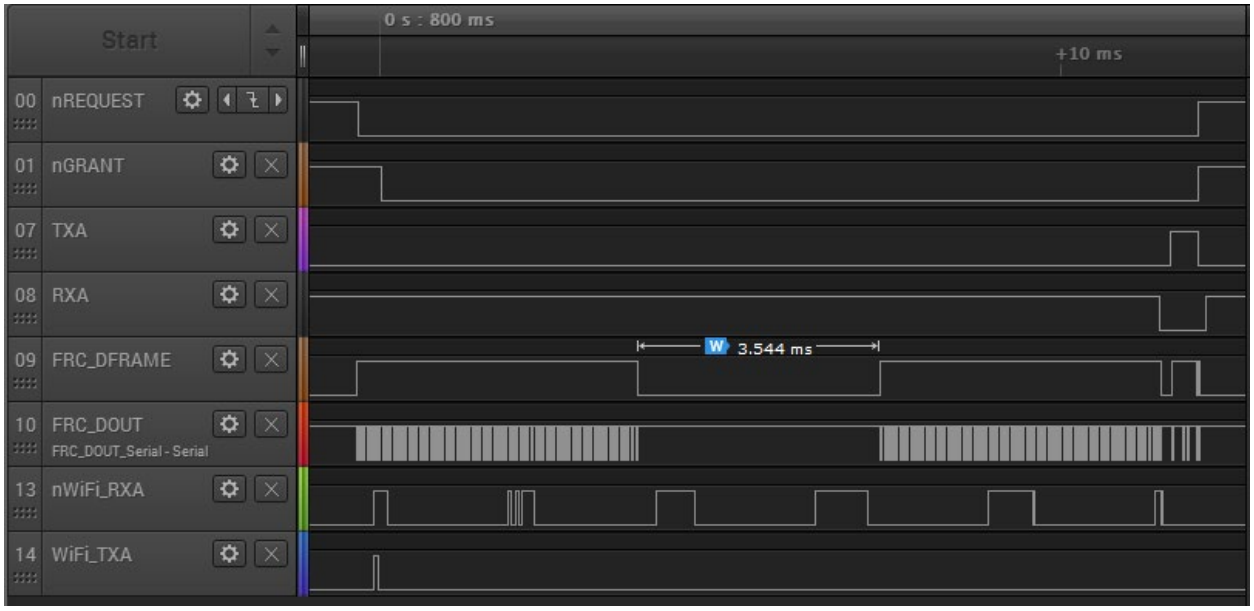


Figure 13. Example 802.15.4 RX for Multi-EFR32 2-Wire Wi-Fi/PTA with active-low REQUEST Logic Analyzer Capture

Where:

- **nREQUEST**: active low, shared (open-drain) REQUEST input/output
- **nGRANT**: active low GRANT input
- **TXA**: EFR32 FEM TX Active control signal (configured via FEM Control plugin)
- **RXA**: EFR32 FEM RX Active control signal (configured via FEM Control plugin)
- **FRC_DFRAME**: EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **FRC_DOUT**: EFR32 Frame Control Data Out signal (packet trace data)
- **nWiFi_RXA**: Wi-Fi RX Active signal
- **WiFi_TXA**: Wi-Fi TX Active signal

This logic analyzer sequence shows:

1. 802.15.4 packet is detected (FRC_DFRAME asserted) while Wi-Fi is receiving a packet (nWiFi_RXA asserted).
2. Shared REQUEST signal is tested and found not asserted by another EFR32 radio, so receiving EFR32 radio asserts REQUEST.
3. Wi-Fi ACK is transmitted (WiFi-TXA asserted) during 802.15.4 receive (no Wi-Fi TX pre-emption or higher priority Wi-Fi activity).
4. After Wi-Fi ACK completes, GRANT is asserted by Wi-Fi/PTA.
5. 802.15.4 receive is completed but CRC failed as packet was corrupted by co-located Wi-Fi ACK transmit during receive.
6. Since PTA configured with *Receive retry REQUEST enabled* using 16ms time-out, REQUEST is held up to 16ms for 802.15.4 retry with 2.4GHz quiet (Wi-Fi held off).
7. Wi-Fi continues to receive packets (nWiFi_RXA asserts), but does not ACK while 802.15.4 radio has GRANT.
8. After 3.5ms gap for end-node ACK time-out and MAC random delay, the 802.15.4 retry packet arrives and is received without error.
9. 802.15 ACK is transmitted (TXA asserted). <= *802.15.4 RX message successfully completed*
10. After 802.15.4 ACK completes, REQUEST is deasserted, followed by GRANT deassert.

Example 3: Configure EFR32 PTA support to operate with multi-radio typical 3-Wire Wi-Fi/PTA

- Multiple EFR32 radios (external 1 kΩ ±5% pull-down required on REQUEST and external 1 kΩ ±5% pull-down required on PRIORITY)
- RHO unused
- REQUEST shared, active high, PC10
- GRANT, active low, PF3
- PRIORITY shared (see Section 5.4 Multi-EFR32 Shared "PRIORITY" Signal), active high RX and TX (escalation disabled), PD12
- PWM REQUEST disabled
- Other options enabled to maximize 802.15.4 performance:
  - Receive retry REQUEST enabled with 16ms time-out.
  - Enabled ACKing when GRANT deasserted.

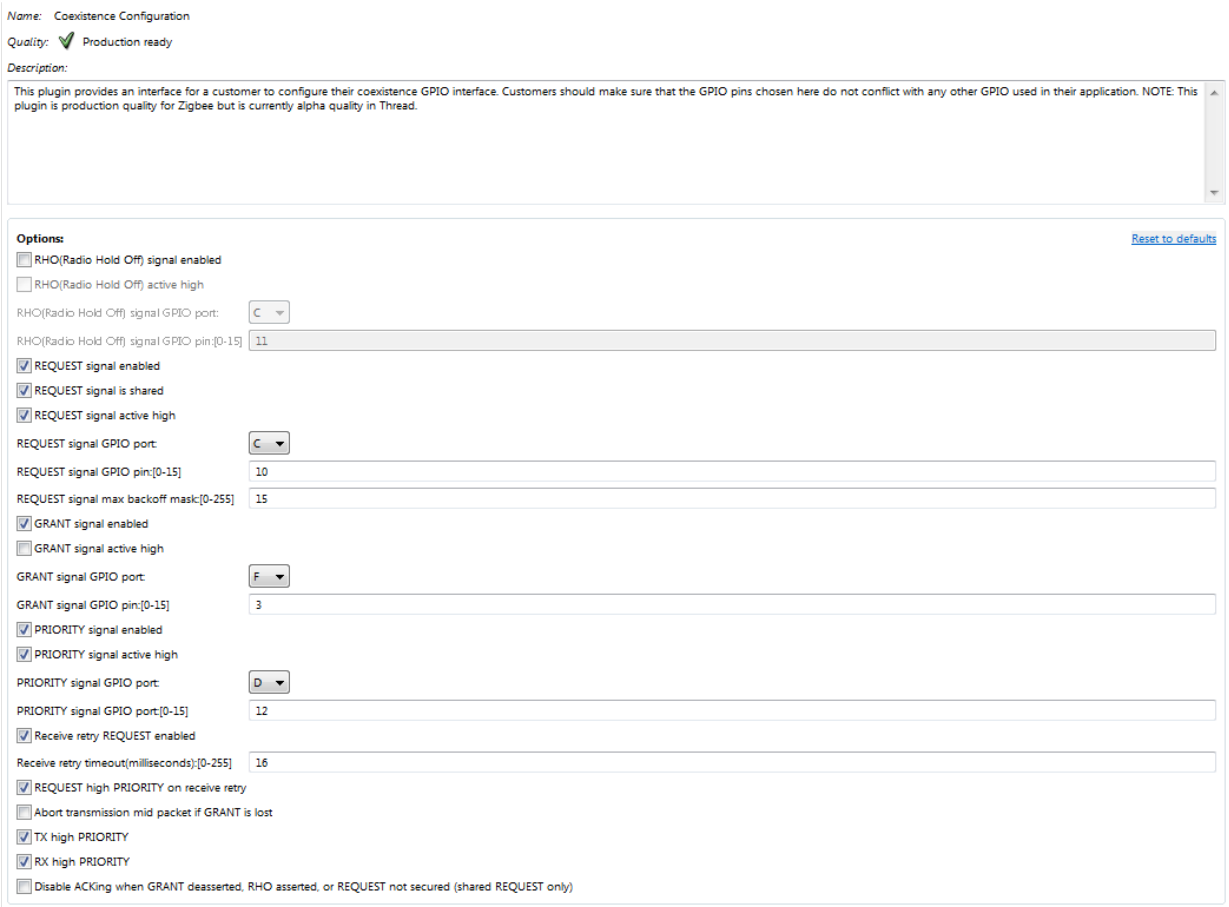


Figure 14. EFR32 PTA Support Configured to operate with Multi-radio typical 3-Wire Wi-Fi/PTA

5 Application Code Coexistence Extensions

This section describes application code available to extend the coexistence solution for particular applications. The application code files are available to download via: http://github.com/SiliconLabs/AN1017-extensions

5.1 Coexistence Custom CLI Console Commands

In both Host and SoC applications, custom CLI commands can be added to support run-time console control of PTA enabled/disabled, PTA run-time configuration (ptaOptions), and PTA debug counters. These custom CLI commands are useful in manual testing of various coexistence configurations, but also support run-time reconfiguration. To add such custom CLI commands to application, download and follow the instructions in ***pta-custom-cli.c***.

MAC and APS stack counters are documented in the stack API documentation. The six coexistence PTA-specific debug counters have the following meanings:

Counter Index	Meaning
EMBER_COUNTER_PTA_LO_PRI_REQUESTED	Occurrences of REQUEST asserted with low priority
EMBER_COUNTER_PTA_HI_PRI_REQUESTED	Occurrences of REQUEST asserted with high priority
EMBER_COUNTER_PTA_LO_PRI_DENIED	Occurrences of GRANT denied with low priority REQUEST
EMBER_COUNTER_PTA_HI_PRI_DENIED	Occurrences of GRANT denied with high priority REQUEST
EMBER_COUNTER_PTA_LO_PRI_TX_ABORTED	Occurrences of TX aborted by GRANT deasserted with low priority REQUEST
EMBER_COUNTER_PTA_HI_PRI_TX_ABORTED	Occurrences of TX aborted by GRANT deasserted with high priority REQUEST

Notes

1. In **EmberZNet PRO 6.3.0 and 6.3.1 and Silicon Labs Thread 2.7.0 and 2.7.1**, GRANT deasserts occurring in the initial stages or slightly before a TX event may not be detected and the TX event can continue unaborted. For **earlier or later EmberZNet PRO and Silicon Labs Thread 2.8.0 versions**, the TX abort is correctly detected and the TX event is halted.
2. In **EmberZNet PRO 6.0.0 and 6.1.0**, the MAC Retry counter (EMBER_COUNTER_MAC_TX_UNICAST_RETRY) always reports zero. In **EmberZNet PRO 5.10.1 or earlier and EmberZNet PRO 6.2.0 or later**, the MAC Retry counter correctly reports MAC retries.

5.2 Prevent Sleep/Idle in SoC and xNCP/SPI Applications

Note: For **EmberZNet PRO 6.1.0 or earlier and Silicon Labs Thread 2.5.0 or earlier**, SoC and xNCP/SPI applications require an event to disable sleep/idle. For **EmberZNet PRO 6.2.0 or later and Silicon Labs Thread 2.6.0 or later**, this application code adding an event to disable sleep/idle is not required.

As described in Section 4.1.1 Single EFR32 Connected to Wi-Fi/PTA and Section 4.1.2 Multiple EFR32s Connected to Wi-Fi/PTA, the "Receive Retry REQUEST" feature can produce REQUEST holds longer than the programmed REQUEST hold time-out for SoC and xNCP/SPI applications. These long REQUEST holds can be problematic for some Wi-Fi devices, but can be avoided by adding an always active event to the application.

To add an always active event to the application, download and follow the instructions in ***prevent-idle-sleep.c***.

5.3 TX PRIORITY Escalation

Notes

1. For **EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later,** TX PRIORITY Escalation is integrated within the SDK and this application sample code is not required.
2. For **EmberZNet PRO 6.3.0 and Silicon Labs Thread 2.7.0, TX PRIORITY Escalation** is always included in application, regardless of plugin hardware configuration, but can be disabled by setting both plugin hardware configuration thresholds to 0 and/or setting

both thresholds to 0 using run-time API. For **EmberZNet PRO 6.3.1 or later and Silicon Labs Thread 2.7.1** or later, TX PRIORITY Escalation can be excluded from application through plugin configuration.

To improve Wi-Fi performance with coexistence Zigbee/Thread, it is possible to start all Zigbee/Thread TX messages at low priority. However, to avoid blocking all Zigbee/Thread TX messages during busy Wi-Fi, TX PRIORITY Escalation will escalate TX to high priority after a programmable number of CCA/GRANT or MAC failures. Then, after a successful TX message, de-escalate TX back to low priority.

As described in Section 4.1.2 Multiple EFR32s Connected to Wi-Fi/PTA, TX PRIORITY Escalation can be controlled only at run-time via "CCA/GRANT TX PRIORITY Escalation Threshold" and "MAC Fail TX PRIORITY Escalation Threshold" fields of ptaOptions. When using this feature, "TX high PRIORITY" field must be set to 0 to avoid driving PRIORITY high on all TX messages.

However, in addition to setting the run-time programmable thresholds in ptaOptions, application code is required to support TX PRIORITY Escalation. To add **TX PRIORITY Escalation** to the application, download and follow the instructions in ***tx-priority-escalation.c***.

5.4 Multi-EFR32 Shared "PRIORITY" Signal

Note: In EmberZNet PRO 6.1.0 or later and Silicon Labs Thread 2.5.0 or later, Shared PRIORITY is implemented in the stack and can be enabled through Hardware Configurator. The following application code to implement Multi-EFR32 Shared "PRIORITY" Signal is for EmberZNet PRO 5.10.1 or Silicon Labs Thread 2.3.1 or earlier.

For multi-EFR32 radio applications requiring a shared PRIORITY, it is necessary to change the PRIORITY GPIO pin from default push-pull mode to either open-drain or open-source mode. To add **Multi-EFR32 Shared "PRIORITY" Signal** to the application (EmberZNet PRO 5.10.1 or earlier and Thread 2.3.1 or earlier), download and follow the instructions in ***multi-efr32-shared-priority.c***.

5.5 PWM for High Duty Cycle Wi-Fi

Note: For **EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later**, **PWM for High Duty Cycle Wi-Fi** is integrated within the SDK and this application sample code is not required.

As noted earlier, Zigbee and Thread have no knowledge of when an incoming packet will arrive and must capture the 802.15.4 PREAMBLE/SYNCH (160 us duration) to detect an incoming packet. As co-located Wi-Fi TX RF duty cycle increases, the Wi-Fi TX idle periods become smaller and the number of idle periods exceeding 160µs becomes fewer. As such, the probability of a 160µs PREAMBLE/SYNCH arriving during an acceptably large Wi-Fi TX idle period decreases significantly.

This PWM extension feature implements regular PTA REQUESTs with high PRIORITY to interrupt the high Wi-Fi RF duty cycle and ensure adequate windows for PREAMBLE/SYNCH detection. However, this feature does degrade Wi-Fi performance and exact PWM period and duty cycle must be carefully selected to avoid collapsing the Wi-Fi network (e.g., do not alias PWM period with Wi-Fi beacons).

Note: Ensure aggregation is enabled on Wi-Fi/PTA device. During suitable conditions, aggregation combines multiple smaller packets into fewer larger packets, reducing total packet overhead. As such, aggregation aids Zigbee and Thread packet detection as Wi-Fi TX idle periods combine into larger idle periods for improved PREAMBLE/SYNCH detection.

5.5.1 Background

The following 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 15. High-Duty Cycle Wi-Fi TX Active

From the first group, only the 0.306ms idle period is sufficient for PREAMBLE/SYNCH detection 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 3. 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 PREAMBLE/SYNCH detection [0.428 / 15.485 x 100%], but, again, this calculation does not include Wi-Fi RX Activity. With only 2.8% probability of detection and a target 1% or less message loss, 165 retries, per 802.15.4 message, are required [CEILING(LOG(1%)/LOG(100%–2.8%)]. This large number of required retries is far beyond the MAC, NWK, and APS retries and, even if retries are extended, the message latency becomes impractical. Large retries also quickly degrade life-time for battery power end-nodes.

To resolve, 802.15.4 radios need more RF air-time 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 802.15.4 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 and this PWM extension feature can be applied to any use case where EFR32 incoming Zigbee and Thread packet detection 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
- Zigbee and Thread co-channel with Wi-Fi
- External T/R switch requiring Zigbee and Thread to share a single antenna with Wi-Fi

5.5.2 PWM Extension Feature Description

In most EFR32 applications, particularly xNCP applications, many EFR32 resources are unused and available for feature implementation. This PWM extension feature requires the following:

- Two available EFR32 GPIOs
- One available EFR32 internal TIMER
- Three consecutive and available EFR32 Peripheral Reflex System (PRS) channels with output mappable to target REQUEST||PWM GPIO
- Two consecutive and available EFR32 Peripheral Reflex System (PRS) channels with output mappable to target PRIORITY||PWM GPIO

In the following figure, a BRD4151A EFR32 reference radio module drives REQUEST and PRIORITY to the recommended QFN48 GPIO defaults, PC10 and PD12, respectively. PD13 and PD14 are two available GPIOs and TIMER1 is also available. PRS channels 9, 10, and 11 are available and selected for REQUEST||PWM as PRS[11] output can drive PC10. PRS channels 3 and 4 are also available and selected for PRIORITY||PWM as PRS[4] output can drive PD12.

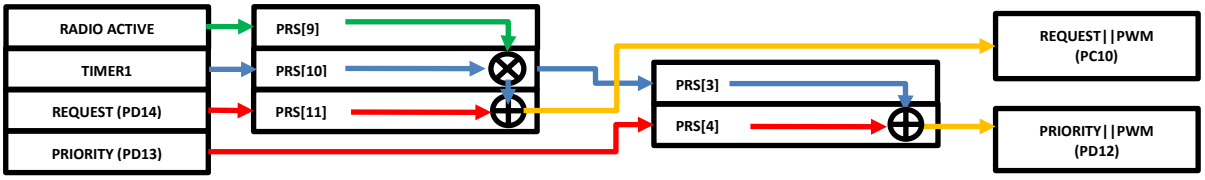


Figure 16. PWM Extension Block Diagram

With the above configuration, the Coexistence-Configuration plugin operates as before, except REQUEST and PRIORITY are mapped to drive PD13 and PD14. TIMER1 is programmed with a PWM waveform (that is, 19.5ms period and 20% duty cycle), which, using PRS channels 10 & 11 and 3 & 4 are OR'ed with REQUEST and PRIORITY to generate REQUEST||PWM and PRIORITY||PWM at PC10 and PD12, respectively.

Under the same Wi-Fi TX Active throughput conditions as in Section 5.5.1 Background, the following figure shows the REQUEST||PWM (shown as REQUEST||TIMER1) and PRIORITY||PWM (shown as PRI||TIMER1) signals interrupting the Wi-Fi TX sufficiently enough to allow 802.15.4 RX with < 1% message loss.

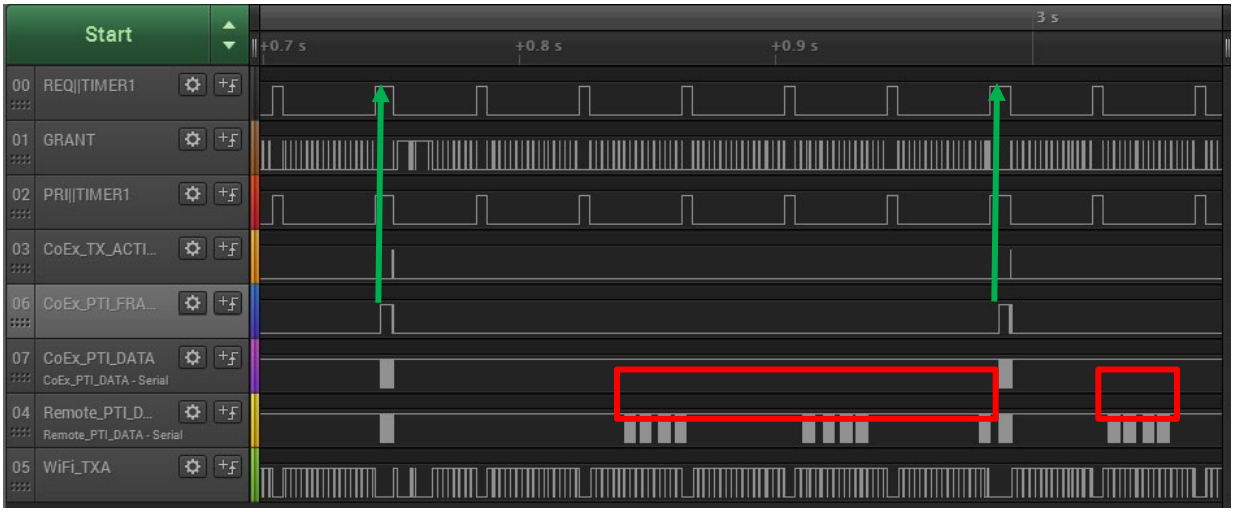


Figure 17. EFR32 RX Success with PWM during High-Duty Cycle Wi-Fi TX

The green arrows show cases where the PREAMBLE/SYNCH was detected during a high PWM cycle, driving both REQUEST||PWM and PRIORITY||PWM high. This high PWM cycle stopped Wi-Fi TX Activity to allow detection of incoming 802.15.4 PRERAMBLE/SYNCH. As also seen, those two cycles are longer than all other cycles due to assertion of normal PTA REQUEST and PRIORITY signals OR'ed with PWM cycle.

The red boxes are cases where the remote 802.15.4 radio executed packet transmits that were not detectable due to Wi-Fi TX Activity and packets' PREAMBLE/SYNCH not aligning with the PWM cycle.

Optimum PWM period and duty cycle can vary based on 802.15.4 operation and Wi-Fi activity. If Wi-Fi activity is low, PWM could be disabled and fall back to normal PTA activity as described earlier. During very high Wi-Fi duty cycle for unicast 802.15.4 traffic, a PWM programmed to 19.5ms period and 20% duty is effective in reducing message loss to less than 1% with APS retries disabled. As noted, Wi-Fi throughput is impacted and, under 19.5ms/20% condition, the high duty cycle Wi-Fi TX throughput noted above dropped ~20% (~270Mbps without PWM to ~220Mbps with PWM).

During very high Wi-Fi duty cycle for broadcast 802.15.4 traffic without ACKs (for example, as occurs during network join sequence), a higher PWM duty cycle is required (e.g., 19.5ms period and 80% duty cycle). This higher duty cycle can be disruptive to Wi-Fi, but is only needed during expected broadcast message activity. This higher duty cycle can be reduced as soon as broadcast activity (for example, device join) completes.

5.5.3 PWM Implementation

The optimum PWM implementation varies with single-EFR32 vs. multi-EFR32 applications. This 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.

5.5.3.1 Single-EFR32 Radio

Application code to add PWM extension to single-EFR32 radio PTA REQUEST and PRIORITY coexistence signals, including custom EZSP or custom CLI commands for PWM run-time control (enable/disable, period, duty cycle, and priority), is available, by contacting Silicon Labs' Factory Applications and requesting access to ***pta-pwm.c***.

5.5.3.2 Multi-EFR32 Radio

For multi-EFR32 radio applications, 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 is applied to the shared REQUEST pin, other EFR32 radios interpret the REQUEST as busy when it actually may just be forcing Wi-Fi TX idle to allow any of the IoT radios to detect an incoming packet. The shared REQUEST and PWM extension features can both be accommodated by separating the shared REQUEST and, if enabled, shared PRIORITY signals from the REQUEST||PWM and PRIORITY||PWM signals.

In the example below, the PTA coexistence configuration for Zigbee, Thread, and BLE radio can be the same as the multi-EFR32 radio with typical 3-Wire PTA example shown in the following figure and in example #3 of Section 4.3 Coexistence Configuration Setup Examples for Different Wi-Fi/PTA Applications. While only one radio needs to implement the PWM and OR the PWM signal with REQUEST and PRIORITY, it may be necessary to have a second radio also implement the PWM and OR the PWM signal with REQUEST and PRIORITY, but with signals tristated. If the radio driving REQUEST||PWM and PRIORITY||PWM needs to go off-line (e.g., a firmware update), its PWM outputs can be tristated and the second radio's PWM outputs can then be activated for continued operation. In the example below, the Zigbee radio implements the PWM feature, the Thread radio provides PWM backup, and BLE radios implements multi-EFR32 coexistence exactly as before.

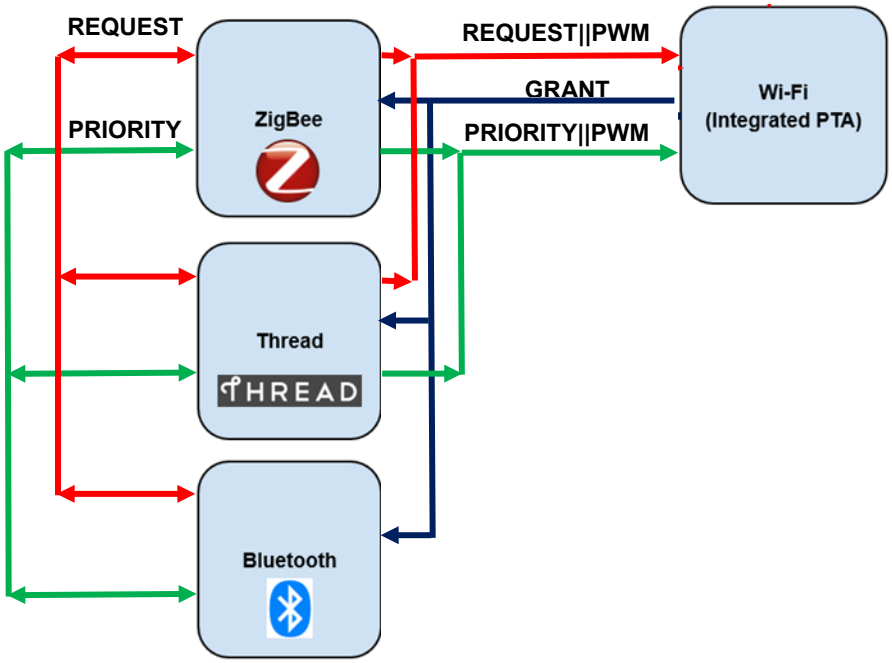


Figure 18. Three EFR32s Supporting PWM and Wi-Fi/PTA with Single 3-Wire PTA Interface
(REQUEST and PRIORITY pull-ups or pull-downs not shown)

With shared REQUEST and shared PRIORITY set to open-drain/open-source, those GPIOs drive the PRS channel inputs with wire-OR/wire-AND signals to which the PWM signal can be OR'ed. As such, each EFR32 will arbitrate without PWM interaction and the resulting REQUEST||PWM and PRIORITY||PWM drives the Wi-Fi/PTA as if a single external radio.

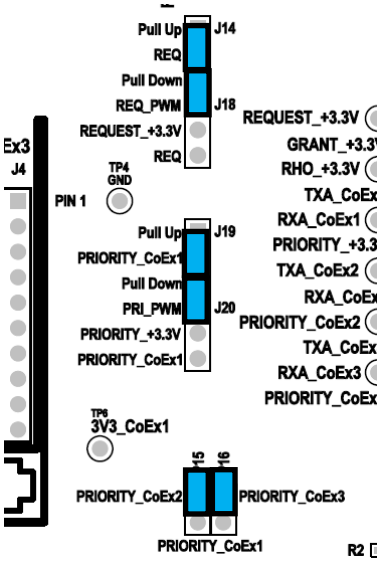
The PWM implementation need only be executed on the EFR32 directly driving the Wi-Fi/PTA device and, except for possible different GPIO and PRS channel selection, that implementation is identical to the single-EFR32 radio.

The multi-EFR32 radio case with PWM extension feature can be evaluated using the SLWSTK-COEXBP EVB by:

1. Implementing the multi-EFR32 coexistence-configuration plugin as before on CoEx1, CoEx2, and CoEx3 (if needed).
2. Implementing the PWM feature on the EFR32 radio installed in CoEx1 position:
   - Drive REQUEST||PWM to PC6, which can be driven by the following PRS channels/locations: CH0/LOC8, **CH9/LOC11**, CH10/LOC0, and CH11/LOC5.
   - Drive PRIORITY||PWM to PC7, which can be driven by the following PRS channels/locations: CH0/LOC9, CH9/LOC12, CH10/LOC1, and **CH11/LOC0**.

   Note: On BRD4151A (EFR32MG1) or BRD4168A (EFR32MG13), PC6 and PC7 are used to evaluate the PWM extension feature on SLWSTK-COEXBP EVB due to available GPIO pins on the WSTK EXP header and the xNCP/UART support provided by PA0-PA3 connected to optional CP2102N-MINIEKs. PC6 and PC7 typically implement SPI MOSI and SPI MISO, respectively, and different GPIO choices are likely desired on end-application EVB.

   On BRD4161A (EFR32MG12), PA6 and PA7 drive these same signals. However, PA6 and PA7 cannot be driven by PRS channels. As such, while the PWM extension is supported on EFR32MG12, the PWM extension is more easily evaluated on Co-existence Backplane EVB using BRD4151A or BRD4168A.
3. Change SLWSTK-COEXBP J18 and J20 jumpers to select REQ_PWM and PRI_PWM.
   - For multi-EFR32 radio implementing PWM using typical 3-Wire PTA, REQUEST (active-high), PRIORITY (active-high), and GRANT (active-low), the REQUEST and PRIORITY jumpers are:



4. On CoEx1 radio only, add the same code as in single-EFR32 case, except for #defines selecting and configuring the REQUEST, PRIORITY, REQUEST||PWM, and PRIORITY||PWM GPIOs.
5. Application code to add PWM extension to multi-EFR32 radio PTA REQUEST and PRIORITY coexistence signals, including custom EZSP or custom CLI commands for PWM run-time control (enable/disable, period, duty cycle, priority, and tristate), is available, by contacting Silicon Labs' Factory Applications and requesting access to ***pta-pwm.c***.

5.5.4 PWM Run-Time Control

Note: For EmberZNet PRO 6.3.0 or later and Silicon Labs Thread 2.7.0 or later, **PWM for High Duty Cycle Wi-Fi** is integrated within the SDK and this application sample code is not required.

As noted above, the optimum PWM configuration can vary with Wi-Fi activity and 802.15.4 activity and an API is needed to allow the host application to control the PWM period, duty cycle, and priority. To implement PWM run-time control (enable/disable, period, duty cycle, priority, and tristate), contact Silicon Labs' Factory Applications and request access to ***pta-pwm.c***.

6 Coexistence Backplane Evaluation Board (EVB)

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

7 Single-EFR32 Coexistence Test Results with a typical 3-Wire Wi-Fi/PTA

Additional details on the test results are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

7.1 Measured Result Observations

Additional details on the test results are available in an expanded version of this application note, *AN1017-NDA: ZigBee and Thread Coexistence with Wi-Fi*, available under non-disclosure from Silicon Labs technical support.

8 Conclusions

Co-located, strong Wi-Fi can have a substantial impact on 802.15.4 performance. 802.15.4 performance with co-located Wi-Fi can be improved through unmanaged and managed coexistence techniques. Unmanaged coexistence recommendations include:

1. Implement frequency separation.
2. Operate Wi-Fi with 20.MHz bandwidth.
3. Increase antenna isolation.
4. Use Zigbee/Thread 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 802.15.4 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. Managed coexistence utilizes:

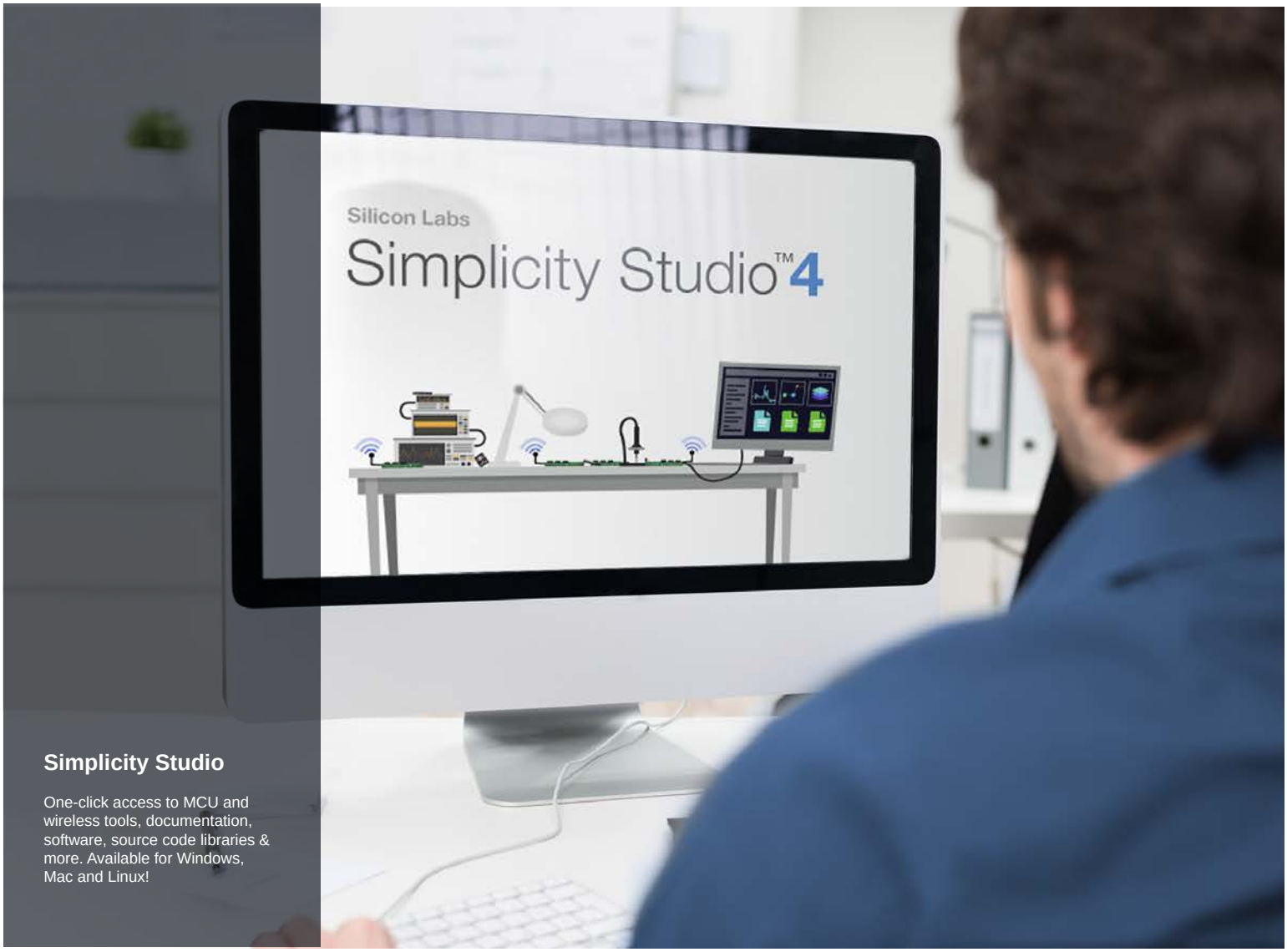
1. Wi-Fi/PTA devices providing 802.15.2-derived Packet Traffic Arbitration.
2. Silicon Labs' EFR32 PTA solution:
    1. One to four GPIOs implementing a combination of REQUEST, GRANT, PRIORITY, and RHO (two additional GPIOs are required to implement *PWM with High-Duty Cycle Wi-Fi* feature)
    2. Silicon Labs' AppBuilder coexistence-configuration plugin to configure EFR32 PTA support for available GPIO pins and for compatibility with the chosen Wi-Fi/PTA device.
    3. Silicon Labs' API, supporting run-time PTA reconfiguration.

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

1. Improved device join success:
    1. However, device join utilizes broadcast messages, which are not retried.
    2. *If possible, device join success can be further improved by temporarily reducing Wi-Fi traffic during devices joining 802.15.4 network.*
2. Substantially reduced MAC retries:
    1. Reduces message latency.
    2. Improves end-node battery life.
    3. Frequency separation remains important, as best managed coexistence performance is for "far-away" channels.

Substantially reduced message failure:

1. 802.15.4 network remains operational, even during high Wi-Fi duty cycles.



Simplicity Studio

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

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com