AN1128: *Bluetooth*® Coexistence with Wi-Fi®

This application note describes methods to improve coexistence of 2.4 GHz IEEE 802.11b/g/n Wi-Fi and Bluetooth® radios. These techniques are applicable to the EFR32MGx family and EFR32BGx family. This application note assumes you have a basic understanding of how Wi-Fi coexistence is implemented on EFR32 devices. For more information, see *UG103.17: Wi-Fi® Coexistence Fundamentals.*

This application note describes EFR32 Bluetooth coexistence support for Silicon Labs Bluetooth SDK version 3.0.1.0 and Bluetooth Mesh SDK version 1.7.2.0. See 5 Document Revision History for a summary of key changes in previous revisions of this application note.

Additional details about the implementation of managed coexistence are included in *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi,* available under non-disclosure from Silicon Labs Sales.

KEY POINTS

- Configure PTA support for Bluetooth.
- Use application code existence extensions.
- Order the Coexistence Backplane Evaluation Board.

Contents

1. Introduction **2**

2. PTA Support Software Setup **3**

2.1. Compile Time PTA Setup and Defaults 3

2.2. Run-Time PTA Re-configuration 11

2.3. Run-Time PTA Debug Counters 12

2.4. Coexistence Configuration Setup Examples for Different Wi-Fi/PTA Applications 13

3. Application Code Coexistence Extensions **18**

3.1. Example TX_ACTIVE/RX_ACTIVE 18

4. Coexistence Backplane Evaluation Board (EVB) **19**

5. Document Revision History **20**

1. Introduction

This application note includes the following sections:

- 2 PTA Support Software Setup describes how to configure the Silicon Labs Packet Traffic Arbitration (PTA) for Bluetooth.
- 3 Application Code Coexistence Extensions describes how to use the application code existence extensions.
- 4 Coexistence Backplane Evaluation Board (EVB) explains how to order the EVB for evaluating the Silicon Labs EFR32 software coexistence solution.

Notes:

1. Not all coexistence support features are present in SDK versions earlier than Bluetooth 3.0.1.0 and Bluetooth Mesh 1.7.2.0. Users of Bluetooth SDK 2.13.7 or earlier and Bluetooth Mesh SDK 1.7.1 or earlier may see different features from those documented in this application note.
2. Throughput this application note “Bluetooth Low Energy” is referenced as “Bluetooth”.
3. This application note addresses Bluetooth coexistence applications using EFR32 devices as per Bluetooth Core Specification v5.0 Vol 6 “Low Energy Controller” (point-to-point) and as per Bluetooth Specification Mesh Profile v1.0 (mesh network). These two applications have different coexistence considerations and, where necessary, this application note differentiates using the following terms:
    - “Bluetooth device” to reference Bluetooth Core Specification v5.0 Vol 6 “Low Energy Controller” (point-to-point) operation
    - “Bluetooth mesh device” or “Bluetooth mesh node” to reference Bluetooth Specification Mesh Profile v1.0 (mesh network) operation

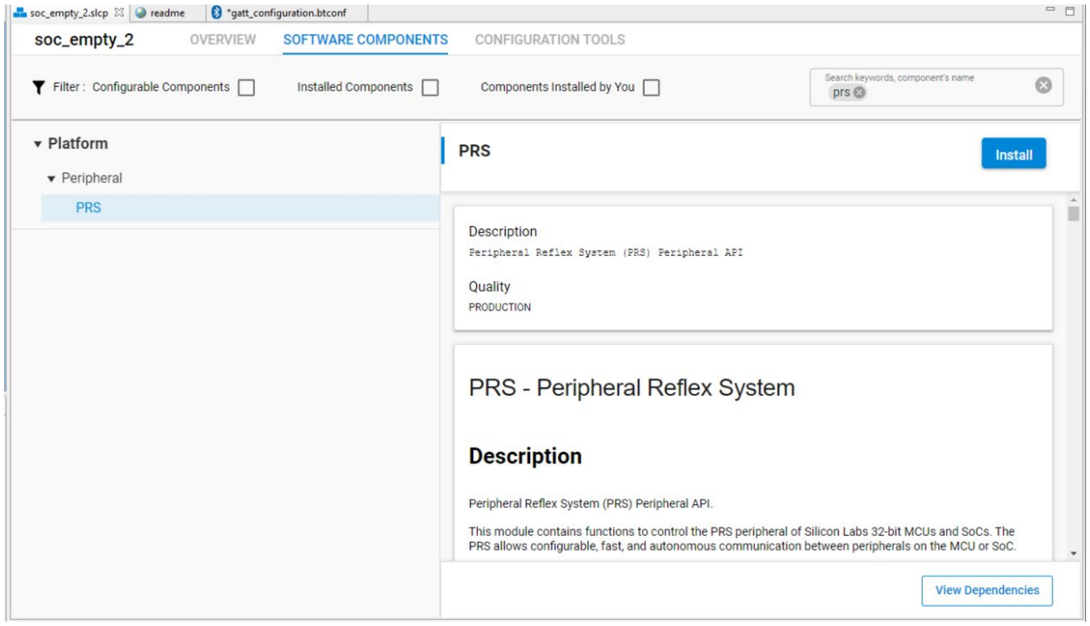
2. PTA Support Software Setup

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

2.1. Compile Time PTA Setup and Defaults

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

1. Create Bluetooth or Bluetooth Mesh project in Simplicity Studio.
2. Add the PRS component to your project. Double-click the slcp file of your project to open the Project Configurator, and go to the SOFTWARE COMPONENTS tab. Search for PRS, select it, and click **Install.**



PRS

Description
Peripheral Reflex System (PRS) Peripheral API

Quality
PRODUCTION

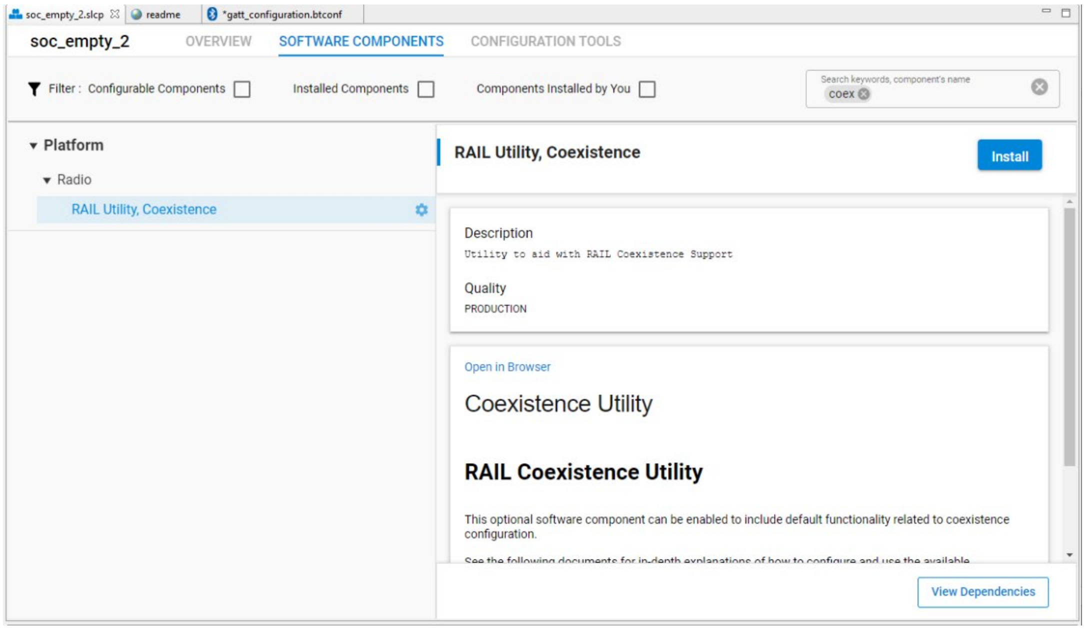
PRS - Peripheral Reflex System

Description

Peripheral Reflex System (PRS) Peripheral API.

This module contains functions to control the PRS peripheral of Silicon Labs 32-bit MCUs and SoCs. The PRS allows configurable, fast, and autonomous communication between peripherals on the MCU or SoC.

3. Add the RAIL Utility, Co-existence component to your project. Search for COEX, select **RAIL Utility. Co-existence**, and click **Install**.



RAIL Utility, Coexistence

Description
Utility to aid with RAIL Coexistence Support

Quality
PRODUCTION

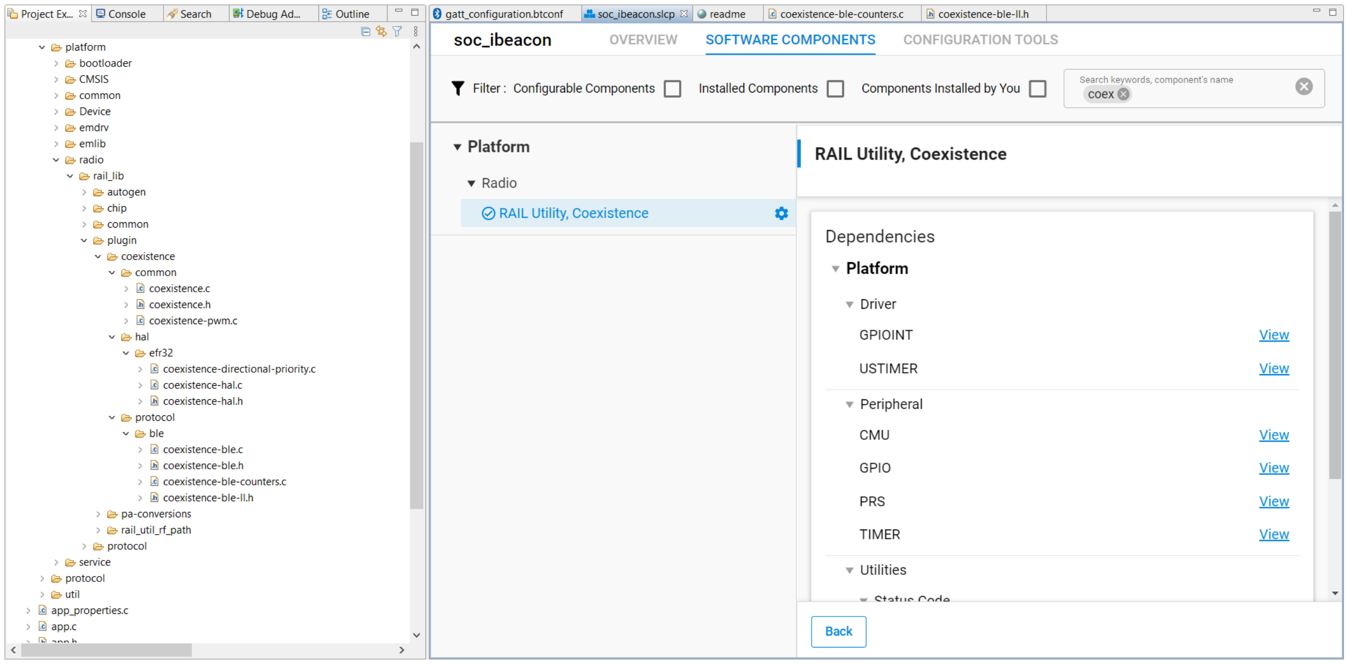
Coexistence Utility

RAIL Coexistence Utility

This optional software component can be enabled to include default functionality related to coexistence configuration.

See the following documents for in-depth explanations of how to configure and use the available

The coexistence directory under gecko_sdk_3.x/platform/radio/rail_lib/plugin gets populated with the coexistence sources:



```
platform
  bootloader
  CMSIS
  common
  Device
  emdrv
  emlib
  radio
    rail_lib
      autogen
      chip
      common
      plugin
        coexistence
          common
            coexistence.c
            coexistence.h
            coexistence-pwm.c
          hal
            efr32
              coexistence-directional-priority.c
              coexistence-hal.c
              coexistence-hal.h
          protocol
            ble
              coexistence-ble.c
              coexistence-ble.h
              coexistence-ble-counters.c
              coexistence-ble-ll.h
        pa-conversions
        rail_util_rf_path
      protocol
  service
protocol
util
app_properties.c
app.c
app.h
```

soc_ibeacon — OVERVIEW | SOFTWARE COMPONENTS | CONFIGURATION TOOLS

Filter : Configurable Components ☐ Installed Components ☐ Components Installed by You ☐ — Search keywords, component's name: coex

Platform
- Radio
  - RAIL Utility, Coexistence

RAIL Utility, Coexistence

Dependencies
- Platform
  - Driver
    - GPIOINT View
    - USTIMER View
  - Peripheral
    - CMU View
    - GPIO View
    - PRS View
    - TIMER View
  - Utilities
    - Status Code

Back

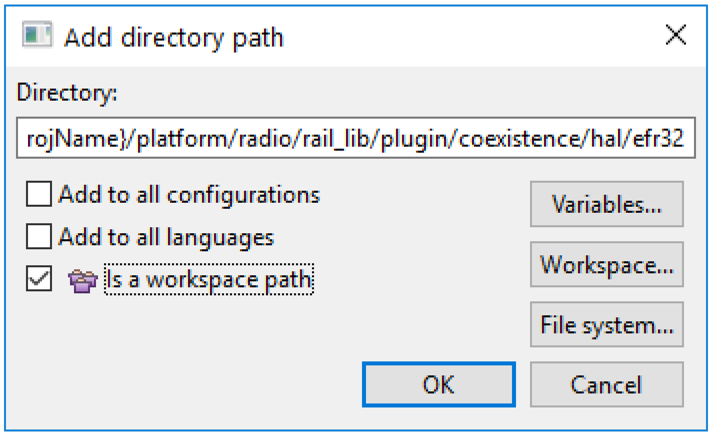
4. Edit project includes to include additional paths to files added to project:
   1. Right-click the project and select **Properties**.
   2. Expand **C/C++ General** and select **Paths and Symbols**.
   3. Select the **Includes** tab.
   4. In languages, select the desired C compiler (for example, **GNU C**).
   5. Press **add**, check **Is a workspace path**, and enter each of the following paths:

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

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

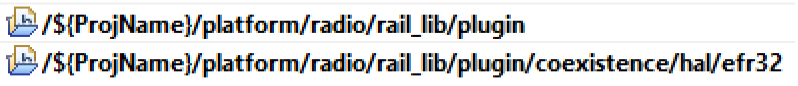
      /${ProjName}/platform/radio/rail_lib/plugin/coexistence/protocol/ble

      For example:



Directory: rojName}/platform/radio/rail_lib/plugin/coexistence/hal/efr32

☐ Add to all configurations
☐ Add to all languages
☑ Is a workspace path



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

   6. Click **OK** and **Yes** to rebuild index now
5. Add the following #defines to coexistence-hal-config.h:

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

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

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

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

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

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

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

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

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

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

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

#define BSP_COEX_DP_CHANNEL                                 (3)
```

6. Enable and edit new #defines to enable/configure PTA coexistence.
   - Enable the PTA feature.

     The following #define is required:

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

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

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

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

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

   - REQUEST Window

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

     The following #define example sets the REQUEST Window to 500µs:

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

   - REQUEST signal is shared.

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

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

     The following #define example enables Shared REQUEST.

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

   - REQUEST signal max backoff mask

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

     The following #define sets backoff to recommended value:

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

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

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

     ```
     #define BSP_COEX_GNT_PIN                        (3)
     ```

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

Notes:

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

- Abort transmission mid packet if GRANT is lost.

  If enabled, losing GRANT (or RHO asserted) during a Bluetooth TX will abort the Bluetooth TX. If not enabled, losing GRANT (or RHO asserted) after the start of a Bluetooth TX will not abort the Bluetooth TX.

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

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

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

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

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

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

```
#define BSP_COEX_PRI_PIN                   (8)
#define BSP_COEX_PRI_PORT                  (gpioPortD)
#define BSP_COEX_PRI_ASSERT_LEVEL          (1)
```

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

- PRIORITY Assert Enable

  The following #define example defaults PRIORITY to always deasserted:

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

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

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

- PRIORITY signal is shared

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

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

  The following #define example enables Shared PRIORITY.

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

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

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

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

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

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

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

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

  The following #defines example disables PWM at reset:

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

  The following #defines example enables PWM at reset:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The following #defines example selects the base PRS channel, REQUEST invert PRS channel, and RACPAEN invert channel used to create Directional PRIORITY. By default RAIL reserves PRS channel 7 for clock synchronization, but this PRS channel reservation can be configured through the `RAILCb_ConfigSleepTimerSync()` callback API.

```
#define BSP_COEX_DP_CHANNEL                   (3)
#define BSP_COEX_DP_REQUEST_INV_CHANNEL       (4)
#define BSP_COEX_DP_RACPAEN_INV_CHANNEL     (8)
```

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

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

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

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

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

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

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

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

     or

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

which is based on the following definition:

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

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

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

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

or

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

which is based on following definition:

```
typedef struct {
       uint8_t scan_min;
       uint8_t scan_max;
       uint8_t adv_min;
       uint8_t adv_max;
       uint8_t conn_min;
       uint8_t conn_max;
       uint8_t init_min;
       uint8_t init_max;
       uint8_t rail_mapping_offset;
       uint8_t rail_mapping_range;
       uint8_t afh_scan_interval;
       uint8_t adv_step;
       uint8_t scan_step;

}sl_bt_bluetooth_ll_priorities;
//Default priority configuration
#define SL_BT_BLUETOOTH_PRIORITIES_DEFAULT { 191, 143, 175, 127, 135, 0, 55, 15, 16, 16, 0,
4, 4 }
```

- Enable or disable Passive SCAN.

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

or

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

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

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

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

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

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

// Initialize coexistence
sl_bt_init_coex_hal();

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

2.2. Run-Time PTA Re-configuration

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

1. Disable/Enable the PTA feature.

   At runtime, the following code disables the PTA feature:

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

   At runtime, the following code enables the PTA feature:

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

2. REQUEST Window

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

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

   Where `desired_request_window` is the REQUEST_WINDOW in µs.

3. Abort transmission mid packet if GRANT is lost.

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

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

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

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

4. PRIORITY Escalation capability

   At runtime, the following code disables PRIORITY assertion:

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

   At runtime, the following code enables PRIORITY assertion:

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

5. Channel Map Masking

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

If EFR32 becomes the connection master, the Bluetooth channel map can be specified using this function call:

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

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

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

0: Channel *n* is bad.

1: Channel *n* is unknown.

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

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

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

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

7. Link layer Priority table.

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

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

where `data` is an array containing:

```
typedef struct {
  uint8_t scan_min;
  uint8_t scan_max;
  uint8_t adv_min;
  uint8_t adv_max;
  uint8_t conn_min;
  uint8_t conn_max;
  uint8_t init_min;
  uint8_t init_max;
  uint8_t rail_mapping_offset;
  uint8_t rail_mapping_range;
  uint8_t afh_scan_interval;
  uint8_t adv_step;
  uint8_t scan_step;
} sl_bt_bluetooth_ll_priorities;
```

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

2.3. Run-Time PTA Debug Counters

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

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

where:

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

  where, since startup or last reset:

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

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

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

- Single EFR32 radio
- REQUEST unshared, active high, PC10
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as RF_ACTIVE or BT_ACTIVE (active high)
- GRANT, active low, PF3
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as WLAN_DENY (deny is active high, making grant active low)
- PRIORITY, active high, PD12
  - Compatible 3-Wire Wi-Fi/PTA devices sometimes refer to this signal as RF_STATUS or BT_STATUS (active high)
  - PRIORITY is static, not directional. If operated with a 3-Wire Wi-Fi/PTA expecting directional:
    - Static high PRIORITY is interpreted as high PRIORITY and always in TX mode, regardless of actual TX or RX
    - Static low PRIORITY is interpreted as low PRIORITY and always in RX mode, regardless of actual TX or RX
- REQUEST_WINDOW is 50 µs
- Disabled Abort transmission mid packet if GRANT is lost
- PRIORITY is always high
- RHO unused

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

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

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

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

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

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

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

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

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

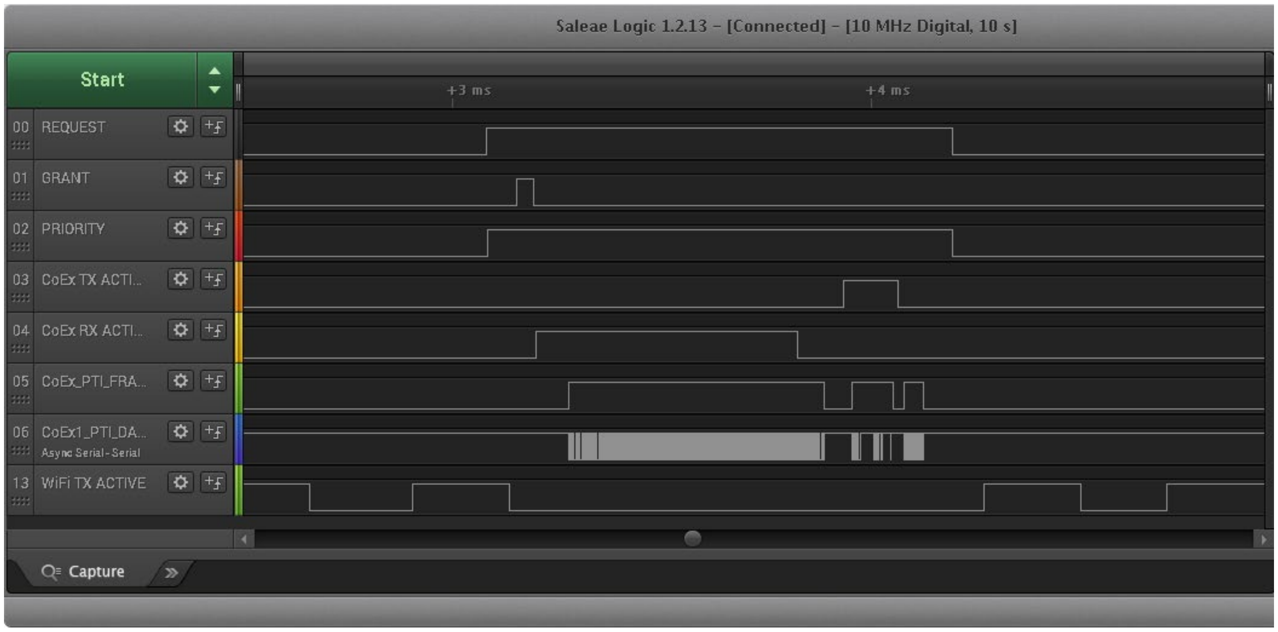


Figure 2-1. Example CONNECTION event (slave) for Single EFR32 typical 3-Wire Wi-Fi/PTA Logic Analyzer Capture

where:

- **REQUEST:** active high, push-pull REQUEST output
- **nGRANT:** active low GRANT input
- **PRIORITY:** active high PRIORITY output
- **CoEx TX ACTIVE:** EFR32 TX Active control signal (configured via sample code in section 3.1 Example TX_ACTIVE/RX_ACTIVE)
- **CoEx RX ACTIVE:** EFR32 RX Active control signal (configured via sample code in section 3.1 Example TX_ACTIVE/RX_ACTIVE)
- **CoEx PTI FRAME:** EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **CoEx PTI DATA:** EFR32 Frame Control Data Out signal (packet trace data)
- **WiFi TX ACTIVE:** Wi-Fi TX Active signal

The logic analyzer sequence in Figure 2-1 shows:

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

Example 2: Configure EFR32 PTA support to operate with multi-radio 2-Wire PTA with active-low REQUEST (for Series 1)

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

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

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

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

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

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

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

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

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

The logic analyzer capture in Figure 2-2 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:

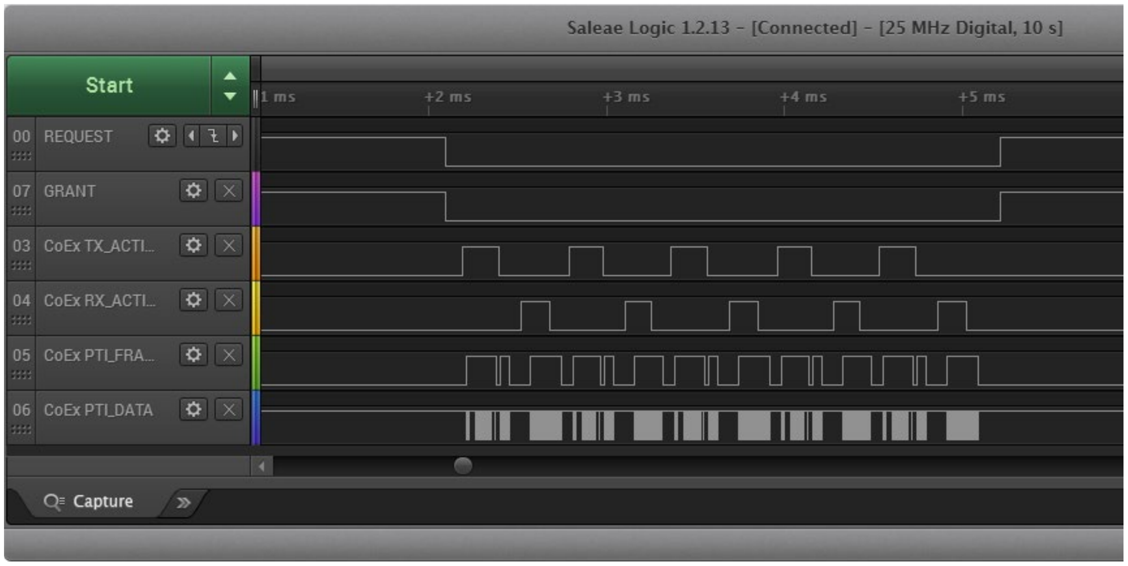


Figure 2-2. Example CONNECTION event (master) for Multi-EFR32 2-Wire Wi-Fi/PTA Logic Analyzer Capture (first anchor point in CONNECTION, using active-low REQUEST)

where:

- **REQUEST:** active low, shared (open-drain) REQUEST input/output
- **GRANT:** active low GRANT input
- **CoEx TX ACTIVE:** EFR32 TX Active control signal (configured via sample code in section 3.1 Example TX_ACTIVE/RX_ACTIVE)
- **CoEx RX ACTIVE:** EFR32 RX Active control signal (configured via sample code in section 3.1 Example TX_ACTIVE/RX_ACTIVE)
- **CoEx PTI FRAME:** EFR32 Frame Control Data Frame signal (packet trace frame/synch)
- **CoEx PTI DATA:** EFR32 Frame Control Data Out signal (packet trace data)

The logic analyzer sequence in Figure 2-2 shows:

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

3. Application Code Coexistence Extensions

3.1. Example TX_ACTIVE/RX_ACTIVE

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

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

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

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

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

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

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

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

4. Coexistence Backplane Evaluation Board (EVB)

For evaluating the Silicon Labs EFR32 software coexistence solution, order EFR32MG Wireless SoC Starter Kit (WSTK) #SLWSTK6000B and Coexistence Backplane EVB (#SLWSTK-COEXBP). Detailed instructions for using the Starter Kit and Backplane EVB are found in *UG350: Silicon Labs Coexistence Development Kit (SLWSTK-COEXBP)*. To see a demonstration of Wi-Fi coexistence and obtain links to additional coexistence documentation, visit the Silicon Labs Wi-Fi Coexistence Learning Center.

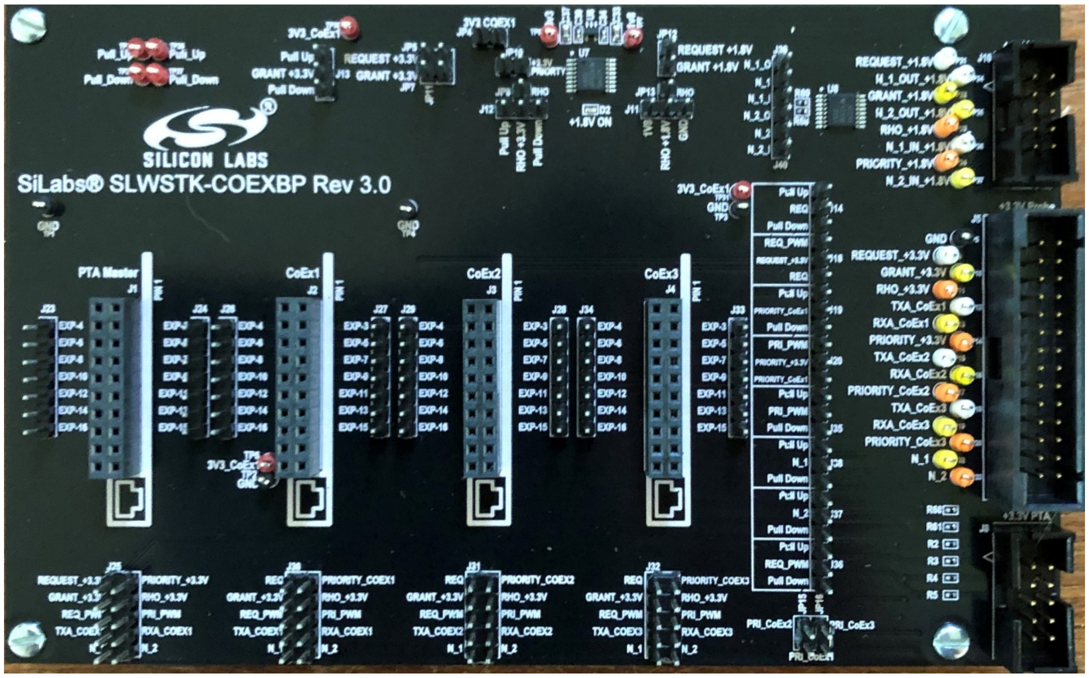


Figure 4-1. Coexistence Backplane EVB (#SLWSTK-COEXBP)

5. Document Revision History

Revision 1.7

January 2021

Bluetooth SDK version: 3.1.0.0 Bluetooth Mesh SDK version: 2.0.0.0

- Update section 2.1 Compile Time PTA Setup and Defaults to stay consistent with gecko SDK v3.1

Revision 1.6

December 2020

Bluetooth SDK version: 3.0.1.0 Bluetooth Mesh SDK version: 1.7.2.0

- Moved section 3 Unmanaged Coexistence, section 4 Managed Coexistence, and section 5 Conclusions from Revision 1.5 of this application note to *UG103.17: Wi-Fi® Coexistence Fundamentals*.

Revision 1.5

June 2020

Bluetooth version: 2.13.5.0 Bluetooth Mesh version: 1.6.3.0

- Renamed section 4.1 to PTA Support Options; added heading level three to 1-Wire PTA, 2-Wire PTA, 3-Wire PTA, and 4-Wire PTA.
- Added section 4.2 Wi-Fi/PTA Considerations, section 4.3 PWM for High Duty Cycle Wi-Fi, and section 4.4 Directional PRIORITY from *AN1243: Timing and Test Data for EFR32 Coexistence with Wi-Fi*.
- Updated section 2.4 Coexistence Configuration Setup Examples for Different Wi-Fi/PTA Applications due to changes in the Bluetooth SDK 3.0.0.
- Updated figures in section **Error! Reference source not found. Error! Reference source not found.** and section **Error! Reference source not found. Error! Reference source not found.**. They use the RACPAEN signal. RACLNAEN is no longer in use.
- Corrected the Static PRIORITY signal assignment in section **Error! Reference source not found. Error! Reference source not found.** and section **Error! Reference source not found. Error! Reference source not found.**.

Revision 1.4

March 2020

Bluetooth version: 2.13.3.0 Bluetooth Mesh version: 1.6.2.0

- Made minor text changes.

Revision 1.3

February 2020

Bluetooth version: 2.13.2.0 Bluetooth Mesh version: 1.6.1.0

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

Revision 1.2

January 2020

Bluetooth version: 2.13.1.0 Bluetooth Mesh version: 1.6.1.0

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

Revision 1.1

December 2019

Bluetooth version: 2.13.0.0 Bluetooth Mesh version: 1.6.1.0

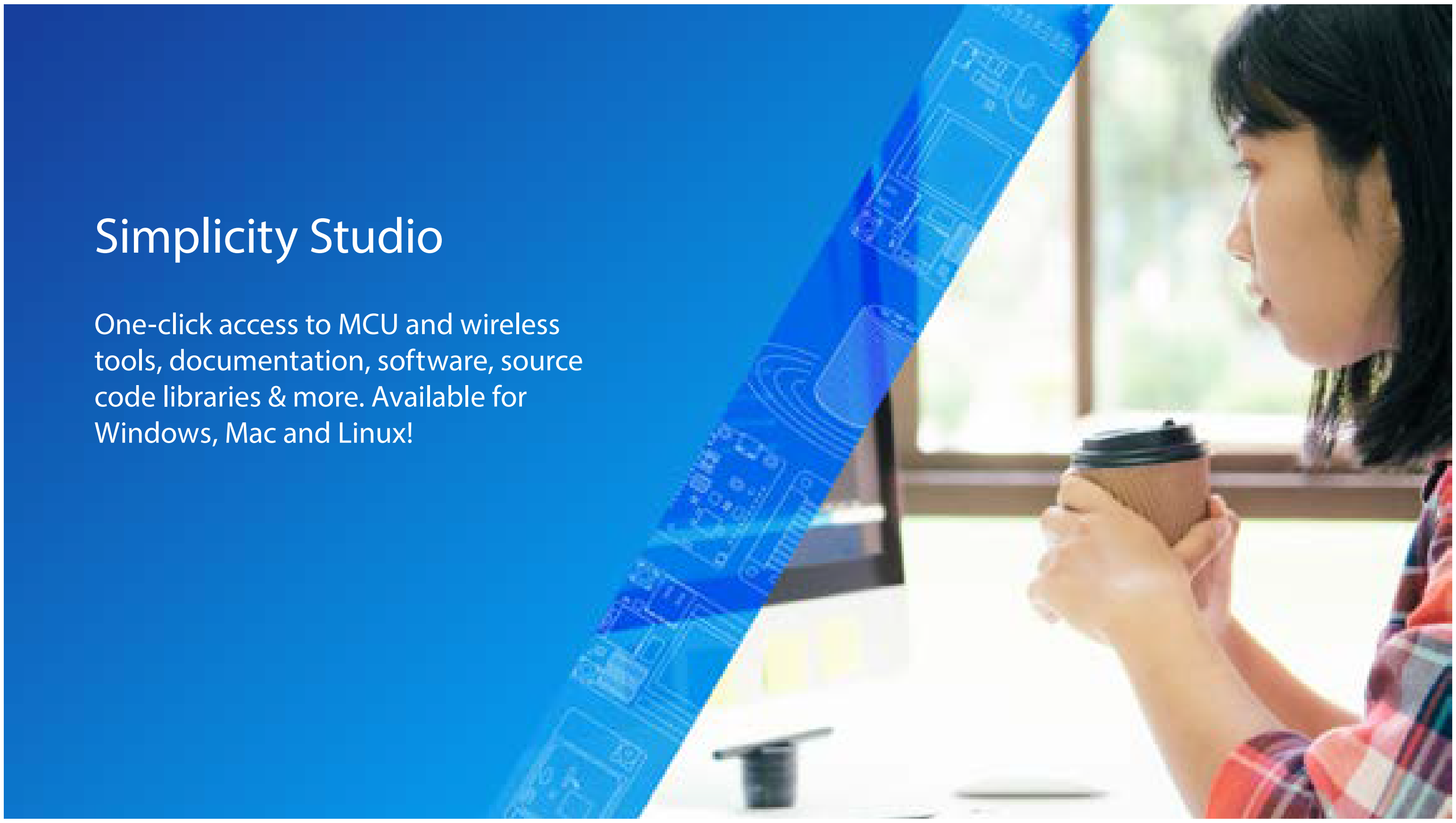
- Added SL Thread notice on first page.
- Added Link Layer PRIORITY support.
- Added 100% Passive SCAN and Bluetooth mesh ADV-Bearer support.
- Added PWM information (not functional in Bluetooth 2.13.0.0).
- Added Directional PRIORITY support.

Revision 1.0

December 2018

Bluetooth version: 2.11.0.0 Bluetooth Mesh version: 2.8.0.0

- Initial release


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 & Community
www.silabs.com/community

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

http://www.silabs.com