S310 nRF51422

Concurrent ANT™ and *Bluetooth*® Low Energy SoftDevice

SoftDevice Specification v1.0

Key Features

- Advanced ANT stack
  - Simple to complex network topologies:
    - Peer-to-peer
    - Star
    - Tree
    - Star-to-star and more
  - Broadcast, acknowledged, and burst communication modes
  - Supports up to 8 logical channels each with configurable channel periods (5.2 ms – 2 s)
  - Built-in device search and pairing
  - Enhanced ANT features
  - Multiple ANT channel concurrency with a single BLE channel
- *Bluetooth*® 4.0 compliant low energy single-mode protocol stack:
  - Link layer
  - L2CAP, ATT, and SM protocols
  - GATT, GAP, and L2CAP
  - Peripheral and Broadcaster roles
  - GATT Client and Server
  - Full SMP support including MITM and OOB pairing

Applications

- Personal area networks
  - Sport and fitness sensors and monitoring devices
  - Healthcare sensors and medical devices
  - Key fobs
- Environmental sensor networks
- Home and industrial automation
- Logistics/goods tracking
- Active RFID
- ANT/*Bluetooth*® low energy protocol bridging

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Life support applications

Nordic Semiconductor's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest distributor, please visit http://www.nordicsemi.com.
Information regarding product updates, downloads, and technical support can be accessed through your My Page account on our homepage.

Main office:
Otto Nielsens veg 12
7052 Trondheim
Norway
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89

Mailing address:
Nordic Semiconductor
P.O. Box 2336
7004 Trondheim
Norway

Document Status

Status	Description
v0.5	This specification contains target specifications for product development.
v0.7	This specification contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
v1.0	This specification contains final product specifications. Nordic Semiconductor ASA reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

Revision History

Date	Version	Description
March 2014	1.0	**Added content:** • ***Chapter 8 "Flash memory API"*** on page 21. • ***Chapter 9 "Radio disable API"*** on page 23. • ***Chapter 10 "Bootloader"*** on page 25. **Updated content:** • Changed value for RAM when S310 is disabled in ***Table 21*** on page 27. • ***Chapter 7 "Radio Notification"*** on page 16. • ***Chapter 11 "SoftDevice performance"*** on page 28. • ***Chapter 14 "ANT power profiles"*** on page 40. • ***Chapter 16 "ANT/BLE concurrency handling"*** on page 48.
December 2013	0.7.1	**Updated content:** • Changed values for RAM when S310 is enabled in ***Table 21*** on page 27. • Changed all references of nRF51922 to nRF51422.
November 2013	0.7	Preliminary release.

1 Introduction

The S310 SoftDevice is an ANT and *Bluetooth*® low energy (BLE) Peripheral controller and host multiprotocol stack that provides a flexible application programming interface (API) for building concurrent ANT and BLE System on Chip (SoC) solutions for Nordic Semiconductor nRF51422 ICs. The S310 SoftDevice simplifies combining an ANT or BLE protocol stack and an application on the same CPU, therefore eliminating the need for an added device to support concurrent multiprotocol.

This document contains information about the SoftDevice features and performance.

Note: The SoftDevice features and performance are subject to change between revisions of this document. See ***Section 17.1 "Notification of SoftDevice revision updates"*** on page 51 for more information. Pre-production versions may not have support for all features. To find information on any limitations or omissions, the SoftDevice release notes will contain a detailed summary of the release status.

1.1 Documentation

Below is a list of the core documentation for the S310 SoftDevice.

Document	Description
nRF51 Series Reference Manual	"Appendix A: SoftDevice architecture" in the *nRF51 Series Reference Manual* is essential reading for understanding the resource usage and performance related chapters of this document.
nRF51422 Product Specification (PS)	Contains a description of the hardware, modules, and electrical specifications specific to the nRF51422 chip.
nRF51422 Product Anomaly Notification (PAN)	Contains information on anomalies related to the nRF51422 chip.
Bluetooth Core Specification	The *Bluetooth Core Specification* version 4.0, Volumes 1, 3, 4, and 6 describes *Bluetooth* terminology which is used throughout the SoftDevice specification.
ANT Message Protocol and Usage	The *ANT Message Protocol and Usage* document describes the ANT protocol in detail and is the starting point for understanding everything else. It contains the fundamental knowledge you need in order to develop successfully with ANT.

1.2 Writing conventions

This SoftDevice Specification follows a set of typographic rules to ensure that the document is consistent and easy to read. The following writing conventions are used:

- Command, event names, and bit state conditions are written in `Lucida Console`.
- Pin names and pin signal conditions are written in **`Consolas.`**
- File names and User Interface components are written in **bold**.
- Internal cross references are italicized and written in ***semi-bold.***
- Placeholders for parameters are written in *italic regular font*. For example, a syntax of the function initialize will be written as: *initialize(parameter1, parameter2)*.
- Fixed parameters are written in regular text font. For example, a syntax description of SetChannelPeriod will be written as: SetChannelPeriod(0, Period).

2 Product overview

This section provides an overview of the S310 SoftDevice.

2.1 SoftDevice

The S310 SoftDevice is a precompiled and linked binary software implementing an 8 channel ANT protocol stack combined with a BLE Peripheral protocol stack on the nRF51422 chip. The Application Programming Interface (API) is a standard C language set of functions and data types that give the application complete compiler and linker independence from the SoftDevice implementation.

The SoftDevice enables the application programmer to develop their code as a standard ARM® Cortex™-M0 project without needing to integrate with proprietary chip-vendor software frameworks. This means that any ARM® Cortex™-M0 compatible toolchain can be used to develop ANT/ANT+ and *Bluetooth* low energy applications with the S310 SoftDevice.

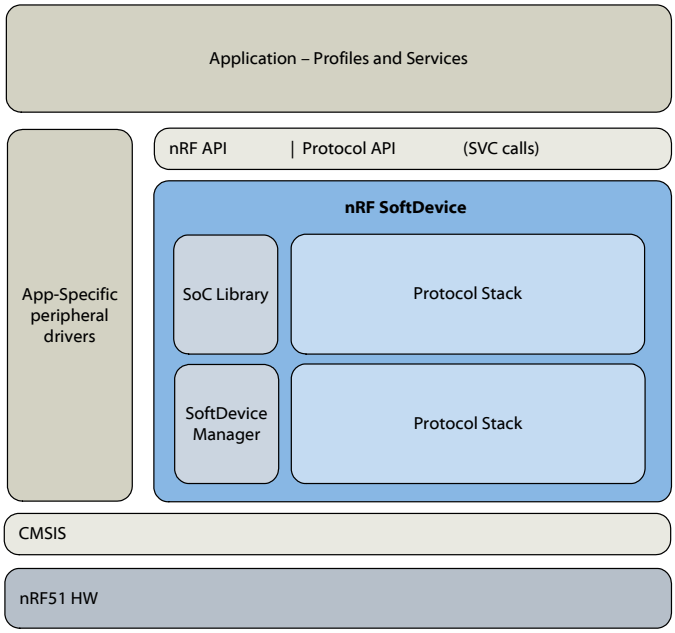

Figure 1 *System on Chip application with the SoftDevice*

The S310 SoftDevice can be programmed onto compatible nRF51 devices during both development and production. This specification outlines the supported features of a Production level SoftDevice. Alpha and beta versions may not support all features.

2.2 Multiprotocol support

The S310 SoftDevice supports non-concurrent multiprotocol implementations. This means that in addition to the concurrent operating stacks in S310, a proprietary 2.4 GHz protocol can be implemented in the application program area and can access all hardware resources when the SoftDevice is disabled.

3 ANT protocol stack

The SoftDevice is a fully qualified ANT compliant stack that supports all ANT core functionality, including ANT+ profiles. See http://thisisant.com for further information regarding ANT.

Note: ANT+ implementations must pass the ANT+ certification process to receive ANT+ Compliance Certificates.

3.1 Overview

The nRF51 Software Development Kit (SDK) supplements the ANT protocol stack with complete peripheral drivers, example applications, and ANT+ profile implementations.

Note: ANT+ network keys are needed to make ANT+ compliant products. The keys can be obtained by registering as an ANT+ adopter at http://thisisant.com.

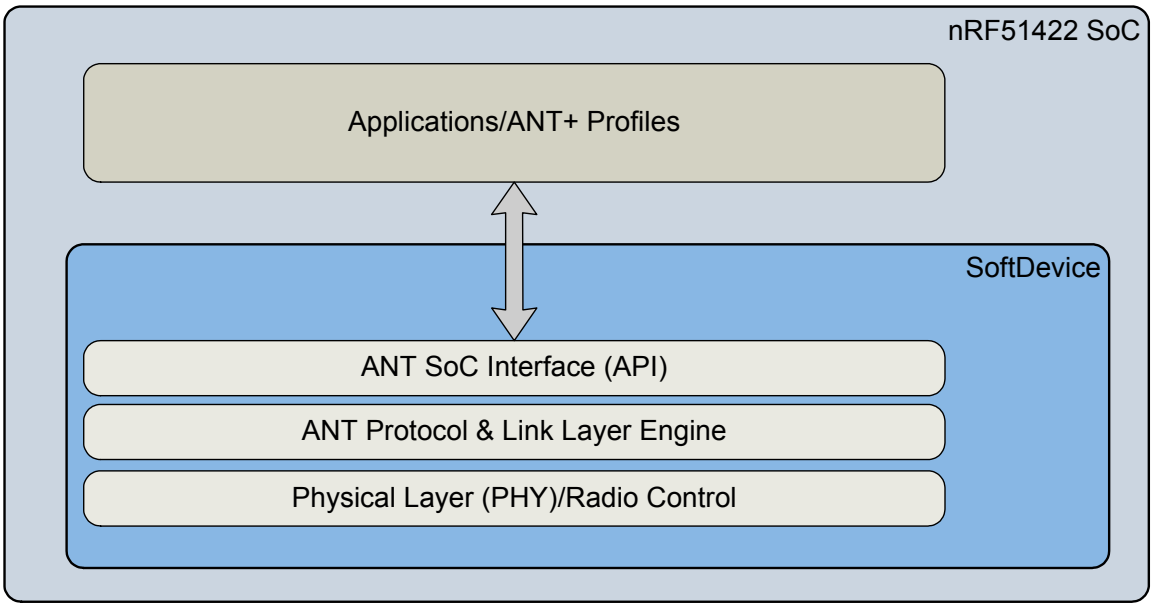

Figure 2 *ANT stack architecture*

3.2 ANT profile and feature support

All ANT messages supported by the nRF24AP1 and nRF24AP2 devices are described in "Table 4 ANT message summary supported by nRF24AP2" of the *nRF24AP2 Product Specification* are supported by the SoftDevice. Additional enhanced ANT features are available in the SoftDevice and are described below.

3.2.1 Advanced Burst Transfer

Advanced Burst Transfer is intended to facilitate and improve the ability of devices to transfer information. This is employed primarily in ANT-FS use cases, with an emphasis on longer file transfers. Advanced Burst Transfer improves the efficiency of transferring a file by increasing the transfer speed (up to 60 kbps) while providing greater immunity to RF interference. Burst stalling and adjustable retry mechanisms are introduced to allow greater flexibility of host data source and sink rates.

Note: The peer device must also support this feature to achieve higher transfer rates. This feature is backward compatible with existing burst transfer methods (up to 20 kbps).

3.2.2 Single channel encryption

The ANT channel encryption engine provides the ability for a single channel to transmit and receive data in a secure manner using AES-128. This feature simplifies applications that require data security such as medical and health devices. Broadcast messaging, acknowledged messaging, and burst transfers/advanced burst transfers are supported data communication modes for encryption.

ANT encryption mode allows broadcast data to be received by multiple receivers. Alternatively, point-to-point encrypted links can be created through the use of inclusion/exclusion encryption ID list.

3.2.3 Multiple network keys

For use cases that benefit from transmission of data on multiple networks, the SoftDevice will support use of up to 8 public, private, and/or managed (ANT+) network keys.

3.2.4 Event filtering

Event filtering is provided for the application to avoid or reduce processing of ANT events, in an effort to conserve power consumption and/or resources. The application can select which event messages to block from the ANT protocol layer to the application.

3.2.5 Selective Data Updates

Selective Data Updates is another feature intended to aid in managing power consumption of the application. The facility may be used when an application needs to process received messages only if the specified data has changed. This could apply to devices such as display units that update the display only when the displayed data has actually changed and are otherwise asleep. This feature can be enabled for Broadcast messages only or Broadcast and Acknowledged messages. This feature does not apply to Burst Transfers.

3.2.6 Asynchronous Transmission channel

Asynchronous Transmission mode introduces the ability for users to initiate transmissions that are not bound by any specified periods or intervals. Asynchronous channels do not need to be “opened” and are simply initiated by sending data to the channel. Asynchronous channels may be used when user generated input on a device needs to cause data transmission to a receiver with minimal delay (for example remote control applications). Receivers must employ scanning channels to effectively use this feature. Broadcast messaging, acknowledged messaging, and burst transfers are supported in this mode.

3.2.7 Fast Channel Initiation

Enabling the Fast Channel Initiation feature allows ANT synchronous transmission channels to start as soon as possible. The latency from the channel opening to transmission is reduced. This feature should only be used in scenarios where a device opens for a brief period of time and requires low latency.

4 *Bluetooth* low energy protocol stack

The *Bluetooth* 4.0 compliant low energy (BLE) Host and Controller embedded in the SoftDevice are fully qualified with multi-role support (Peripheral and Broadcaster). The API is defined above the Generic Attribute Protocol (GATT), Generic Access Profile (GAP), and Logical Link Control and Adaptation Protocol (L2CAP). The SoftDevice allows applications to implement standard *Bluetooth* low energy profiles as well as proprietary use case implementations.

The nRF51 Software Development Kit (SDK) completes the BLE protocol stack with Service and Profile implementations. Single-mode System on Chip (SoC) applications are enabled by the full BLE protocol stack and nRF51xxx integrated circuit (IC).

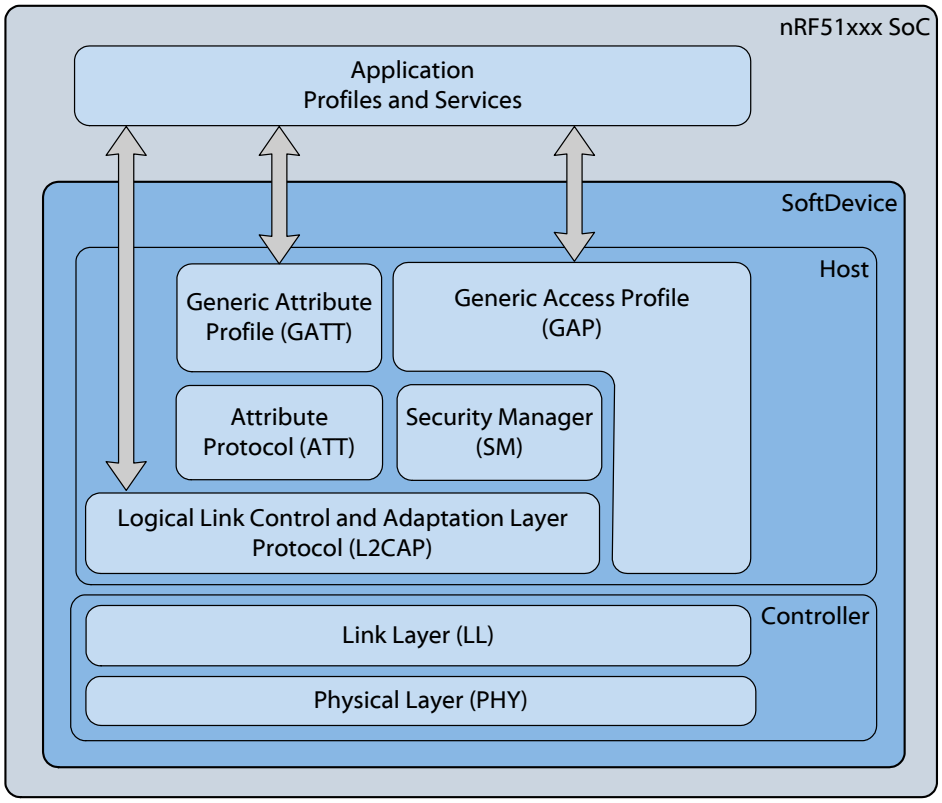

Figure 3 *SoftDevice stack architecture*

4.1 Profile and service support

The Profiles and corresponding Services supported by the SoftDevice are shown in ***Table 1***.

Adopted Profile	Adopted Services	Supported
HID over GATT	HID Battery Device Information	YES
Heart Rate	Heart Rate Device Information	YES
Proximity	Link Loss Immediate Alert TX Power	YES
Blood Pressure	Blood pressure	YES
Health Thermometer	Health Thermometer	YES
Glucose	Glucose	YES
Phone Alert Status	Phone Alert Status	YES
Alert Notification	Alert Notification	YES
Time	Current Time Next DST Change Reference Time Update	YES
Find Me	Immediate Alert	YES
Cycling speed and cadence	Cycling speed and cadence Device information	YES
Running speed and cadence	Running speed and cadence Device information	YES
Location and Navigation	Location and Navigation	YES
Cycling Power	Cycling Power	YES
Scan Parameters	Scan Parameters	YES

Table 1 *Adopted Profile and Service support*

Note: Examples for selected profiles and services are available in the nRF51 SDK. See the SDK documentation for details.

4.2 *Bluetooth* low energy features

The BLE protocol stack in the SoftDevice has been designed to provide an abstract but flexible interface for application development for *Bluetooth* low energy devices. GAP, GATT, SM, and L2CAP are implemented in the SoftDevice and managed through the API. GAP and GATT procedures and modes that are common to most profiles, such as the handling of discoverability, connectability, pairing, and bonding, are implemented in the stack.

The BLE API is consistent across *Bluetooth* role implementations where common features have the same interface. The following tables describe the features found in the BLE protocol stack.

API Features	Description
Interface to: GATT/GAP/L2CAP	Consistency between APIs including shared data formats.
GATT DB population and access	Full flexibility to populate the DB at run time, attribute removal is not supported.
Thread-safe, asynchronous, and event driven	Minimizes exposure to concurrency issues.
Vendor-specific (128 bit) UUIDs for proprietary profiles	Compact, fast, and memory efficient management of 128 bit UUIDs.
Packet flow control	Zero-copy buffer management.

***Table 2** API features in the BLE stack*

GAP Features	Description
Multi-role: Peripheral and Broadcaster	
Multiple bond support	Keys and peer information stored in application space. No limitations in stack implementation.
Security mode 1: Levels 1, 2, and 3	
User-defined Advertising data	Full control over advertising and scan response data for the application.
Limited and general discoverable modes	

***Table 3** GAP features in the BLE stack*

GATT Features	Description
Comprehensive GATT Server	
Support for authorization: R/W characteristic value R/W descriptors	Enables control points Enables freshest data Enables GAP authorization
Full GATT Client	Flexible data management options for packet transmission with either fine control or abstract management
Implemented GATT Sub-procedures	Discover all Primary Services Discover Primary Service by Service UUID Find included Services Discover All Characteristics of a Service Discover Characteristics by UUID Discover All Characteristic Descriptors Read Characteristic Value Read using Characteristic UUID Read Long Characteristic Values Write Without Response Write Characteristic Value Notifications Indications Read Characteristic Descriptors Read Long Characteristic Descriptors Write Characteristic Descriptors Write Long Characteristic Values Write Long Characteristic Descriptors Reliable Writes

Table 4 *GATT features in the BLE stack*

Security Manager Features	Description
Lightweight key storage for reduced NV memory requirements	
Authenticated MITM (Man in the middle) protection	
Pairing methods: Just works, Passkey Entry, and Out of Band	

Table 5 *Security Manager (SM) features in the BLE stack*

ATT Features	Description
Server protocol	
Client protocol	

Table 6 *Attribute Protocol (ATT) features in the BLE stack*

L2CAP Features	Description
27 byte MTU size	
Dynamically allocated channels	

Table 7 *Logical Link Control and Adaptation Layer Protocol (L2CAP) features in the BLE stack*

Controller, Link Layer Features	Description
Slave role	
Slave connection update	
27 byte MTU	
Encryption	

Table 8 *Controller, Link Layer (LL) features in the BLE stack*

Proprietary Feature	Description
TX Power control	Access for the application to change TX power settings anytime.

Table 9 *Proprietary features in the BLE stack*

5 SoC library

The following features ensure the Application and SoftDevice coexist with safe sharing of common SoC resources.

Feature	Description
Mutex	Atomic mutex API. Disabling global interrupts in the application could cause dropped packets or lost connections. This API safely implements an atomic operation for the application to use.
NVIC	Gives the application access to all NVIC features without corrupting SoftDevice configurations.
Rand	Gives access to the random number generator hardware.

Power	Access to POWER block configuration while the SoftDevice is enabled:
	• Access to RESETREAS register
	• Set power modes
	• Configure power fail comparator
	• Control RAM block power
	• Use general purpose retention register
	• Configure DC/DC converter state
	  • OFF
	  • ON
	  • AUTOMATIC - The SoftDevice will manage the DC/DC converter state by switching it on for all Radio Events and off all other times.

Clock	Access to CLOCK block configuration while the SoftDevice is enabled. Allows the HFCLK Crystal Oscillator source to be requested by the application.
Wait for event	Simple power management hook for the application to use to enter a sleep or idle state and wait for an event.
PPI	Configuration interface for PPI channels and groups reserved for an application.
Radio disable API	Requests short periods of guaranteed radio inactivity for application activity.
Radio notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See ***Chapter 7 "Radio Notification"*** on page 16.
Block encrypt (ECB)	Safe use of 128 bit AES encrypt HW accelerator.
Event API	Fetch asynchronous events generated by the SoC library.
Flash memory API	Application access to flash write, erase, and protect. Can also safely be used during active BLE connections and ANT activities.
Temperature	Application access to the temperature sensor.

Table 10 *System on Chip features*

6 SoftDevice Manager

The following feature enables the Application to manage the SoftDevice on a top level.

Feature	Description
SoftDevice control API	Control of SoftDevice state through enable and disable. On enable, the low frequency clock source selects between the following options: • RC oscillator • Synthesized from high frequency clock • Crystal oscillator

***Table 11** SoftDevice Manager*

Note: The BLE and ANT protocols are active when the SoftDevice is in the enabled state where they can be controlled through their respective APIs.

7 Radio Notification

Radio Notification is a configurable feature that enables ACTIVE and INACTIVE (nACTIVE) signals from the SoftDevice to the application notifying when the Radio is in use. The signal is sent using software interrupt, as specified in ***Table 24*** on page 29.

The ACTIVE signal, if enabled, is sent before the Radio Event starts. The nACTIVE signal is sent at the end of the Radio Event. These signals can be used by the application programmer to synchronize application logic with Radio activity and in the BLE stack, packet transfers. For example, the ACTIVE signal can be used to shut off external devices to manage peak current drawn during periods when the radio is on, or to trigger sensor data collection for transmission in the Radio Event.

7.1 ANT

It is recommended to use Radio Notification to synchronize application logic with Radio Activity, but it should not be used to synchronize with packet transfers. Instead, ANT event messages should be used as triggers for data loading. ***Figure 4*** shows the active signal in relation to ANT radio activities where t_{ndist} is the time between ACTIVE and the first TX or RX activity of an ANT event.

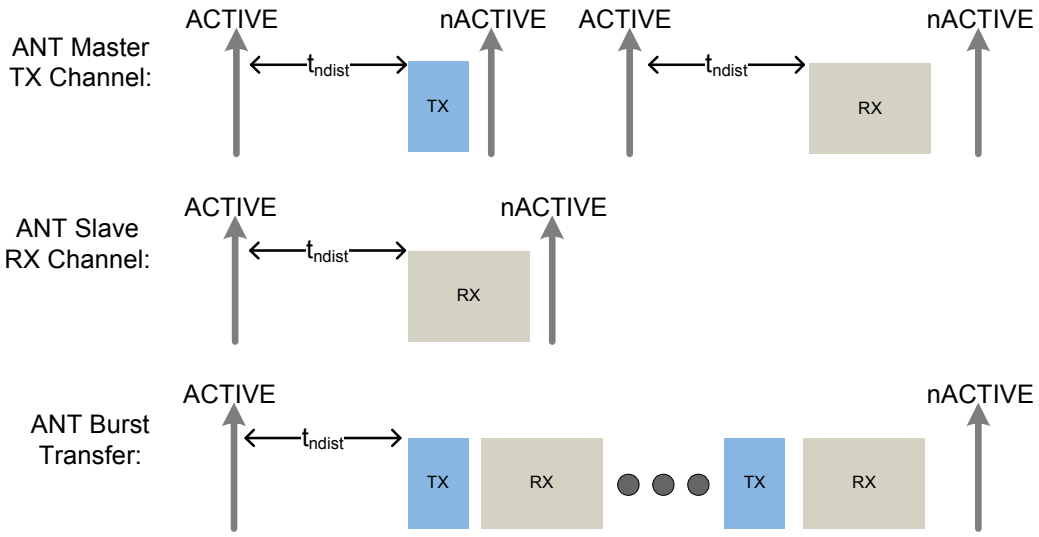

Figure 4 *Radio Notifications*

Table 12 shows the ranges of the timing symbols in ***Figure 4***.

ANT activity	Value	Range (µs)
ANT RX Search, Scanning, TX/RX Broadcast, TX/RX Acknowledged, TX/RX Burst Transfer	t_{ndist} [1]	800, 1740, 2680, 3620, 4560, 5500

1. In concurrent operations, the minimum t_{ndist} should be used (800 µs).

Table 12 *ANT Radio Notification timing ranges*

Extremely fast ANT channel intervals (for example > 200 Hz) may not generate any radio notifications and are treated as one continuous radio event.

The previously supported feature, ANT RFActive Notification, is still available for use through the ANT API set. However, it is recommended that the SoC Radio Notification be used instead due to the following reasons:

- SoC radio notification uses dedicated software interrupt for immediate notification to the application. ANT RFActive Notification uses the ANT event queue for notification and could be subjected to application event processing delay.
- Notifications in concurrent multi-protocol radio solutions are properly handled by the SoC Radio Notification feature. ANT RFActive Notification only applies to ANT radio events.

7.2 BLE

Figure 5 shows the active signal in relation to the Radio Event.

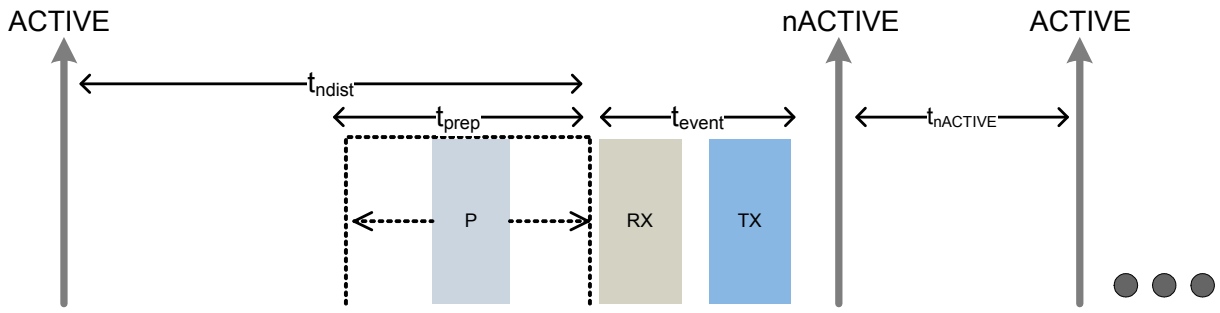

Figure 5 *BLE Radio Notification*

Many packets can be sent and received in one Radio Event. Radio Notification events will be as shown in ***Figure 6***.

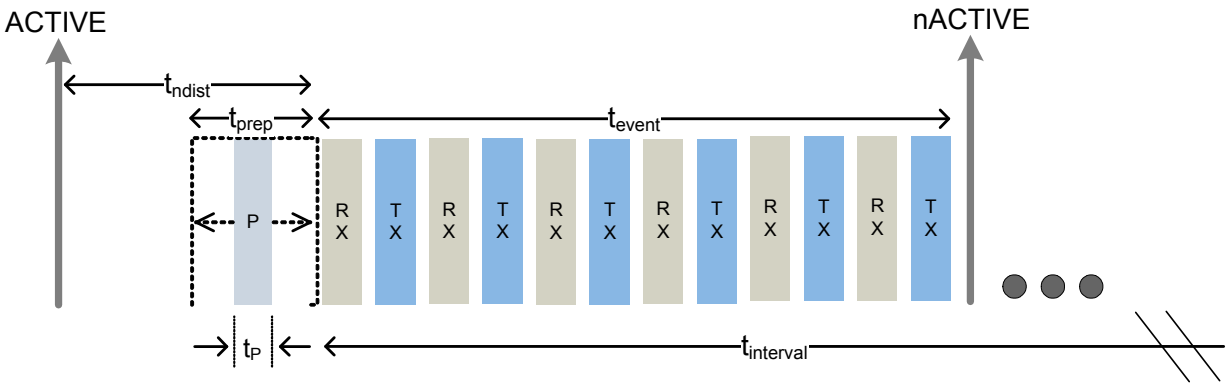

Figure 6 *BLE Radio Notification, multiple packet transfers*

Table 13 describes the notation used in ***Figure 5*** on page 17 and ***Figure 6***.

Label	Description	Notes
ACTIVE	The ACTIVE signal prior to a Radio Event.	
nACTIVE	The nACTIVE signal after a Radio Event.	Because both ACTIVE and nACTIVE use the same software interrupt, it is up to the application to manage them. If both ACTIVE and nACTIVE are configured ON by the application, there will always be an ACTIVE signal before an nACTIVE signal.
P	CPU processing in the lower stack interrupt between ACTIVE and RX.	The CPU processing may occur anytime, up to t_{prep} before RX.
RX	Reception of packet.	
TX	Transmission of packet.	
t_{ndist}	The notification distance - the time between ACTIVE and first RX/TX in a Radio Event.	This time is configurable by the application developer and can be changed in between Radio Events.
t_{event}	The time used in a Radio Event.	
t_{prep}	The time before first RX/TX to prepare and configure the radio.	The application will be interrupted by the LowerStack during t_{prep}. **Note**: All packet data to send in an event should be sent to the stack t_{prep} before the Radio starts.
t_P	Time used for preprocessing before the Radio Event.	
$t_{interval}$	Time between Radio Events as per the protocol.	

Table 13 *Radio Notification figure labels*

Table 14 shows the ranges of the timing symbols in ***Figure 5*** on page 17.

Value	Range (µs)
t_{ndist}	800, 1740, 2680, 3620, 4560, 5500
t_{event}	550 to 4850 - Undirected and scannable advertising, 0 to 31 byte payload, 3 channels 550 to2250 - Non-connectable advertising, 0 to 31 byte payload, 3 channels 1.28 seconds - Directed advertising, 3 channels 900 to 5400 Slave - 1 to 6 packets RX and TX unencrypted data when connected 1000 to 5800 Slave - 1 to 6 packets RX and TX encrypted data when connected
t_{prep}	290 to 1550
t_P	≤150

Table 14 *BLE Radio Notification timing ranges*

Using the numbers from ***Table 14***, the amount of CPU time available between ACTIVE and a Radio Event is:

$$t_{ndist} - t_P$$

Shown here is the amount of time before stack interrupts begin. Data packets must be transferred to the stack using the API within this time from the ACTIVE signal if they are to be sent in the next Radio Event.

$$t_{ndist} - t_{prep(maximum)}$$

Note: t_{prep} may be greater than t_{ndist} when t_{ndist} = 800. If time is required to handle packets or manage peripherals before interrupts are generated by the stack, t_{ndist} should be set greater than 1500.

To maximize the chance that the notification signal is available to the application in the configured time, the following rule is applied:

$$t_{ndist} + t_{event} < t_{interval}$$

To maximize the probability that this rule is true, the stack may limit the length of a Radio Event (t_{event}) thereby reducing the maximum throughput and effecting maximum data rate. ***Figure 7*** shows consecutive Radio Events with Radio Notification and illustrates the limitation in t_{event} which may be required to ensure t_{ndist} is preserved.

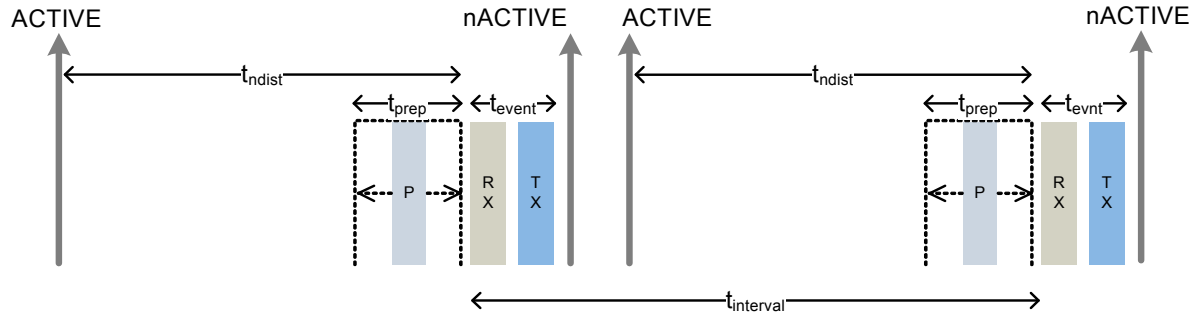

Figure 7 *Consecutive Radio Events with BLE Radio Notification*

Table 15 shows the limitation on the maximum number of packets which can be transferred per Radio Event given a t_{ndist} and $t_{interval}$ combination.

t_{ndist}	$t_{interval}$		
	7.5 ms	**10 ms**	**≥ 15 ms**
800	6	6	6
1740	5	6	6
2680	4	6	6
3620	3	5	6
4560	2	4	6
5500	1	3	6

Table 15 *Maximum packet transfer per BLE Radio Event for given combinations of t_{ndist} and $t_{interval}$.*

8 Flash memory API

Asynchronous flash memory operations are performed using the SoC library API and provide the application with flash write, flash erase, and flash protect support through the SoftDevice. This interface can safely be used during ANT radio activities and active *Bluetooth* connections.

When the BLE and ANT stacks are operating concurrently, or when the Radio Disable API is used, flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR may be generated more often, especially if the stacks are operating with high connection frequencies/low connection intervals. If this occurs, retry the flash erase or write operation.

8.1 ANT

The flash memory access is scheduled in between the protocol radio events. The time required for memory access may be larger than is allowed for certain ANT operations.

- ANT Burst Transfers - In this case, flash operations may be denied or delayed until the ANT burst transfer is finished.
- ANT Transmit/Receive Keep Alive – These critical ANT transmit/receive radio operations may delay flash operation requests.

ANT activity	Flash write
ANT RX Search/Scanning ANT TX/RX Broadcast ANT TX/RX Acknowledged	Typically allows full write size (256 words) attempts. May generate flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR if critical radio activities needs to occur. In this case, retry flash write.
ANT TX/RX Burst Transfer	• Maximum flash write size attempt for concurrent operation: 48 words. Up to 30% reduction of burst transfer rate during continuous flash write activity. May generate flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR if critical radio activities needs to occur. In this case, retry flash write. • Larger flash write size attempts, in general, will consistently generate flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR until the burst transfer has ended. In this case, retrying flash writes will only succeed after the burst activity has finished.
ANT activity	**Flash erase**
ANT RX Search/Scanning ANT TX/RX Broadcast ANT TX/RX Acknowledged	Typically allows flash erase attempts. May generate flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR if critical radio activities needs to occur. In this case, retry flash erase.
ANT TX/RX Burst Transfer	Flash erase attempts, in general, will consistently generate flash timeout event: NRF_EVT_FLASH_OPERATION_ERROR until the burst transfer has ended. In this case, retrying flash erases will only succeed after the burst activity has finished.

Table 16 *Behavior with ANT traffic and concurrent flash write/erase*

8.2 BLE

The flash memory access is scheduled in between the protocol radio events. For short connection or advertisement intervals, the time required for the flash memory access may be larger than the connection or advertisement interval. In this case, protocol radio events may be skipped, up to a maximum of three connection events or one advertisement event. The flash memory access may also be delayed slightly to minimize the disturbance of the BLE radio protocol. In some cases as described below, the flash memory access may fail and generate a timeout event: `NRF_EVT_FLASH_OPERATION_ERROR`. In this case, retry the flash erase or write operation.

BLE activity	Flash write
BLE Connectable Undirected Advertising BLE Nonconnectable Advertising BLE Scannable Advertising	Typically allows full write size (256 words) attempts.
BLE Connectable Directed Advertising	Does not allow write attempts while advertising is active (maximum 1.28 seconds). In this case, retrying flash writes will only succeed after the advertising activity has finished.
BLE Connected state	Typically allows full write size (256 words) attempts. May generate flash timeout event: `NRF_EVT_FLASH_OPERATION_ERROR` if critical radio events need to occur. Critical radio events are expected at connection setup, at connection update, at disconnection and just before supervision timeout. In this case, retry the flash write operation.
BLE activity	**Flash erase**
BLE Connectable Undirected Advertising BLE Nonconnectable Advertising BLE Scannable Advertising	Typically allows flash erase attempts.
BLE Connectable Directed Advertising	Does not allow flash erase attempts while advertising is active (maximum 1.28 seconds). In this case, retrying flash erase will only succeed after the directed advertising is finished.
BLE Connected state	Typically allows flash erase attempts. May generate flash timeout event: `NRF_EVT_FLASH_OPERATION_ERROR` if critical radio events need to occur. Critical radio events are expected at connection setup, at connection update, at disconnection and just before supervision timeout. In this case, retry the flash erase operation.

Table 17 *Behavior with BLE traffic and concurrent flash write/erase*

9 Radio disable API

The SoC library API provides a set of interfaces in which the application can request short periods of time where radio activity will not occur. This is useful in particular situations where the application may need to reserve guaranteed time for activities and cannot run/rely on signals from SoC Radio Notification.

Similar to Flash Memory API, if the radio disable request is larger than allowed in certain protocol states, then the operation may be delayed or denied. If the radio disable session is denied, the application is notified of this occurrence via NRF_EVT_RADIO_BLOCKED or NRF_EVT_RADIO_CANCELED event. The application may choose to retry the operation after receiving these events. Shorter requests have a higher probability of succeeding than longer requests. Judicious use of this feature is required as this can impact overall radio protocol performance.

9.1 ANT

This section describes the Radio disable API used with the ANT protocol stack.

ANT activity	Radio disable
ANT RX Search	Typically allows up to 20 ms radio disable sessions. May generate session blocked event NRF_EVT_RADIO_BLOCKED or NRF_EVT_RADIO_CANCELED if critical radio activities needs to occur. Frequent radio disable sessions can impact search performance.
ANT RX Scanning	Will generally not allow any radio disable sessions.
ANT TX/RX Broadcast ANT TX/RX Acknowledged	Typically allows up to full duration (100ms) radio disable sessions. May generate session blocked event NRF_EVT_RADIO_BLOCKED or NRF_EVT_RADIO_CANCELED if critical radio activities needs to occur. Frequent radio disable sessions can impact broadcast/acknowledged message throughput.
ANT TX/RX Burst Transfer	• Maximum radio disable session size: 1.25 ms. Up to 50% reduction of burst transfer rate during continuous radio disable requested sessions. May generate session blocked event NRF_EVT_RADIO_BLOCKED or NRF_EVT_RADIO_CANCELED if critical radio activities need to occur. • Larger requested radio disable session time, in general, will consistently generate session blocked event NRF_EVT_RADIO_BLOCKED or NRF_EVT_RADIO_CANCELED until the burst transfer has ended. In this case, radio disable session can only occur outside of ANT burst transfers.

Table 18 *Behavior with ANT traffic and concurrent Radio Disable*

9.2 BLE

This section describes the Radio disable API used with the BLE protocol stack.

BLE activity	Radio disable
BLE Connectable Undirected Advertising BLE Nonconnectable Advertising BLE Scannable Advertising	Typically allows radio disable sessions of length up to the advertisement interval. Frequent radio disable sessions can impact advertising throughput.
BLE Connectable Directed Advertising	Does not allow radio disable sessions while advertising is active (maximum 1.28 seconds). In this case, requesting a radio disable session will only succeed after the directed advertising is finished.
BLE Connected state	Typically allows radio disable sessions of length up to 2.5 times the connection interval. May generate session blocked event `NRF_EVT_RADIO_BLOCKED` or `NRF_EVT_RADIO_CANCELED` if critical radio events need to occur. Critical radio events are expected at connection setup, at connection update, at disconnection and just before supervision timeout.

Table 19 *Behavior with BLE traffic and concurrent Radio Disable*

Note: When the BLE and ANT stacks are operating concurrently, session blocked events `NRF_EVT_RADIO_BLOCKED` or `NRF_EVT_RADIO_CANCELED` may be generated more often, especially if the stacks are operating with high connection frequencies/low connection intervals. In this case, retry the radio disable request.

10 Bootloader

The SoftDevice supports the use of a bootloader. A bootloader has access to the full SoftDevice API and can be implemented just as any other application that uses a SoftDevice. In particular, the bootloader can make use of the SoftDevice API to enable protocol stack interaction.

The use of a bootloader is supported in the SoftDevice architecture by dividing the application code space region (R1) into two separate regions. The lower region, from CLENR0 and upwards, contains the application, while the upper region contains the bootloader. The start of the upper region, the bootloader's base address, is set by the UICR.BOOTADDR register.

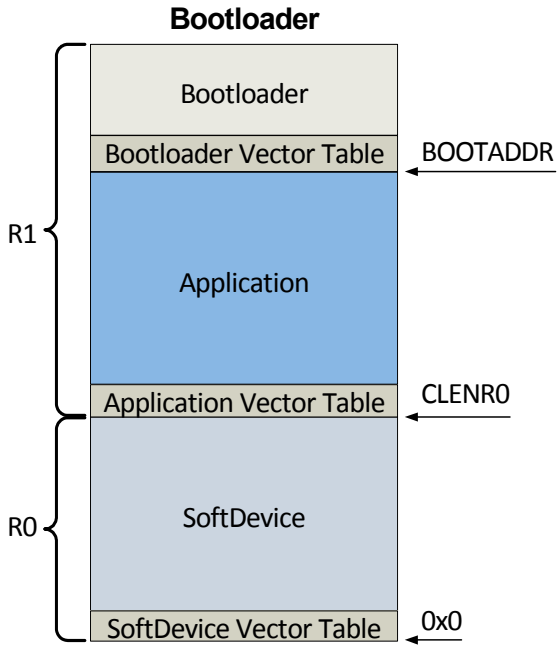

Figure 8 *R0 (SoftDevice) and R1 (Application + Bootloader).*

At reset, the SoftDevice checks the UICR.BOOTADDR register. If this register is blank (0xFFFFFFFF), the SoftDevice assumes that no bootloader is present. It then forwards interrupts to the application and executes the application as usual. If the BOOTADDR register is set to an address different from 0xFFFFFFFF, the SoftDevice assumes that the bootloader vector table is located at this address. Interrupts are then forwarded to the bootloader at this address and execution will be started at the bootloader reset handler.

For a bootloader to transfer execution from itself to the application, the bootloader should first call the `sd_softdevice_forward_to_application()` SoC function to forward interrupts to the application instead of to the bootloader. The bootloader should then branch to the application's reset handler after reading the address of the handler from the Application Vector Table.

UICR.BOOTADDR contents	Interpretation
0xFFFFFFFF	No bootloader present or enabled.
<ADDR>	Bootloader present with base address <ADDR>.

Table 20 *UICR.BOOTADDR register contents*

11 SoftDevice resource requirements

After the SoftDevice is installed on a System on Chip (SoC) it is located in the lower part of the code memory space. When enabled, the SoftDevice controls and uses resources from the chip, including reserving RAM space for its operation and access to hardware peripherals. This chapter describes how the SoftDevice – when both enabled and disabled - uses memory and hardware resources.

11.1 Memory resource map and usage

The memory map for program memory and RAM at run time with the SoftDevice enabled is illustrated in ***Figure 9*** below. Memory resource requirements, both when the SoftDevice is enabled and disabled, are shown in ***Table 21*** on page 27.

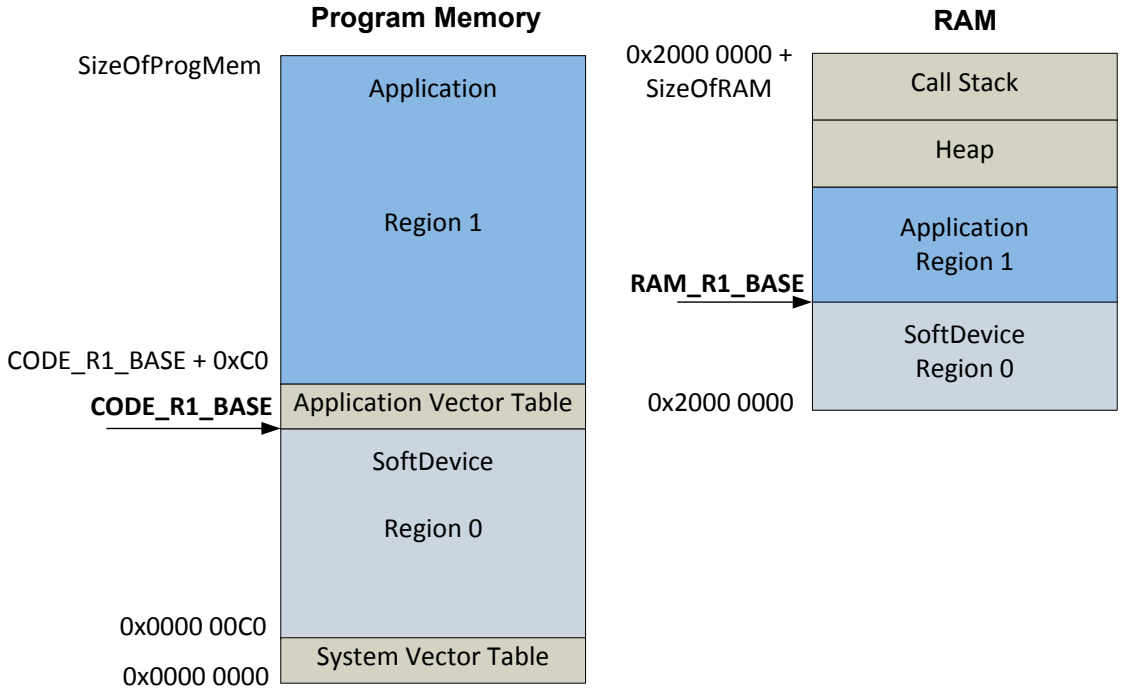

Figure 9 *Memory resource map*

Flash	S310 Enabled	S310 Disabled
Amount	128 kB	128 kB
CODE_R1_BASE	0x0002 0000	0x0002 0000

RAM	S310 Enabled	S310 Disabled
Amount	9 kB	4 bytes
RAM_R1_BASE	0x2000 2400	0x2000 0004

Call stack[1]	S310 Enabled	S310 Disabled
Maximum usage	2 kB	0 kB

Heap	S310 Enabled	S310 Disabled
Maximum allocated bytes	0 bytes	0 bytes

1. This is only the call stack used by the SoftDevice at run time. The application call stack memory usage must be added for the total call stack size to be set in the user application.

Table 21 *S310 Memory resource requirements*

11.2 Hardware blocks and interrupt vectors

Table 22 defines access types used to indicate the availability of hardware blocks to the application. ***Table 23*** on page 28 specifies the access the application has, per hardware block, both when the SoftDevice is enabled and disabled.

Access	Definition
Restricted	Used by the SoftDevice and outside the application sandbox. Application has limited access through the SoftDevice API.
Blocked	Used by the SoftDevice and outside the application sandbox. Application has no access.
Open	Not used by the SoftDevice. Application has full access.

Table 22 *Hardware access type definitions*

ID	Base address	Instance	Access (SoftDevice enabled)	Access (SoftDevice disabled)
0	0x40000000	MPU	Restricted	Open
0	0x40000000	POWER	Restricted	Open
0	0x40000000	CLOCK	Restricted	Open
1	0x40001000	RADIO	Blocked	Open
2	0x40002000	UART0	Open	Open
3	0x40003000	SPI0/TWI0	Open	Open
4	0x40004000	SPI1/TWI1/SPIS1	Open	Open
		...		
6	0x40006000	GPIOTE	Open	Open
7	0x40007000	ADC	Open	Open
8	0x40008000	TIMER0	Blocked	Open
9	0x40009000	TIMER1	Open	Open
10	0x4000A000	TIMER2	Open	Open
11	0x4000B000	RTC0	Blocked	Open
12	0x4000C000	TEMP	Restricted	Open
13	0x4000D000	RNG	Restricted	Open
14	0x4000E000	ECB	Restricted	Open
15	0x4000F000	CCM	Blocked	Open
15	0x4000F000	AAR	Blocked	Open
16	0x40010000	WDT	Open	Open
17	0x40011000	RTC1	Open	Open
18	0x40012000	QDEC	Open	Open
19	00x4001300	LCOMP	Open	Open
20	0x40014000	Software interrupt	Open	Open
21	0x40015000	Radio Notification	Restricted[1]	Open
22	0x40016000	ANT/BLE/SoC Events	Blocked	Open
23	0x40017000	Software interrupt	Blocked	Open
24	0x40018000	Software interrupt	Blocked	Open
25	0x40019000	Software interrupt	Blocked	Open
		...		
30	0x4001E000	NVMC	Restricted	Open
31	0x4001F000	PPI	Restricted	Open
NA	0x50000000	GPIO	Open	Open
NA	0xE000E100	NVIC	Restricted[2]	Open

1. Blocked only when radio notification signal is enabled. See ***Table 24*** on page 29 for software interrupt allocation.
2. Not protected. For robust system function, the application program must comply with the restriction and use the NVIC API for configuration when the SoftDevice is enabled.

Table 23 *Peripheral protection and usage by SoftDevice*

11.3 Application signals - software interrupts

Software interrupts are used by the SoftDevice to signal a change in events. ***Table 24*** shows the allocation of software interrupt vectors to SoftDevice signals.

Software interrupt (SWI)	Peripheral ID	SoftDevice Signal
0	20	Unused by the SoftDevice and available to the application.
1	21	Radio Notification - optionally configured through API.
2	22	SoftDevice Event Notification.
3	23	Reserved.
4	24	LowerStack processing - not user configurable.
5	25	UpperStack signaling - not user configurable.

Table 24 *Software interrupt allocation*

11.4 Programmable Peripheral Interconnect (PPI)

When the SoftDevice is enabled, the PPI is restricted with only some PPI channels and groups available to the application. ***Table 25*** shows how channels and groups are assigned between the application and SoftDevice.

Note: All PPI channels are available to the application when the SoftDevice is disabled.

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 - 7	Channels 0 - 15
SoftDevice	Channels 8 - 15	-

Preprogrammed channels	SoftDevice enabled	SoftDevice disabled
Application	-	Channel 20 - 31
SoftDevice	Channel 20 - 31	-

PPI group allocation	SoftDevice enabled	SoftDevice disabled
Application	Groups 0 - 1	Groups 0 - 3
SoftDevice	Groups 2 - 3	-

Table 25 *PPI channel and group availability*

11.5 SVC number ranges

Table 26 shows which SVC numbers an application program can use and which numbers are used by the SoftDevice.

Note: The SVC number allocation does not change with the state of the SoftDevice (enabled or disabled).

SVC number allocation	SoftDevice enabled	SoftDevice disabled
Application	0x00-0x0F	0x00-0x0F
SoftDevice	0x10-0xFF	0x10-0xFF

Table 26 *SVC number allocation*

12 Processor availability and interrupt latency

This chapter documents key SoftDevice performance parameters for processor availability and interrupt latency.

12.1 General performance

This section describes additional interrupt latency due to the SoC framework and gives SoftDevice interrupt latency definitions.

12.1.1 Interrupt latency due to SoC framework

Latency, additional to ARM® Cortex™-M0 hardware architecture latency, is introduced by SoftDevice logic to manage interrupt events. This latency occurs when an interrupt is forwarded to the application from the SoftDevice and is part of the minimum latency for each application interrupt. The maximum application interrupt latency is dependent on protocol stack activity as described in ***Section 12.1.2 "Processor availability"*** on page 32.

Interrupt	CPU cycles	Latency at 16 MHz
Open peripheral interrupt	50	3.2 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	63	4 µs
Application SVC interrupt	14	1 µs

Table 27 *Additional latency due to SoftDevice processing*

See ***Table 23*** on page 28 for open, blocked, and restricted peripherals.

12.1.2 Processor availability

"Appendix A: SoftDevice architecture" in the *nRF51 Reference Manual* describes interrupt management in SoftDevices and is required knowledge for understanding this section.

The SoftDevice protocol stack(s) run in the LowerStack and UpperStack interrupts. These protocol stack interrupts determine the processor availability and latencies for the interrupts/priorities available to the application - App(H), App(L), and main.

LowerStack processing will determine the processor availability and interrupt latency for App(H) (and all lower priorities), while LowerStack, App(H), and UpperStack processing together will determine the processor availability for App(L) and main. ***Figure 10*** illustrates UpperStack activity (API calls) and LowerStack activity (Protocol events) and the time reserved/not reserved for those interrupts.

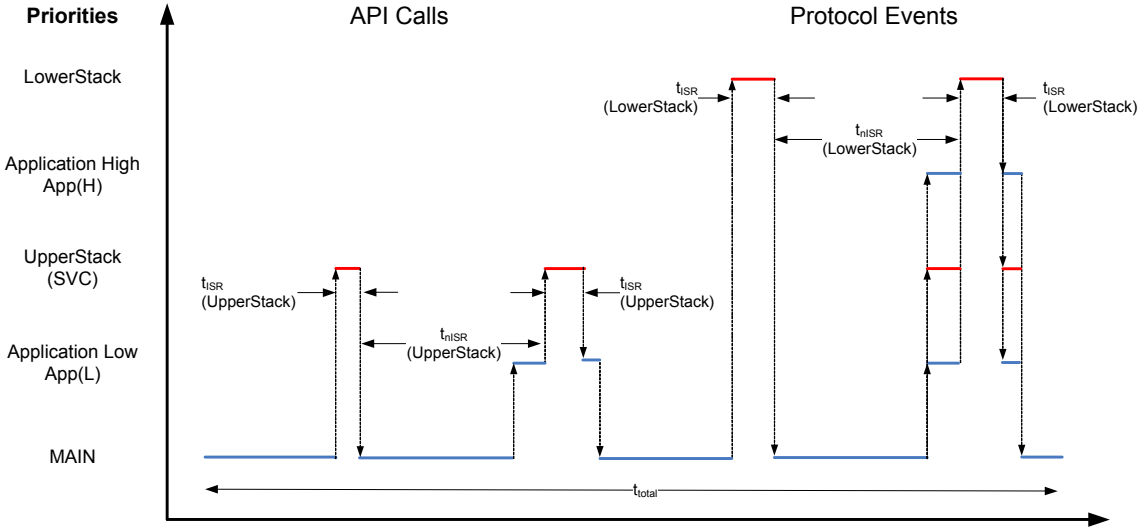

Figure 10 *UpperStack and LowerStack activity*

Table 28 describes the terms used for interrupt latency timings.

Parameter	Description
t_{ISR} (LowerStack)	Interrupt processing time in LowerStack. This is the interrupt latency for App(H) (and lower priorities).
t_{nISR} (LowerStack)	Time between LowerStack interrupts. This is the time available to run for App(H) (and lower priorities).
t_{ISR} (UpperStack)	Interrupt processing time in UpperStack. This is the interrupt latency for App(L) and processing latency for main.
t_{nISR} (UpperStack)	Time between UpperStack interrupts. This is the time available to run for App(L) and main.

Table 28 *SoftDevice interrupt latency definitions*

12.2 ANT performance

During ANT protocol events, high priority radio and stack processing activities do not extend across the duration of the event. Due to this, additional processing time is made available to allow time critical application services to be run. This is valuable in cases where frequent radio activity might be required (that is, ANT burst transfers).

12.2.1 ANT protocol event

The breakdown for a single ANT transmit or receive protocol event instance is shown in ***Figure 11***.

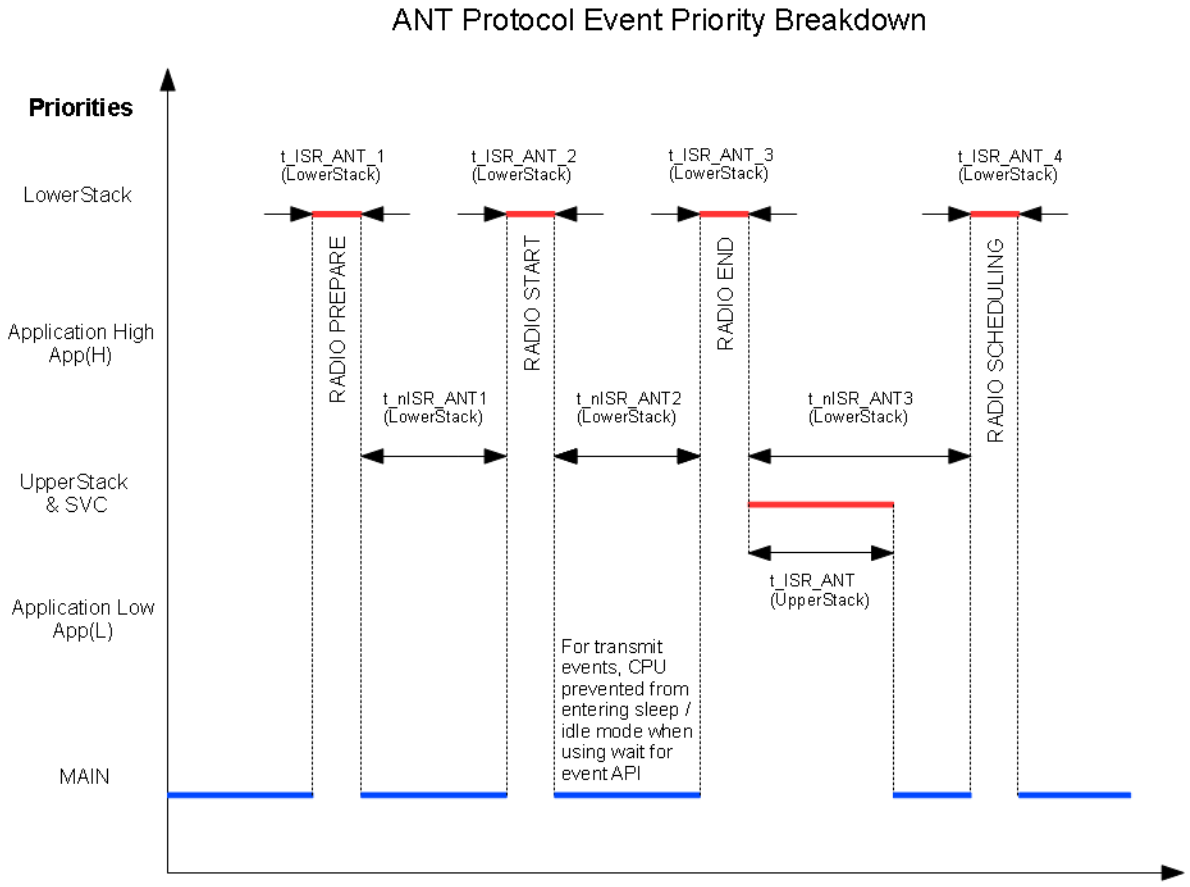

Figure 11 *ANT protocol event*

If required, application App(H) priority interrupts may be used. However, the interrupt should not exceed 100 µs every 500 µs interval or ANT performance will be adversely affected. If high application interrupts are not needed, all application processes should be run in App(L) and/or MAIN level.

In order to improve RF transmit integrity, system power changes are prevented during radio transmissions (RADIO START to RADIO END). Applications issuing the SoftDevice wait for event API call will not result in the CPU entering idle/sleep mode until the transmission activity has completed.

ANT radio and stack processing times for all supported ANT activity types in typical use cases with no high application priority interruption are summarized in ***Table 29***. High priority application interrupts will directly affect ANT STACK PROCESSING overhead time (t_UP_ISR).

Parameter	Description	Min.	Typ.	Max.
	LowerStack			
$t_{ISR\_ANT\_1}$ (LowerStack)	High priority process – RADIO PREPARE.			163 µs
$t_{nISR\_ANT\_1}$ (LowerStack)	Free processing time during RADIO PREPARE and START.	80 µs		
$t_{ISR\_ANT\_2}$ (LowerStack)	High priority process – RADIO START.		30 µs	
$t_{nISR\_ANT\_2}$ (LowerStack)	Free processing time between RADIO START and END.	251 µs		
$t_{ISR\_ANT\_3}$ (LowerStack)	High priority process – RADIO END.			68 µs
t_{nISR_ANT3} (LowerStack)	Time between high priority RADIO END and SCHEDULING process.[1]	185 µs		
$t_{ISR\_ANT\_4}$ (LowerStack)	High priority process – RADIO SCHEDULING.		88 µs	
	UpperStack			
t_{ISR_ANT} (UpperStack)	Low priority stack process.[1]			310 µs

1. High priority application processes, App(H), in these regions will delay ANT stack protocol scheduling activities. These application processes should not exceed 100 µs or else ANT operation for acknowledged and burst transfer messaging will be adversely affected.

Table 29 *SoftDevice interrupt latency LowerStack/UpperStack for an ANT protocol event*

12.2.2 ANT API calls

The the timing for API call handling in the UpperStack is described in ***Table 30***.

Parameter	Description	Min.	Typ.	Max.
t_{ISR_API} (UpperStack)	Interrupt processing time for API/SVC Call.			250 µs[1]
$t_{\_nISR\_AP1}$ (UpperStack)	Time between API interrupts.		Application dependent[2]	

1. Typical API/SVC calls have execution times less than this specified maximum value except for the following scenarios where it can be delayed up to a maximum for one channel interval:
   - Issuance of SoftDevice disable API call while one or more active ANT channels are running.
   - Issuance of ANT stack reset API call while one or more active ANT channels are running.
2. Calls to the SoftDevice API tribber the UpperStack interrupt.

Table 30 *SoftDevice interrupt latency UpperStack for ANT API calls*

12.3 BLE peripheral performance

This section describes the processor availability and interrupt latency for the BLE peripheral stack.

During BLE protocol events, LowerStack interrupts are extended by a CPU Suspend state during radio activity to improve link integrity. This means LowerStack interrupts will block application and UpperStack processing during a Radio Activity for a time proportional to the number of packets transferred during the Radio activity period.

12.3.1 BLE peripheral advertising

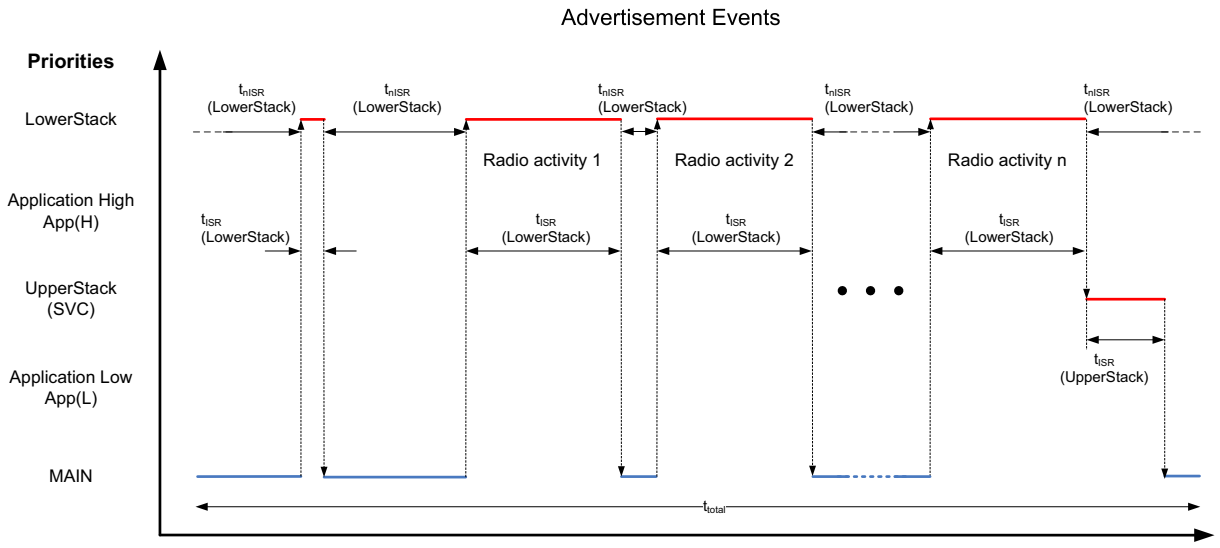

Figure 12 *Advertising*

For advertising, the pattern of LowerStack activity is as follows: there is first a Radio prepare, followed by three (or more for directed advertising) instances of Radio activity. The last Radio activity may be followed by UpperStack processing.

Parameter	Description	Nominal
t_{ISR} (LowerStack)	Maximum interrupt latency during Radio activity. Includes the time the CPU is used by the LowerStack for processing and the time the CPU is suspended during Radio activity. The longest radio activity consists of an advertisement packet with maximum data, a scan request, and a scan response with maximum data.	1700 µs
t_{nISR} (LowerStack)	Minimum time between LowerStack interrupts within an advertisement event. Time within and between advertisement events is application dependent.	150 µs

Table 31 *SoftDevice interrupt latency LowerStack for an advertising event*

12.3.2 BLE peripheral connection

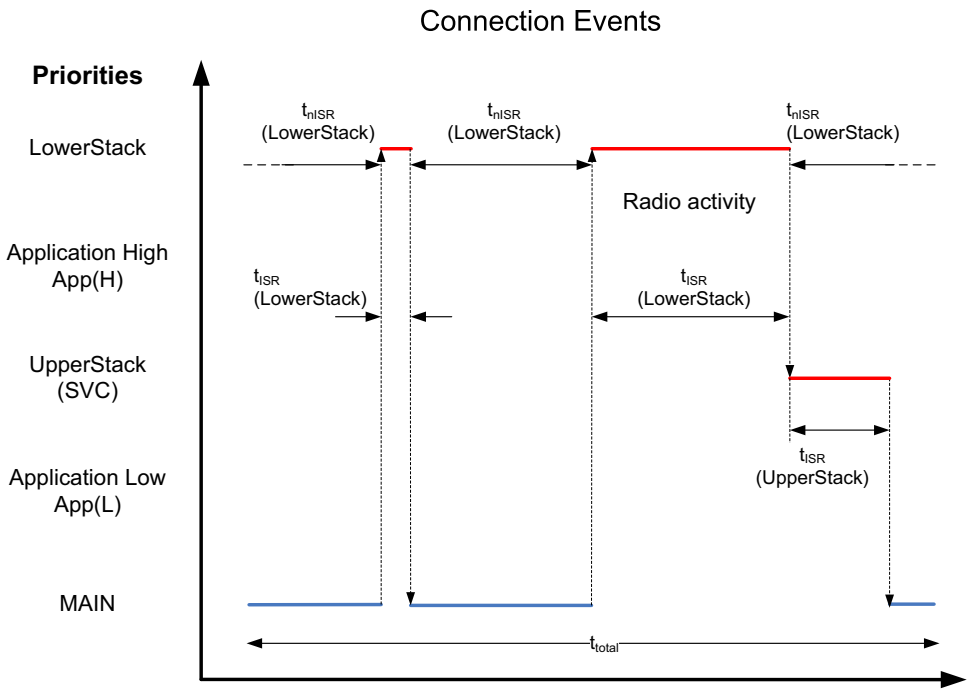

Figure 13 *Connection*

For connection events, the LowerStack activity consists of RadioPrepare followed by Radio activity. The Radio activity may be followed by UpperStack processing.

Parameter	Description	Packets	Nominal
$t_{ISR(lower\ stack)1}$	Maximum interrupt latency during Radio Event. Includes the time the CPU is used by the LowerStack for processing and the time the CPU suspended during radio activity. In each case, maximum encrypted packet length in both RX and TX are assumed.	1	1180 µs
$t_{ISR(lower\ stack)\ 2}$		2	2136 µs
$t_{ISR(lower\ stack)\ 3}$		3	3092 µs
$t_{ISR(lower\ stack)\ 4}$		4	4048 µs
$t_{ISR(lower\ stack)\ 5}$		5	5004 µs
$t_{ISR(lower\ stack)\ 6}$		6	5960 µs
$t_{nISR(lower\ stack)}$	Minimum time between LowerStack interrupts.	n/a	140 µs

Table 32 *SoftDevice interrupt latency LowerStack for a connection event*

The data in ***Table 32*** is for a connection under good conditions. Continued packet loss, clock drift, and other effects may force longer Radio activity and longer LowerStack processing. This may affect the CPU availability and interrupt latency for lower priorities.

12.3.3 API calls

The following table describes the timing for API call handling in the UpperStack.

Parameter	Description	UpperStack		
		Min	Nom	Max
$t_{ISR(upper\ stack)}$	Maximum interrupt processing time	-	-	250 µs
$t_{nISR(upper\ stack)}$	Minimum time between interrupts	Application dependent.[1]		

1. Calls to the SoftDevice API trigger the upper stack interrupt.

Table 33 *SoftDevice interrupt latency - UpperStack*

12.3.4 CPU utilization in connection

Table 34 shows expected CPU utilization percentages for the UpperStack and LowerStack given a set of typical stack connection parameters.

Note: UpperStack utilization is based only on the processing required to update the database and transfer data to and from the application when the data is transferred.

BLE connection configuration	LowerStack	UpperStack	CPU suspend	Remaining
Connection interval 4 s No data transfer	0.01%	0.01%	0.03%	~99%
Connection interval 7.5 ms 4 packet transfer per event	11%	27%	42%	~20%
Connection interval 100 ms 1 packet transfer per event (bidirectional)	0.4%	0.6%	0.7%	~98%

Table 34 *Processor usage and remaining availability for example BLE connection configurations*

12.4 Concurrent ANT and BLE peripheral performance

Running the ANT and BLE peripheral protocols concurrently may affect processor availability and interrupt latencies. The two protocols are running independently, but with shared scheduling. It is possible for an ANT protocol event to be followed by a BLE peripheral protocol event (and vice versa) in such a way that LowerStack processing in one protocol will be directly followed by LowerStack processing in the other protocol. In such a case, the App(H) interrupt latency will be higher than for a protocol event not followed by an event from the other protocol.

Whether protocol events from the two protocols will follow directly after each other is dependent upon the application, the use case, and how the timings for the two protocols are set up. In general, when running the two protocols concurrently, it should be expected that the total protocol stack resource usage (CPU, radio, and so on) will be higher compared to running only one of the protocols, with a correspondingly smaller fraction of the resources available for the application.

12.5 Performance with Flash memory API and Radio Disable API

The LowerStack interrupt is also used by the Flash memory API processing and by the Radio Disable API processing. Use of these APIs may therefore affect CPU availability and interrupt latencies for all lower priorities. The effects of this are dependent upon the application and the use case.

13 BLE data throughput

The maximum data throughput limits in ***Table 35*** apply to encrypted packet transfers. To achieve maximum data throughput, the application must exchange data at a rate that matches on-air packet transmissions and use the maximum data payload per packet.

Protocol	Role	Method	Maximum data throughput
L2CAP		Receive	140 kbps
		Send	140 kbps
		Simultaneous send and receive	130 kbps (each direction)
GATT	Client	Receive Notification	120 kbps
		Send Write command	120 kbps
		Send Write request	10 kbps
		Simultaneous receive Notification and send Write command	100 kbps (each direction)
GATT	Server	Send Notification	120 kbps
		Receive Write command	110 kbps
		Receive Write request	10 kbps
		Simultaneous send Notification and receive Write command	80 kbps (each direction)

Table 35 *L2CAP and GATT maximum data throughput*

14 ANT power profiles

This chapter provides power profiles for MCU activity during ANT radio events implemented in the SoftDevice. These profiles give a detailed overview of the stages of a radio event, the approximate timing of stages within the event, and how to calculate the peak current draw at each stage using data from the product specification. The CPU profile during the event is shown separately. Currently only the master channel and slave channel profiles are shown. Similar calculations can be extended to other ANT activity modes.

14.1 Master Channel

Radio transmit and receive event occurrence for a single ANT master channel instance is shown in ***Figure 14***.

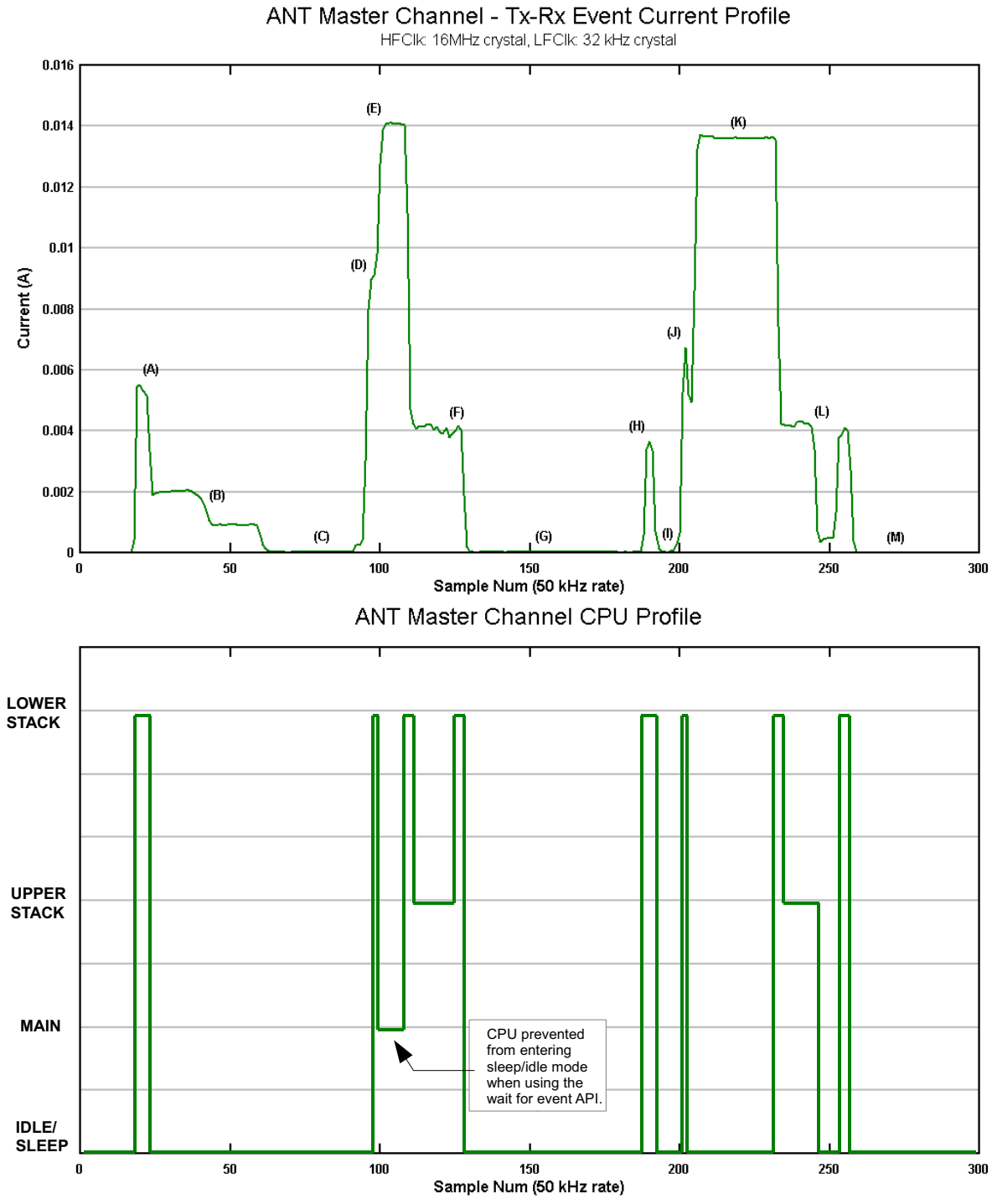

Figure 14 *ANT Master Transmit Channel*

Stage	Description	Current Calculations[1]
(A)	Preprocessing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$[2]
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{STBYX,16M}$
(D)	Radio TX Start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \int(I_{START,TX})$
(E)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{TX,0dBm} + I_{CPU,Flash}$[3]
(F)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(G)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{STBYX,16M}$
(H)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{STBYX,16M}$
(J)	Radio RX Start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \int(I_{START,RX})$
(K)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$[4]
(L)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(M)	Idle	$I_{ON} + I_{RTC} + I_{X32k}$

1. See the corresponding product specification for the symbol values.
2. Startup transients not included.
3. Application given CPU time to run processing and CPU sleeping is not allowed.
4. Application given CPU time to run processing (account for ~$I_{CPU,Flash}$ if running).

Table 36 *ANT Master Transmit Channel*

Note: When using 32.768 kHz synthesized or 32.768 kHz RC clock, replace I_{X32k} with $I_{SYNT32k}$ or I_{RC32k}, respectively.

14.2 Slave Receive Channel

A radio receive event occurrence for a single ANT slave channel instance are shown in ***Figure 15***.

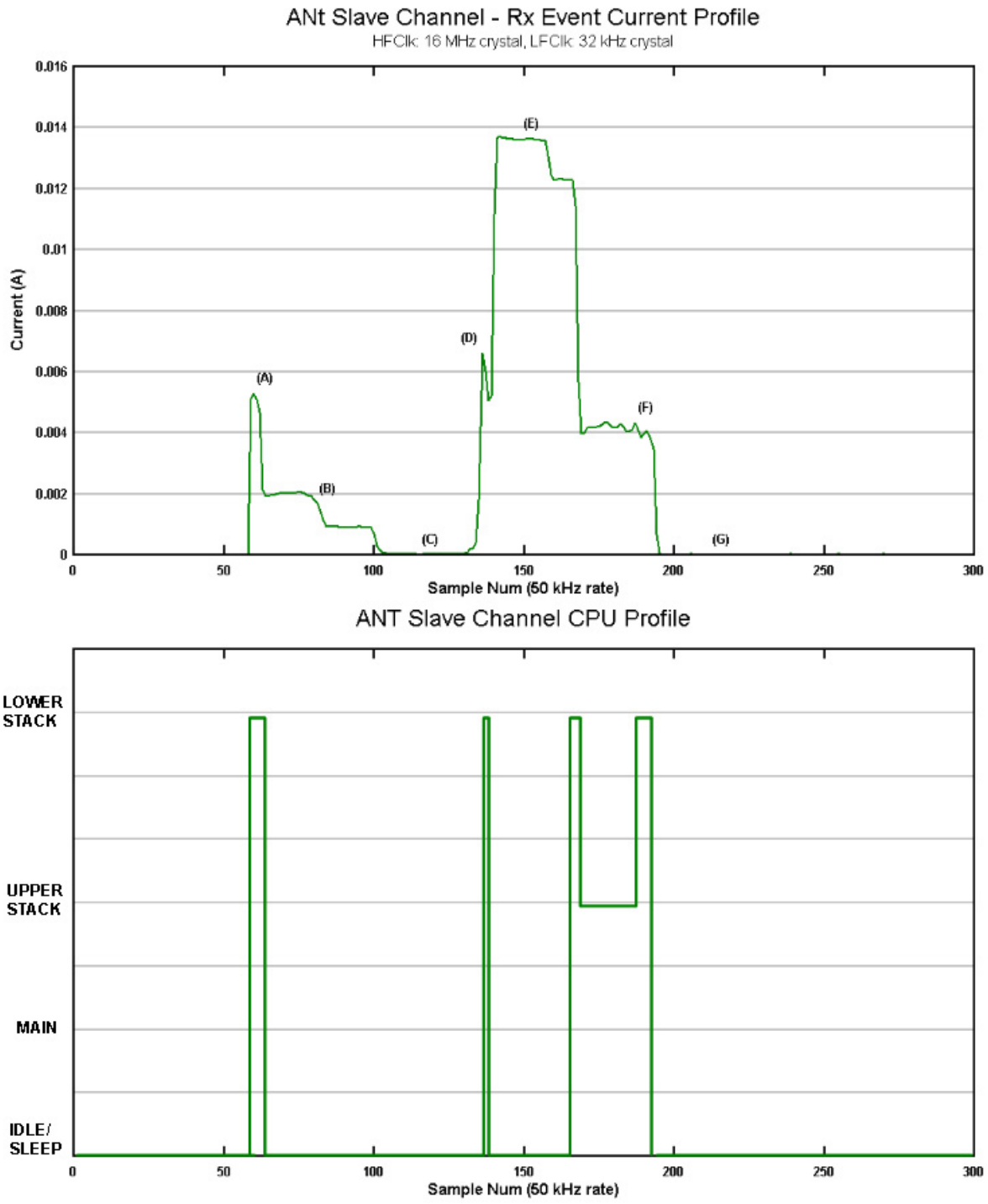

Figure 15 *ANT Slave Receive Channel*

Stage	Description	Current Calculations[1]
(A)	Preprocessing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$[2]
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{STBYX,16M}$
(D)	Radio RX Start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \int(I_{START,RX})$
(E)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$[3]
(F)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(G)	Idle	$I_{ON} + I_{RTC} + I_{X32k}$

1. See the corresponding product specification for the symbol values.
2. Startup current transients not included.
3. Application given CPU time to run processing (account for ~$I_{CPU,Flash}$ if running).

Table 37 *ANT Slave Receive Channel*

Note: When using 32 k synthesized or 32 k RC clock, replace I_{X32k} with $I_{SYNT32k}$ or I_{RC32k}, respectively.

15 BLE power profiles

This chapter provides power profiles for MCU activity during *Bluetooth* low energy Radio Events implemented in the SoftDevice. These profiles give a detailed overview of the stages of a Radio Event, the approximate timing of stages within the event, and how to calculate the peak current at each stage using data from the product specification. The LowerStack CPU profile (including CPU activity and CPU suspend) during the event is shown separately. These profiles are based on events with empty packets.

15.1 Connection event

Stage	Description	Current Calculations[1]
(A)	Preprocessing	$ION + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$ION + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$ION + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio Start	$ION + I_{RTC} + I_{X32k} + I_{X16M} + \int (I_{START,RX})$
(E)	Radio RX	$ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX} + I_{CRYPTO}$
(F)	Radio turn-around	$ION + I_{RTC} + I_{X32k} + I_{X16M} + \int (I_{START,TX})$
(G)	Radio TX	$ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM} + I_{CRYPTO}$
(H)	Post-processing	$ION + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$ION + I_{RTC} + I_{X32k}$

1. See the corresponding product specification for the symbol values.

Table 38 Connection event

Note: When using the 32.768 kHz RC oscillator, I_{RC32k} must be used instead of I_{X32k}.

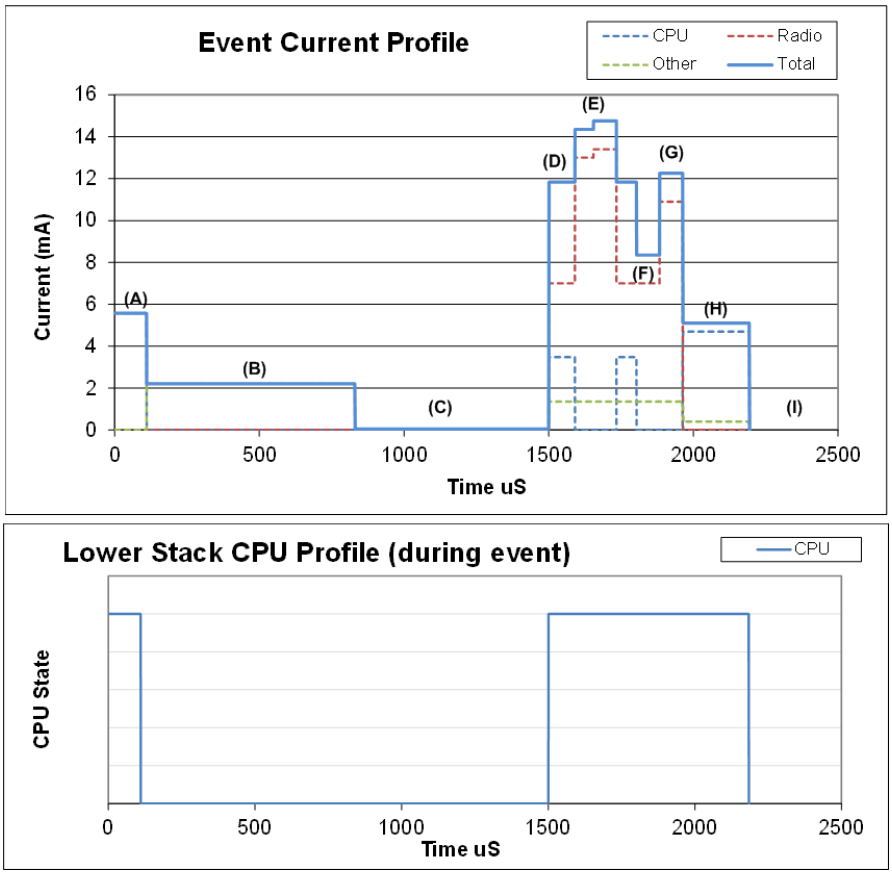

Figure 16 *Connection event*

15.2 Advertising event

Stage	Description	Current Calculation[1]
(A)	Pre-processing	ION + I_{RTC} + I_{X32k} + $I_{CPU,Flash}$
(B)	Standby + XO ramp	ION + I_{RTC} + I_{X32k} + $I_{START,X16M}$
(C)	Standby	ION + I_{RTC} + I_{X32k} + I_{X16M}
(D)	Radio start/switch	ION + I_{RTC} + I_{X32k} + I_{X16M} + ∫ ($I_{START,TX}$)
(E)	Radio TX	ION + I_{RTC} + I_{X32k} + I_{X16M} + $I_{TX,0dBM}$
(F)	Radio turn-around	ION + I_{RTC} + I_{X32k} + I_{X16M} + ∫ ($I_{START,RX}$)
(G)	Radio RX	ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}
(H)	Post-processing	ION + I_{RTC} + I_{X32k} + $I_{CPU,Flash}$
(I)	Idle	ION + I_{RTC} + I_{X32k}

1. See the corresponding product specification for the symbol values.

Table 39 Advertising event

Note: IRC32k should be substituted for I_{X32k} when using the 32.768k R_{COSC}.

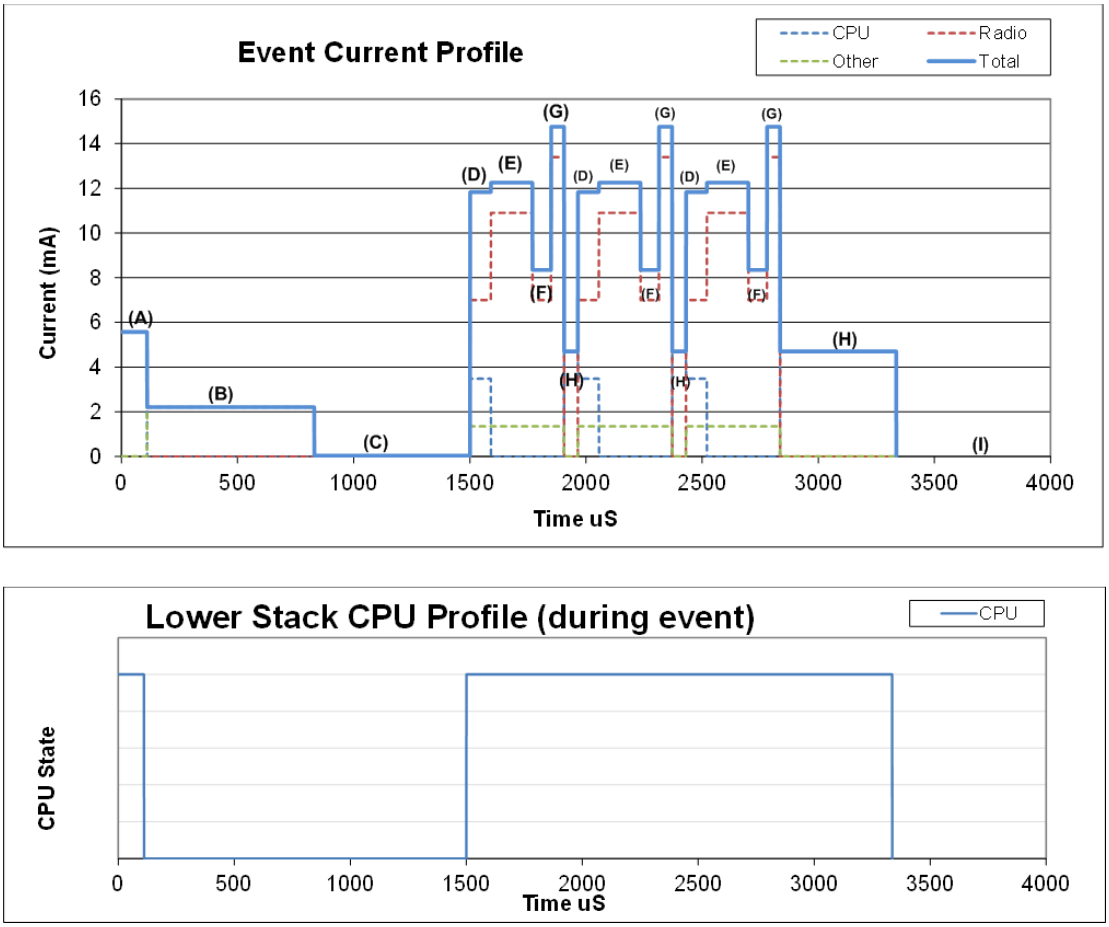

Figure 17 *Advertising event*

16 ANT/BLE concurrency handling

The SoftDevice internally manages time multiplexed on-air transactions for the ANT and BLE protocols to operate concurrently.

Both ANT and BLE protocols use time synchronization between peer devices when sending and receiving packets. It is possible that both may request to access the Radio for an on-air transaction at the same time resulting in a collision. In general, the protocols will not request access to the Radio at the same time, the probability of packet loss due to collisions will be low, and access to the Radio will be fairly granted between protocols in the case of a collision. Short connection intervals (less than 20 ms) or high burst Radio utilization by one or both stacks will increase the rate of collisions. This will be application dependent.

In general, there should be no requirement from the application to specify configurations to operate the protocols concurrently.

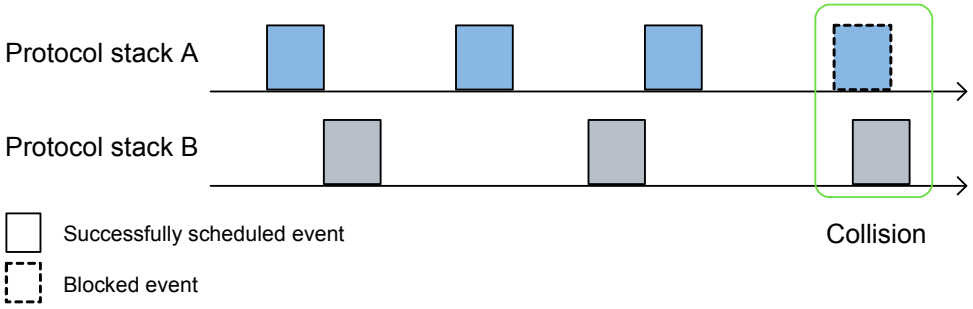

Figure 18 *Collision*

16.1 ANT

ANT provides a set of APIs to help manage ANT radio coexistence scheduling. By default, these settings should not be changed unless very specific coexistence behavior is required. The following rules describe the default configuration.

- On a per channel basis, if the master synchronous TX channel blockage time (calculated by the number of times consecutively blocked multiplied by the channel interval) exceeds 450 ms, the next synchronous TX activity will be performed at elevated priority. For example, a 4 Hz master channel that experiences two consecutive blockages results in the next transmission to be raised in priority (2 x 250 ms > 450 ms). For a 10 Hz channel, five consecutive blocks are required for the next transmission to be raised in priority (5 x 100 ms > 450 ms).
- On a per channel basis, if the slave synchronous RX channel missed RX event time (calculated by the number of consecutive RX fail events multiplied by the channel interval) exceeds 450 ms, the next synchronous RX activity will be performed at elevated priority. The previously mentioned examples (in the point above) can be similarly applied here.
- For channels that are searching, every 4th RX search window is raised to elevated priority in order to provide predictable and minimal acquisition performance based on the default searching rate. The coexistence configuration has been selected to provide minimal intrusion in concurrent BLE activities and scheduled flash writing/erasing operations.
- During burst transfers, when two or more TX blockages and/or missed RX activities are detected, the priority of the next burst packet transaction is elevated until it is successful.

16.2 BLE

To ensure that the BLE connections can be established quickly and that established connections will not be lost due to scheduling collisions, the SoftDevice guarantees the following behavior:

- A maximum of two consecutive BLE advertisement events may be dropped due to collisions; at least every third BLE advertisement event will succeed.
- A maximum of three consecutive BLE connection events may be dropped; at least every forth connection event will succeed.
- Packets transferred during BLE connection setup and connection parameter update will not be dropped due to a scheduling collision. The BLE connection will be subject to the second rule only after the connection is established by successful transfer of data packets.

17 SoftDevice identification and revision scheme

The SoftDevices will be identified by the SoftDevice part code, a qualified IC partcode (for example, nRF51822), and a version string.

For revisions of the SoftDevice which are production qualified, the version string consists of major, minor, and revision numbers only, as described in ***Table 40***.

For revisions of the SoftDevice which are not production qualified, a build number and a test qualification level (alpha/beta) are appended to the version string.

For example: s110_nrf51822_1.2.3-4.alpha, where major = 1, minor = 2, revision = 3, build number = 4 and test qualification level is alpha. Additional SoftDevice revision examples are given in ***Table 41***.

Revision	Description
Major increments	Modifications to the API or the function or behavior of the implementation or part of it have changed. Changes as per Minor Increment may have been made. Application code will not be compatible without some modification.
Minor increments	Additional features and/or API calls are available. Changes as per Revision Increment may have been made. Application code may have to be modified to take advantage of new features.
Revision increments	Issues have been resolved or improvements to performance implemented. Existing application code will not require any modification.
Build number increment (if present)	New build of non-production version.

Table 40 *Revision scheme*

Sequence number	Description
s110_nrf51822_1.2.3-1.alpha	Revision 1.2.3, first build, qualified at alpha level
s110_nrf51822_1.2.3-2.alpha	Revision 1.2.3, second build, qualified at alpha level
s110_nrf51822_1.2.3-5.beta	Revision 1.2.3, fifth build, qualified at beta level
s110_nrf51822_1.2.3	Revision 1.2.3, qualified at production level

Table 41 *SoftDevice revision examples*

The test qualification levels are outlined in ***Table 42.***

Qualification	Description
Alpha	Development release suitable for prototype application development. Hardware integration testing is not complete. Known issues may not be fixed between alpha releases. Incomplete and subject to change.
Beta	Development release suitable for application development. In addition to alpha qualification: Hardware integration testing is complete but may not be feature complete and may contain known issues. Protocol implementations are tested for conformance and interoperability.
Production	Qualified release suitable for product integration. In addition to beta qualification: Hardware integration tested over supported range of operating conditions. Stable and complete with no known issues. Protocol implementations conform to standards.

Table 42 *Test qualification levels*

17.1 Notification of SoftDevice revision updates

When new versions of a SoftDevice become available or the qualification status of a given revision of a SoftDevice is changed, product update notifications will be automatically forwarded, by email, to all users who have a profile configured to receive notifications from the Nordic Semiconductor website.

The SoftDevice will be updated with additional features and/or fixed issues if needed. Supported production versions of the SoftDevice will remain available after updates, so products do not need requalification on release of updates if the previous version is sufficiently feature complete for your product.