S210 nRF51422

ANT™ SoftDevice

SoftDevice Specification v2.0

Key Features

- Simple to complex network topologies:
  - Peer-to-peer, Star, Tree, Star-to-star and more
- Up to 8 logical channels, each with configurable:
  - Channel type, ID and period
  - RF frequency
  - Networks
- Broadcast, Acknowledged, and Burst Data modes
- Device search, pairing and proximity support
- Enhanced ANT features:
  - Advanced Burst Transfer mode (up to 60 kbps)
  - Single channel encryption (AES-128) support
  - Additional networks – up to 8
  - Event Filtering and Selective Data Updates
  - Asynchronous Transmission
  - Fast Channel Initiation
- Built-in NVM access and radio coexistence management
- Master Boot Record for over-the air device firmware update
- Memory isolation between application and protocol stack for robustness and security
- Thread-safe supervisor-call based API
- Asynchronous, event-driven behavior
- No RTOS dependency
  - Any RTOS can be used
- No link-time dependencies
  - Standard ARM® Cortex™ -M0 project configuration for application development
- Support for concurrent and non-concurrent multiprotocol operation
  - Concurrent multiprotocol timeslot API
  - Alternate protocol stack running in application space

Applications

- Personal area networks:
  - Sport and fitness sensors
  - Monitoring devices
  - Healthcare and lifestyle sensors
  - Key fobs, remote controls
- Environment sensor networks
- High density networking and monitoring
- Logistics and goods tracking
- Smart RF tags

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
September 26th, 2014	2.0	**Updated:** • Frontpage • ***Chapter 1 "Introduction"*** on page 5 • ***Section 2.2 "Multiprotocol support"*** on page 6 • ***Section 3.2 "ANT profile and feature support"*** on page 8 • ***Chapter 4 "SoC library"*** on page 10 • ***Chapter 6 "Flash memory API"*** on page 12 • ***Chapter 7 "Radio Notification"*** on page 13 • ***Chapter 9 "Master Boot Record and Bootloader"*** on page 26 • ***Chapter 10 "SoC resource requirements"*** on page 29 • ***Chapter 11 "Processor availability and interrupt latency"*** on page 34 • ***Chapter 12 "ANT power profiles"*** on page 39 • ***Chapter 13 "SoftDevice identification and revision scheme"*** on page 44 **Added:** • ***Chapter 8 "Concurrent Multiprotocol Timeslot API"*** on page 15
January 2014	1.2	**Updated:** • ***Table 1 "System on Chip features"*** on page 10 **Removed:** • ***Figure 4 "Radio Notification"*** on page 14 • ***Figure 5 "Radio Notification, multiple packet transfers"*** on page 15 • ***Figure 6 "Consecutive Radio Events with Radio Notification"*** on page 17 • ***Table 6 "Radio Notification figure labels"*** on page 16 • ***Table 7 "Radio Notification timing ranges"*** on page 16

Date	Version	Description
December 2013	1.1	**Updated:** • ***Figure 10 "Interrupt latencies due to SoftDevice processing"*** on page 32 • ***Figure 12 "ANT Master Transmit Channel"*** on page 40 • ***Figure 13 "ANT Slave Receive Channel"*** on page 42 • ***Table 10 "S210 Memory resource requirements"*** on page 30 • ***Table 20 "LowerStack interrupt latency numbers"*** on page 34 • ***Table 21 "UpperStack interrupt latency numbers"*** on page 35 **Added:** • ***Chapter 6 "Flash memory API"*** on page 12 • ***Chapter 7 "Radio Notification"*** on page 13 • ***Chapter 8 "Bootloader"*** on page 26
December 2012	1.0	**Updated:** • ***Figure 2 "ANT stack architecture"*** on page 7 • ***Figure 10 "Interrupt latencies due to SoftDevice processing"*** on page 32 • ***Section 3.2 "ANT profile and feature support"*** on page 8 • ***Table 10 "S210 Memory resource requirements"*** on page 30 **Added:** • ***Table 20 "LowerStack interrupt latency numbers"*** on page 34 • ***Table 21 "UpperStack interrupt latency numbers"*** on page 35
October 2012	0.6.1	Updated flash size of the S210 SoftDevice to 40 kB.
September 2012	0.6	Preliminary release.

1 Introduction

The S210 SoftDevice is an ANT protocol stack solution that provides a full and flexible Application Programming Interface (API) for building ANT System on Chip (SoC) solutions for the nRF51422 chip. The S210 SoftDevice simplifies combining the ANT protocol stack and an application on the same CPU.

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 13.2 "Notification of SoftDevice revision updates"*** on page 45 for more information. This specification outlines the supported features of a production level SoftDevice. Alpha and beta versions of the SoftDevice may not support all features. To find information on any limitations or omissions, see the SoftDevice release notes, which will contain a detailed summary of the release status.

1.1 Documentation

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

Document	Description
nWP-20, SoftDevice Architecture Overview	White paper that 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.
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.

2 Product overview

This section provides an overview of the SoftDevice.

2.1 SoftDevice

The S210 SoftDevice is a precompiled and linked binary software implementing a full ANT protocol stack for 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+ applications with the SoftDevice.

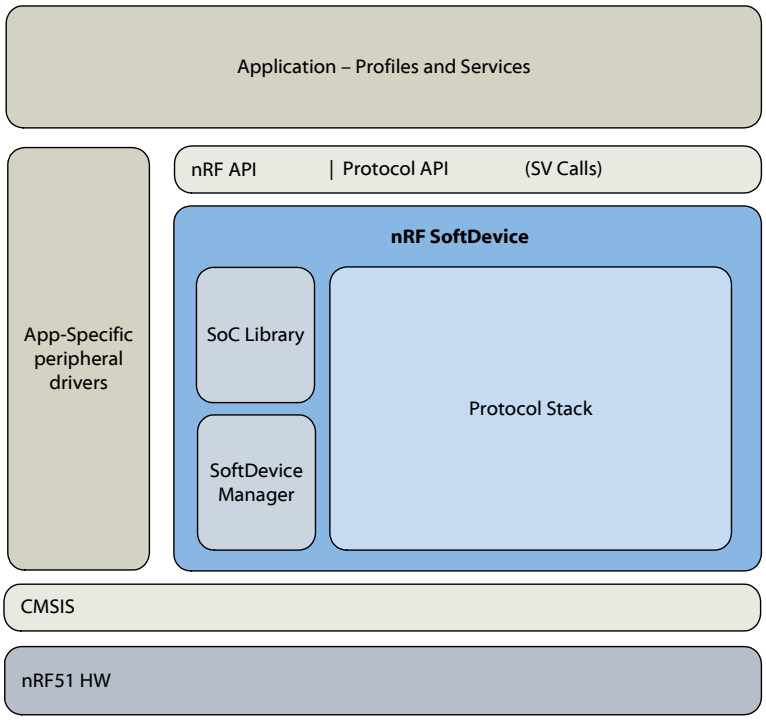

Figure 1 *System on Chip application with the SoftDevice*

The SoftDevice can be programmed onto compatible nRF51 Series chips during both development and production.

2.2 Multiprotocol support

The SoftDevice supports both non-concurrent and fully concurrent multiprotocol implementations. For non-concurrent operation, 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. For concurrent multiprotocol operation, with a proprietary protocol running concurrently with the SoftDevice protocol, see ***Chapter 8 "Concurrent Multiprotocol Timeslot API"*** on page 15.

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.

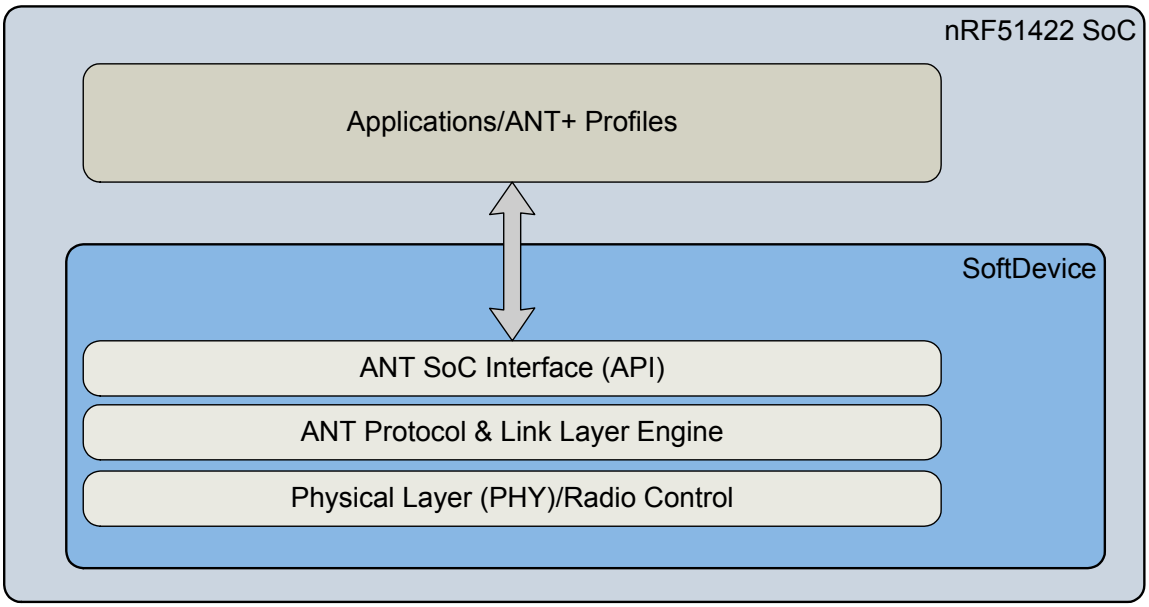

Figure 2 *ANT stack architecture*

3.2 ANT profile and feature support

The S210 SoftDevice supports all applicable ANT commands described in the *nRF24AP2 Product specification* (Table 4. ANT message summary supported by nRF24AP2). Profiles and features available in the nRF24AP1 and nRF24AP2 are supported in the S210 SoftDevice. In addition, the SoftDevice includes the following enhanced ANT features 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 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 the 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, otherwise the display units are 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

Asynchronous Transmission 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 skip the search window check, and thereby reduce the latency between channel opening and transmission. A device that skips the search window check increases the likelihood of interference with other transmitting devices and should therefore only be used in environments with few transmitting devices and scenarios where the device opens for a brief period of time and where low latency is required.

4 SoC library

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

Feature	Description
Mutex	The SoftDevice implements atomic mutex acquire and release operations that are safe for the application to use. Use this mutex to avoid disabling global interrupts in the application, because disabling global interrupts will interfere with the SoftDevice and may lead to dropped packets or lost connections.
NVIC	Gives the application access to all NVIC features without corrupting SoftDevice configurations.
Rand	Provides random numbers from the hardware random number generator.

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 call 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.
Concurrent Multiprotocol Timeslot API	Schedule other radio protocol activity, or periods of radio inactivity.
Radio notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See ***Chapter 7 "Radio Notification"*** on page 13.
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 be safely used during all protocol stack states.
Temperature	Application access to the temperature sensor.
Master Boot Record	The MBR provides support for bootloader implementation and firmware update functions.

Table 1 *System on Chip features*

5 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 can be selected between the following options: • RC oscillator • Synthesized from high frequency clock • Crystal oscillator

***Table 2** SoftDevice Manager*

6 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.

The flash memory access is scheduled between the protocol radio events. For certain memory access operations, the time required may be longer than the time between radio events. In this case, the radio event may be skipped or the flash memory access may be delayed. If the flash operation requires more time than allowed to run concurrently with certain ANT activities, the flash memory access may fail and generate a timeout event: NRF_EVT_FLASH_OPERATION_ERROR. In this case, retry the flash operation.

ANT activity	Flash write
ANT RX Scanning Channel ANT RX Search/Background Search ANT TX/RX Broadcast Messaging ANT TX/RX Acknowledged Messaging	• Support full flash write size (256 words). • Flash timeout event may be generated if critical ANT radio activities need to occur. • ANT transmit/receive performance may be impacted if continuous flash write operations are requested.
ANT TX/RX Burst Transfer	• Maximum recommended flash write size (64 words). Larger flash writes may result in flash timeout events or burst transfer failures. • Continuous flash write activity may result in burst transfers taking up to 2-3 times longer to complete.
ANT activity	**Flash erase**
ANT RX Scanning Channel ANT RX Search ANT TX/RX Broadcast Messaging ANT TX/RX Acknowledged Messaging	• Support flash erase attempts. • Flash timeout event may be generated if critical radio activities need to occur. • ANT transmit/receive performance may be impacted if continuous flash erase operations are requested.
ANT TX/RX Burst Transfer	• Flash erase not supported. Flash erase attempts may result in flash timeout event or burst transfer failures.

Table 3 *Behavior with ANT traffic and concurrent flash write/erase*

7 Radio Notification

Radio Notification is a configurable feature that enables ACTIVE and INACTIVE (nACTIVE) signals from the SoftDevice that notify the application when the radio is in use. The signal is sent using software interrupt, as specified in ***Table 13*** on page 32.

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 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.

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 3*** 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.

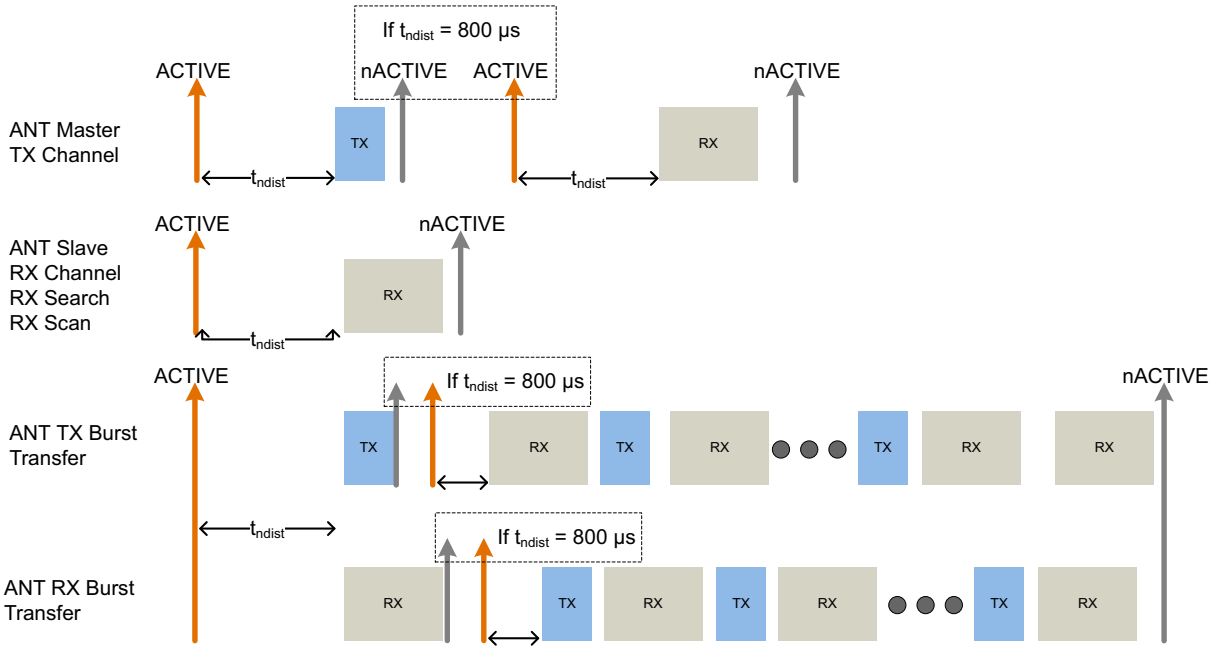

Figure 3 *Radio Notifications*

Table 4 lists the recommended t_{ndist} values for ANT activities in combination with other scheduler requested activities (for example, flash access/concurrent multiprotocol timeslot usage).

	Concurrent operations			
ANT activity	**None**	**Flash write/erase**	**Concurrent multiprotocol timeslot scheduling**	**Flash write/erase + Concurrent multiprotocol timeslot scheduling**
All	t_{ndist} • 800 µs • 1740 µs • 2680 µs • 3620 µs • 4520 µs • 5500 µs	t_{ndist} • 800 µs	t_{ndist} • 800 µs	t_{ndist} • 800 µs

Table 4 *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.

8 Concurrent Multiprotocol Timeslot API

The Multiprotocol Timeslot API allows an application developer to safely schedule 2.4 GHz proprietary radio usage while the SoftDevice protocol stack is in use by the device. This allows the nRF51 device to be part of a network using the SoftDevice protocol stack and an alternative network of wireless devices at the same time.

The Timeslot feature gives the application access to the radio and other restricted peripherals, which it does by queueing the application's use of these peripherals with those of the SoftDevice. Using this feature, the application can run other radio protocols (third party custom or proprietary protocols running from application space) concurrently with the internal protocol stack(s) of the SoftDevice. It can also be used to suppress SoftDevice radio activity and reserve guaranteed time for application activities with hard timing requirements which cannot be met by using the SoC Radio Notifications.

The Timeslot feature is part of the SoC library. The feature works by having the SoftDevice time-multiplex access to peripherals between the application and itself. Through the SoC API, the application can open a Timeslot session and request timeslots. When a timeslot is granted, the application has exclusive and real-time access to the normally blocked RADIO, TIMER0, CCM, AAR, and PPI (channels 8 – 15) peripherals and can use these freely for the length of the timeslot, see ***Table 11 "Hardware access type definitions"*** on page 30 and ***Table 12 "Peripheral protection and usage by SoftDevice"*** on page 31.

8.1 Request types

Timeslots may be requested as *earliest possible*, in which case the timeslot occurs at the first available opportunity. In the request, the application can limit how far into the future the timeslot may be placed. Timeslots may also be requested at a given time. In this case, the application specifies in the request when the timeslot should start and the time is measured from the start of the previous timeslot. Note that the first request in a session must always be *earliest possible* to create the timing reference point for later timeslots. The application may also request to extend an on-going timeslot. Extension requests may be repeated, prolonging the timeslot even further.

Timeslots requested as *earliest possible* are useful for single timeslots and for non-periodic or non-timed activity. Timeslots requested at a given time relative to the previous timeslot are useful for periodic and timed activities; for example, a periodic proprietary radio protocol. Timeslot extension may be used to secure as much continuous radio time as possible for the application; for example, running an "always on" radio listener.

8.2 Request priorities

Timeslots can be requested at either high or normal priority, indicating how important it is for the application to access the specified peripheral. Using normal priority should be considered best practice to minimize the influence of the use of the Multiprotocol Timeslot API on other activities. The high priority should only be used when required, such as for running a radio protocol with certain timing requirements that are not met using normal priority.

8.3 Timeslot length

The length of the timeslot is specified by the application in the request and ranges from 100 µs to 100 ms. Longer continuous timeslots can be achieved by requesting to extend the current timeslot. Successive extensions will give a timeslot as long as possible within the limits set by other SoftDevice activities, up to a maximum of 128 s.

8.4 Scheduling

Timeslots requested by the application are scheduled within the SoftDevice along with the SoftDevice protocol and the Flash API activities.

Whether the timeslot request is granted and access to the peripherals given is based on when the request was made, when the timeslot is wanted, the priority of the request, and the requested length of the timeslot. If the requested timeslot does not collide with other activities, the request will be granted and the timeslot scheduled. If the requested timeslot collides with an already scheduled activity with equal or higher priority, the request will be blocked. If a later arriving activity of higher priority causes a collision, the request will be canceled and the scheduled timeslot revoked. However, a timeslot that has already started cannot be interrupted or canceled. Timeslots requested at high priority will cancel other activities scheduled at lower priorities in case of a collision. Also, requests for short timeslots have a higher probability of succeeding than requests for longer timeslots because shorter timeslots are easier to fit into the schedule.

Note: Radio Notification signals behave the same way for timeslots requested through the Multiprotocol Timeslot interface as for SoftDevice internal activities, see ***Chapter 7 "Radio Notification"*** on page 13 for more information. If Radio Notifications are enabled, Multiprotocol Timeslots will be notified.

8.5 Performance considerations

Since the Multiprotocol Timeslot API shares core peripherals with the SoftDevice, and are scheduled along with other SoftDevice activities, use of the Timeslot feature may influence SoftDevice performance. Therefore the application configuration of the SoftDevice protocol should be considered when using the Multiprotocol Timeslot API.

In general, all timeslot requests should use the lowest priority to ensure that interruptions to other activity is minimized. In addition, timeslots should be kept as short as possible in order to minimize the impact on the overall performance of the device. Similarly, requesting a shorter timeslot and then extending it gives more flexibility to schedule other activities than requesting a longer timeslot.

For certain radio protocol activities, the use of high priority may be required to allow the requested timeslot to run concurrently and not be consistently blocked.

Table 5 on page 17 lists the maximum recommended low and high priority timeslot reservation requests that can be made during ANT activities. For low priority timeslot requests, exceeding the maximum recommended timeslot may result in the timeslot request always being blocked or canceled by higher

priority ANT radio activities. For high priority timeslot requests, exceeding the recommended values could result in severe ANT activity degradation.

ANT activity	Maximum recommended timeslot			
	Low priority		High priority	
	One-time	Continuous/periodic	One-time	Continuous/periodic
ANT RX Search / Background Search	Less than Search Interval (default < 5 ms)	Less than Search Interval (default < 5 ms)	Up to 100 ms	Can support up to 100 ms at the cost of ANT search efficiency
ANT RX Scanning Channel	Not supported, always blocked by ANT	Not supported, always blocked by ANT	Up to 100 ms	Can support up to 100 ms at the cost of ANT scan efficiency
ANT TX/RX Broadcast and Acknowledged Messaging	Less than ANT channel period	Less than ANT channel period	Up to 100 ms	Less than ANT channel period
ANT TX/RX Burst Transfer	Not supported, always blocked by ANT	Not supported, always blocked by ANT	Up to 5 ms	Up to 5 ms. Can result in burst transfers taking 2-3 times longer to complete

Table 5 *ANT performance considerations*

8.6 Multiprotocol timeslot API

A Timeslot session is opened and closed using API calls. Within a session, there is an API call to request timeslots. For communication back to the application the feature will generate events, which are handled by the normal application event handler, and signals, which must be handled by a callback function (the signal handler) provided by the application. The signal handler can also return actions to the SoftDevice. Within a timeslot, only the signal handler is used.

Note: The API calls, events, and signals are only given by their full names in the tables where they are listed the first time. Elsewhere, only the last part of the name is used.

8.6.1 API calls

The following API calls are defined:

API call	Description
sd_radio_session_open()	Open a timeslot session.
sd_radio_session_close()	Close a timeslot session.
sd_radio_request()	Request a timeslot.

Table 6 *API calls*

8.6.2 Timeslot events

Events come from the SoftDevice scheduler and are used for timeslot session management. Events are received in the application event handler callback function, which will typically be run in App(L) priority, see ***Section 11.3 "ANT performance"*** on page 36.

The following events are defined:

Event	Description
NRF_EVT_RADIO_SESSION_IDLE	Session status: The current timeslot session has no remaining scheduled timeslots.
NRF_EVT_RADIO_SESSION_CLOSED	Session status: The timeslot session is closed and all acquired resources are released.
NRF_EVT_RADIO_BLOCKED	Timeslot status: The last requested timeslot could not be scheduled, due to a collision with already scheduled activity or for other reasons.
NRF_EVT_RADIO_CANCELED	Timeslot status: The scheduled timeslot was preempted by higher priority activity.
NRF_EVT_RADIO_SIGNAL_ CALLBACK_INVALID_RETURN	Signal handler: The last signal hander return value contained invalid parameters.

Table 7 *Timeslot events*

8.6.3 Timeslot signals

Signals come from the peripherals and arrive within a timeslot. Signals are received in a signal handler callback function that the application must provide. The signal handler runs in LowerStack priority, which is the highest priority in the system, see ***Section 11.3 "ANT performance"*** on page 36.

Signal	Description
NRF_RADIO_CALLBACK_SIGNAL_TYPE_START	Start of the timeslot. The application now has exclusive access to the peripherals for the full length of the timeslot.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_RADIO	Radio interrupt, for more information, see chapter *2.4 GHz radio (RADIO)* in the *nRF51 Reference Manual*.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_TIMER0	Timer interrupt, for more information, see chapter *Timer/counter (TIMER)* in the *nRF51 Reference Manual*.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_SUCCEEDED	The latest extend action succeeded.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_FAILED	The latest extend action failed.

Table 8 *Timeslot signals*

8.6.4 Signal handler return actions

The return value from the application signal handler to the SoftDevice contains an action. The signal handler action return values are:

Return value	Description
NRF_RADIO_SIGNAL_CALLBACK_ACTION_NONE	The timeslot processing is not complete. The SoftDevice will take no action.
NRF_RADIO_SIGNAL_CALLBACK_ACTION_END	The current timeslot has ended. The SoftDevice can now resume other activities.
NRF_RADIO_SIGNAL_CALLBACK_ACTION_REQUEST_AND_END	The current timeslot has ended. The SoftDevice is requested to schedule a new timeslot, after which it can resume other activities.
NRF_RADIO_SIGNAL_CALLBACK_ACTION_EXTEND	The SoftDevice is requested to extend the ongoing timeslot.

Table 9 *Signal handler action return values*

8.6.5 Ending a timeslot in time

The application is responsible for keeping track of timing within the timeslot and ensuring that the application's use of the peripherals does not last for longer than the granted timeslot. For these purposes, the application is granted access to the TIMER0 peripheral for the length of the timeslot. This timer is started from zero by the SoftDevice at the start of the timeslot, and is configured to run at 1 MHz. The recommended practice is to set up a timer interrupt that expires before the timeslot expires, with enough time left of the timeslot to do any clean-up actions before the timeslot ends. Such a timer interrupt can also be used to request an extension of the timeslot, but there must still be enough time to clean up if the extension is not granted.

8.6.6 The signal handler runs at LowerStack priority

The signal handler runs at LowerStack priority, which is the highest priority. Therefore, it cannot be interrupted by any other activity. Also, as for the App(H) interrupt, SVC calls are not available in the signal handler. It is a requirement that processing in the signal handler does not exceed the granted time of the timeslot. If it does, the behavior of the SoftDevice is undefined and the SoftDevice may malfunction.

The signal handler may be called several times during a timeslot. It is recommended to use the signal handler only for the real time signal handling. When a signal has been handled, exit the signal handler to wait for the next signal. Processing other than signal handling should be run at lower priorities, outside of the signal handler.

8.7 Timeslot usage examples

In this section we provide several timeslot usage examples and describe the sequence of events within them.

8.7.1 Complete session example

Figure 4 shows a complete timeslot session. In this case, only timeslot requests from the application are being scheduled, there is no SoftDevice activity.

At start, the application calls the API to open a session and to request a first timeslot (which must be of type *earliest*). The SoftDevice schedules the timeslot. At the start of the timeslot, the SoftDevice calls the application signal hander with the START signal. After this, the application is in control and has access to the peripherals. The application will then typically set up TIMER0 to expire before the end of the timeslot, to get a signal that the timeslot is about to end. In the last signal in the timeslot, the application uses the signal handler return action to request a new timeslot 100 ms after the first.

The following timeslots (the middle timeslot in ***Figure 4***) are all similar. The signal handler is called with the START signal at the start of the timeslot. The application then has control, but must arrange for a signal to come towards the end of the timeslot. As the return value for the last signal in the timeslot, the signal handler requests a new timeslot using the REQUEST_AND_END action.

Eventually, the application does not require the radio any more. So, at the last signal in the last timeslot, the application returns END from the signal handler. The SoftDevice then sends an IDLE event to the application event handler. The application calls *session_close*, and the SoftDevice sends the CLOSED event. The session has now ended.

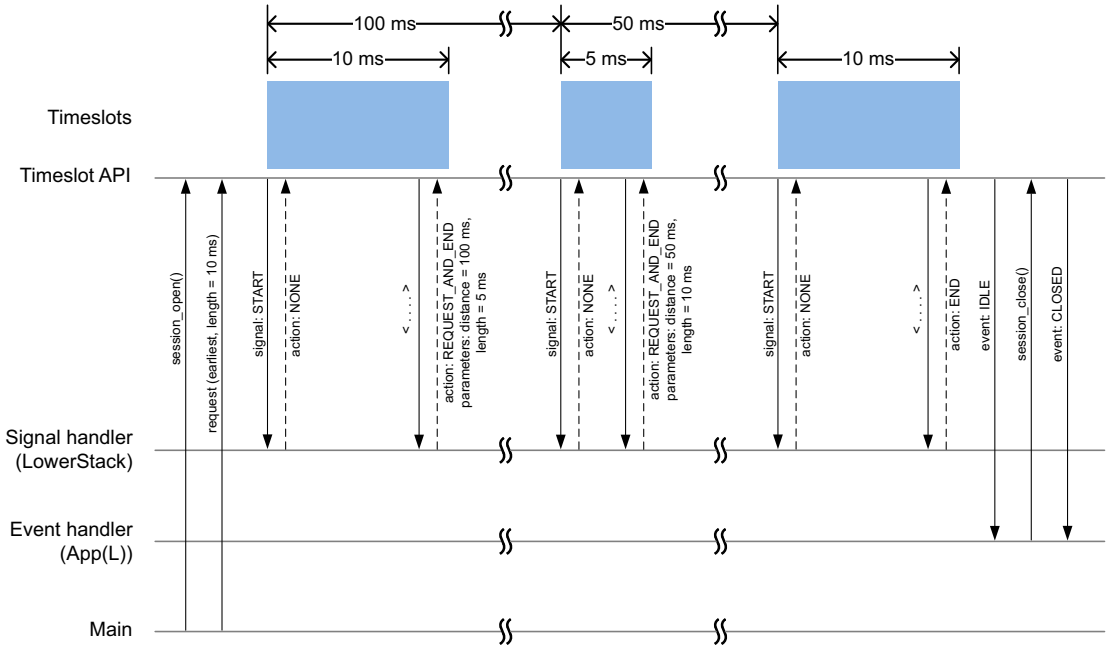

Figure 4 *Complete session example*

8.7.2 Blocked timeslot example

Figure 5 shows a situation in the middle of a session where a requested timeslot cannot be scheduled. At the end of the first timeslot in ***Figure 5***, the application signal handler returns a REQUEST_AND_END action to request a new timeslot. The new timeslot cannot be scheduled as requested, because of a collision with an already scheduled SoftDevice activity. The application is notified about this by a BLOCKED event to the application event handler. The application then makes a new request further out in time. This request succeeds (it does not collide with anything), and a new timeslot is scheduled.

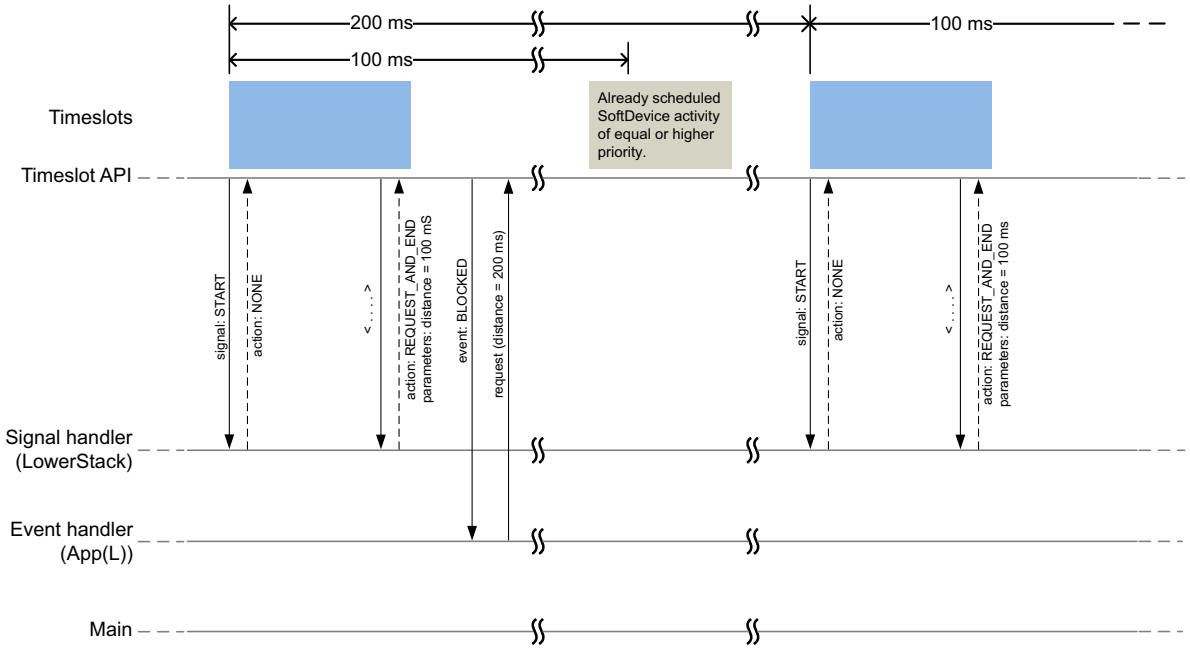

Figure 5 *Blocked timeslot example*

8.7.3 Canceled timeslot example

Figure 6 on page 24 shows a situation in the middle of a session where a requested and scheduled application timeslot is being revoked. The upper part of ***Figure 6*** on page 24 shows that the application has ended a timeslot by returning the REQUEST_AND_END action, and the new timeslot has been scheduled. The new scheduled timeslot has not been started yet, it is still some time into the future. The lower part of ***Figure 6*** on page 24 shows the situation some time later. In the meantime, time for a SoftDevice activity of higher priority has been requested internally in the SoftDevice, at a time which collides with the scheduled application timeslot. To accommodate the higher priority request, the application timeslot has been removed from the schedule, and the higher priority SoftDevice activity scheduled instead. The application is notified about this by a CANCELED event to the application event handler. The application then makes a new request further out in time. This request succeeds (it does not collide with anything), and a new timeslot is scheduled.

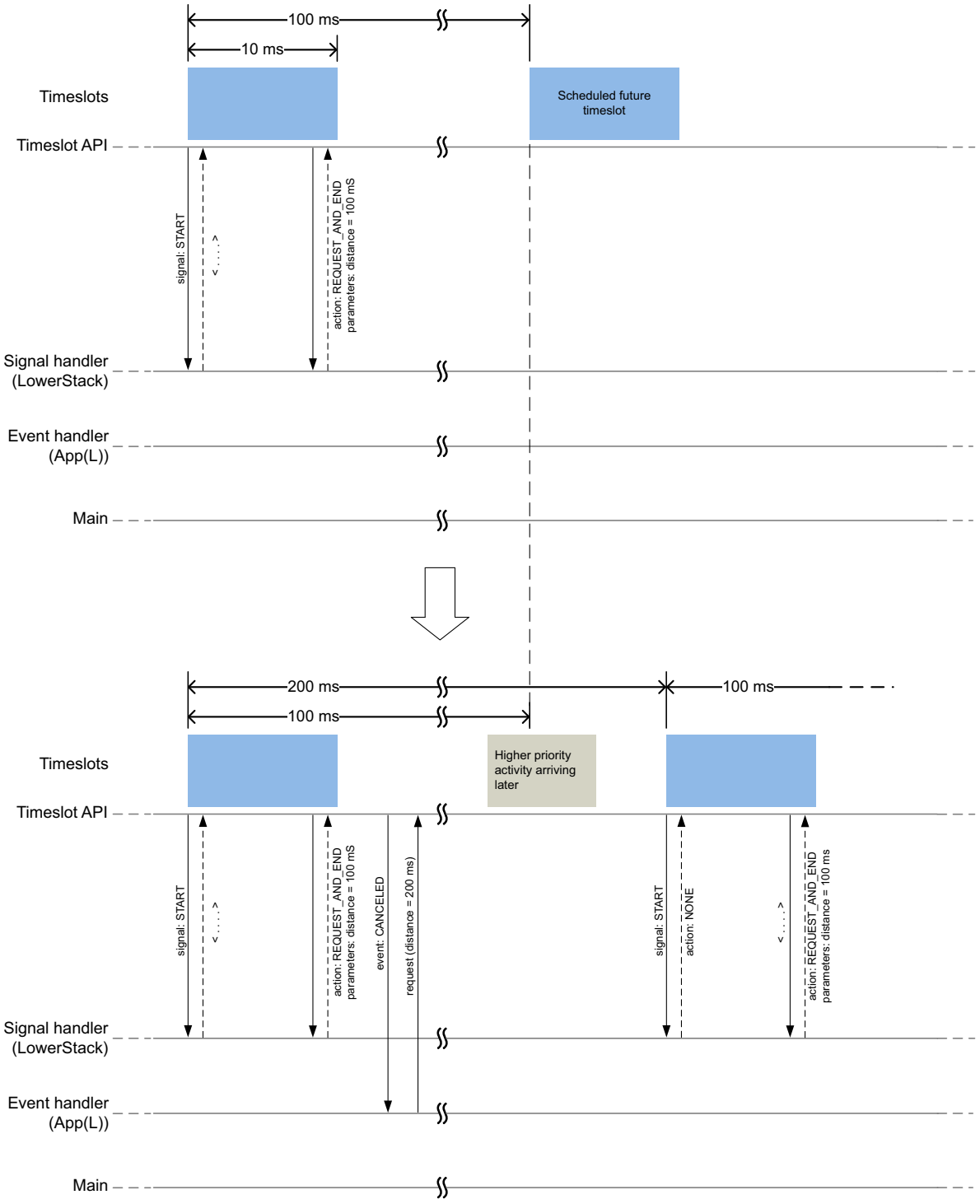

Figure 6 *Canceled timeslot example*

8.7.4 Timeslot extension example

Figure 7 shows how an application can use timeslot extension to create long continuous timeslots that will give the application as much radio time as possible while disturbing the SoftDevice activities as little as possible. In the first slot in ***Figure 7***, the application uses the signal handler return action to request an extension of the timeslot. The extension is granted, and the timeslot is seamlessly prolonged. The second attempt at extending the timeslot fails, as a further extension would cause a collision with a SoftDevice activity that has been scheduled. Therefore the application does a new request, of type *earliest*. This results in a new radio timeslot being scheduled immediately after the SoftDevice activity. This new timeslot can be extended a number of times.

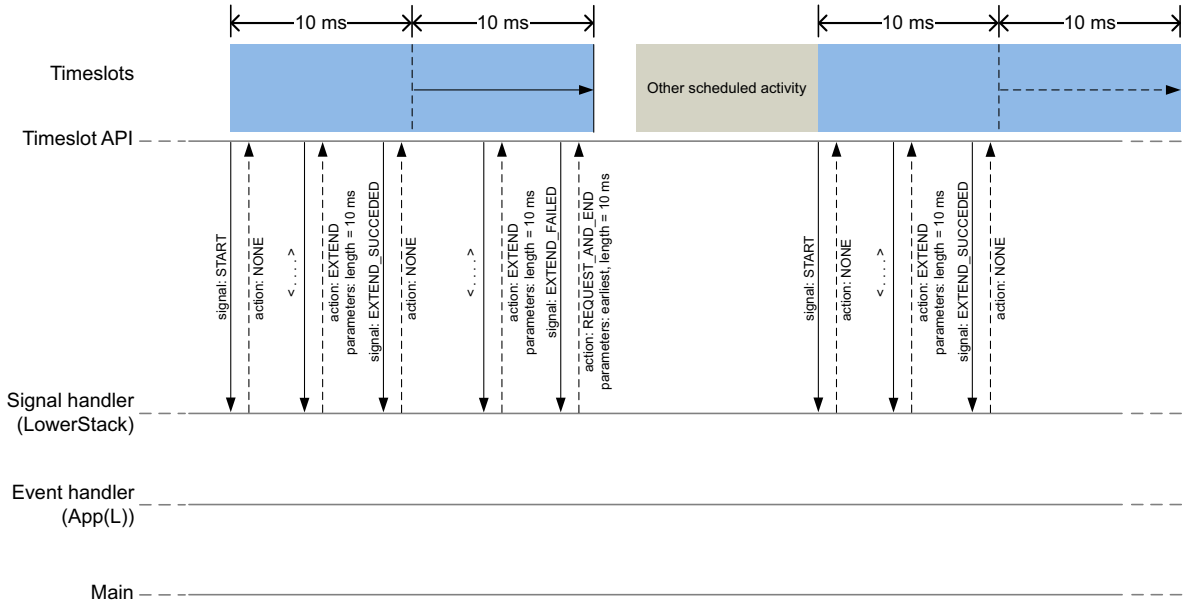

Figure 7 *Timeslot extension example*

9 Master Boot Record and Bootloader

The SoftDevice supports the use of a bootloader. A bootloader may be used to update the firmware on the chip.

The SoftDevice also contains a Master Boot Record (MBR). The MBR is necessary in order for the bootloader to update the SoftDevice, or to update the bootloader itself.

The MBR is a required component in the system. The inclusion of a bootloader is optional.

9.1 Master Boot Record

The Master Boot Record (MBR) module occupies a defined region in flash memory where the System Vector table resides. All exceptions (reset, hard fault, interrupts, SVC) are processed first by the MBR and then forwarded to appropriate handlers (for example, bootloader, SoftDevice). The main feature of the MBR is to provide an interface to allow in-system updates of the SoftDevice and bootloader firmware. The MBR is not updated between versions of the SoftDevice, meaning that during an update process, the MBR is never erased. The MBR ensures safe restart of any ongoing update process if an unexpected reset occurs.

9.2 Bootloader

The bootloader is supported and may be used to handle in-system update procedures. A bootloader has access to the full SoftDevice API and can be implemented just as any application that uses a SoftDevice. In particular, the bootloader can make use of the SoftDevice API to enable protocol stack interaction.

The bootloader is supported in the SoftDevice architecture by using a configurable base address for the bootloader in application code space. The base address is configured by setting the UICR.BOOTADDR register. The bootloader is responsible for determining the start address of the application. It uses sd_softdevice_vector_table_base_set(uint32_t address) to tell the SoftDevice where the application starts. The bootloader is also responsible for keeping track of, and verifying the SoftDevice. If an unexpected reset occurs during an update of the SoftDevice, it is the responsibility of the bootloader to detect this and recover.

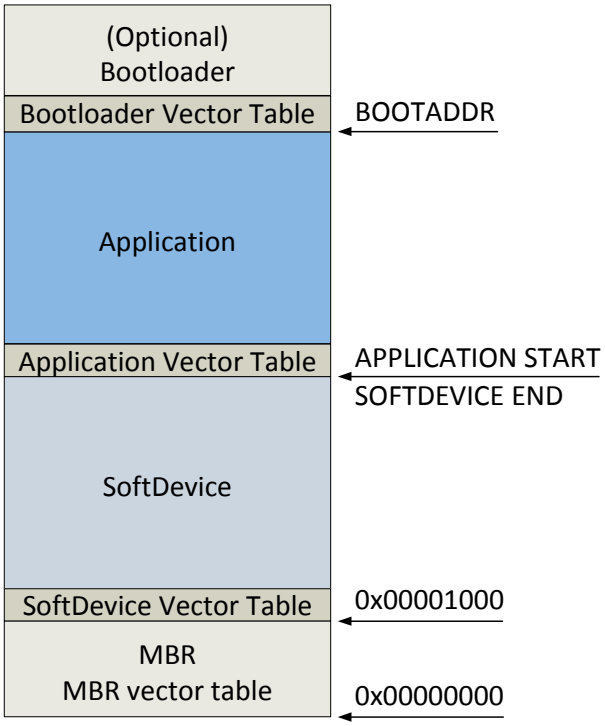

Figure 8 *MBR, SoftDevice, and bootloader architecture*

9.3 Master Boot Record (MBR) and SoftDevice reset behavior

Upon system reset, the MBR Reset Handler is run as specified by the System Vector table. The MBR and SoftDevice reset behavior is as follows:

- If an in-system bootloader update procedure is in progress:
  - Then in-system update procedure is run to completion.
  - System is reset.
- Else if SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET has been called previously:
  - Forward interrupts to the parameter given.
  - Run from Reset Handler (defined in vector table at parameter given).
- Else if a bootloader is present:
  - Forward interrupts to the bootloader.
  - Run Bootloader Reset Handler (defined in bootloader vector table at BOOTADDR).
- Else if a SoftDevice is present:
  - Forward interrupts to SoftDevice.
  - Run SoftDevice Reset Handler (defined in SoftDevice vector table at 0x00001000).
  - In this case, APPLICATION START is hardcoded inside the SoftDevice.
  - SoftDevice run Application Reset Handler (defined in application vector table at APPLICATION START).
- Else system startup error:
  - Sleep forever.

9.4 Master Boot Record (MBR) and SoftDevice initialization

The SoftDevice can be enabled by the bootloader by performing the following in this order:

1. Issue command for MBR to forward interrupts to the SoftDevice using sd_mbr_command() with SD_MBR_COMMAND_INIT_SD.
2. Issue command for the SoftDevice to forward interrupts to the bootloader using sd_softdevice_vector_table_base_set(uint32_t address) with BOOTADDR as parameter.
3. Enable SoftDevice using sd_softdevice_enable().

For a bootloader to transfer execution from itself to the application, the following can be performed:

1. If interrupts have not been forwarded to SoftDevice, issue command for MBR to forward interrupts to SoftDevice using sd_mbr_command() with SD_MBR_COMMAND_INIT_SD.
2. Ensure SoftDevice is disabled using sd_softdevice_disable().
3. Issue command for the SoftDevice to forward interrupts to the application using sd_softdevice_vector_table_base_set(uint32_t address) with APPLICATION_START as parameter.
4. Branch to application's reset handler after reading the handler from the Application Vector Table.

10 SoC resource requirements

The SoftDevice and MBR are designed to be installed on a System on Chip (SoC) in the lower part of the code memory space. After a reset, the MBR will use some RAM to store state information. When the SoftDevice is enabled, it uses resources on the chip including RAM and hardware peripherals like the radio. This chapter describes how the MBR and SoftDevice uses resources. The SoftDevice requirements are shown both when enabled and disabled.

10.1 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 10*** on page 30. Note the definitions of Region 0 (R0) and Region 1 (R1) are valid only when the CLENR0 and RLENR0 registers are optionally programmed to enable memory protection. See the MPU chapter in the *nRF51 Reference Manual* for more details.

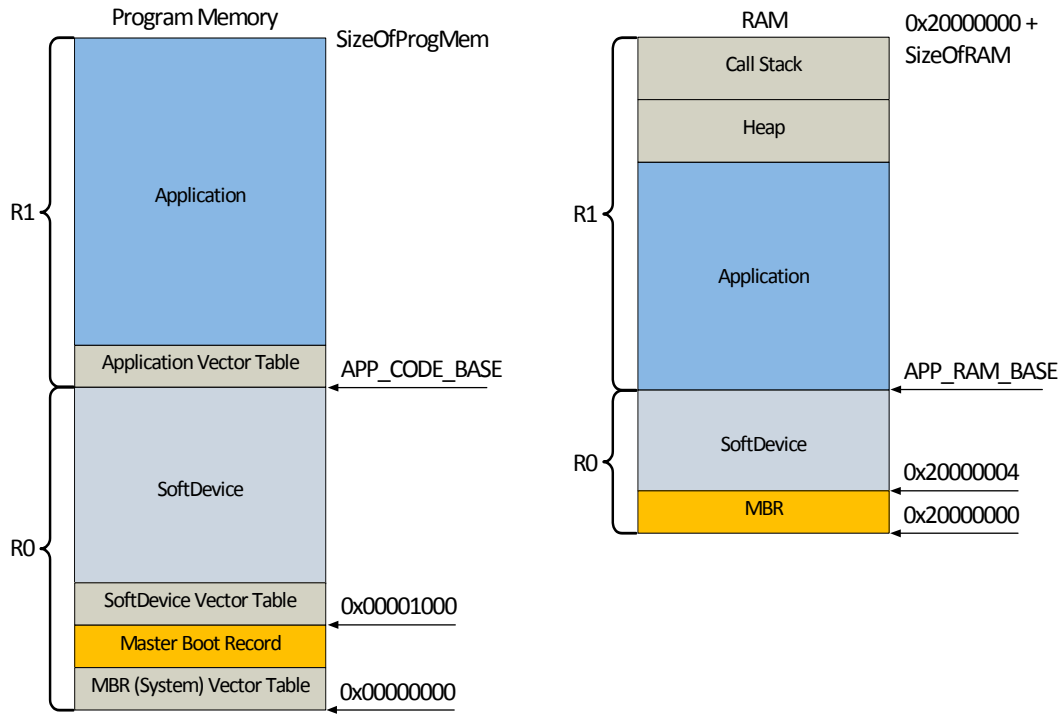

Figure 9 *Memory resource map*

Flash	S210 Enabled	S210 Disabled
SoftDevice MBR	48 kB[1] 4 kB	48 kB 4 kB
APP_CODE_BASE	0x0000D000	0x0000D000

RAM	S210 Enabled	S210 Disabled
SoftDevice MBR	2300 bytes 4 bytes	4 bytes 4 bytes
APP_RAM_BASE	0x20000900	0x20000008

Call stack[2]	S210 Enabled	S210 Disabled
Maximum usage	1 kB	0 bytes

Heap	S210 Enabled	S210 Disabled
Maximum allocated bytes	0 bytes	0 bytes

1. 1 kB = 1024 bytes.
2. 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 10 *S210 Memory resource requirements*

10.2 Hardware blocks and interrupt vectors

Table 11 defines access types used to indicate the availability of hardware blocks to the application. ***Table 12*** on page 31 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 11 *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	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 13*** on page 32 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 12 *Peripheral protection and usage by SoftDevice*

10.3 Application signals - software interrupts

Software interrupts are used by the SoftDevice to signal a change in events. ***Table 13*** 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 13 *Software interrupt allocation*

10.4 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 14*** 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	-	Channels 20 - 31
SoftDevice	Channels 20 - 31	-

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

Table 14 *PPI channel and group availability*

10.5 SVC number ranges

Table 15 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 15 *SVC number allocation*

10.6 External requirements

For correct operation of the SoftDevice, it is a requirement that the 16 MHz crystal oscillator (16 MHz XOSC) startup time is less than 1.5 ms. The external clock crystal and other related components must be chosen accordingly. Data for the device XOSC input can be found in the product specification for the device.

11 Processor availability and interrupt latency

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

11.1 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. Additional latency is incurred due to interrupt processing and forwarding performed by the Master Boot Record (MBR). The maximum application interrupt latency is dependent on protocol stack activity as described in ***Section 11.2 "Processor availability"*** on page 35.

Interrupt	CPU cycles	Latency at 16 MHz
Open peripheral interrupt	49	3.1 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	67	4.2 µs
Application SVC interrupt	43	2.68 µs

Table 16 *Additional latency due to SoftDevice processing*

See ***Table 12*** on page 31 for open, blocked, and restricted peripherals.

11.2 Processor availability

The nWP-20 *SoftDevice Architecture Overview* white paper, describes interrupt management in SoftDevices and is required knowledge for understanding this section.

The SoftDevice protocol stack runs 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 context. ***Figure 10*** illustrates UpperStack activity (API calls) and LowerStack activity (Protocol events) and the time reserved/not reserved for those interrupts.

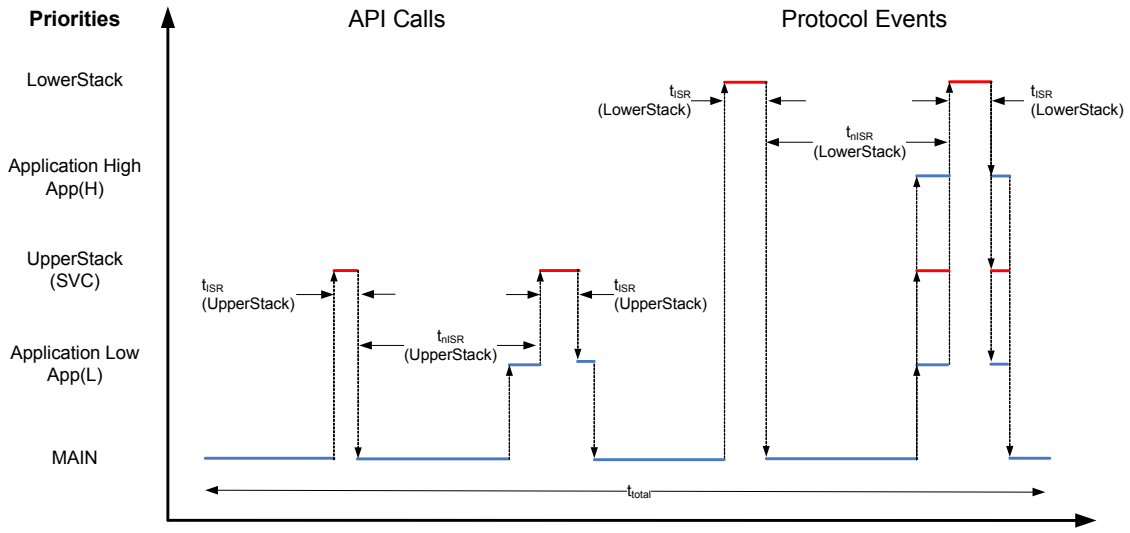

Figure 10 *UpperStack and LowerStack activity*

Table 17 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 context.
t_{nISR} (UpperStack)	Time between UpperStack interrupts. This is the time available to run for App(L) and main context.

Table 17 *SoftDevice interrupt latency definitions*

11.3 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).

11.3.1 ANT protocol event

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

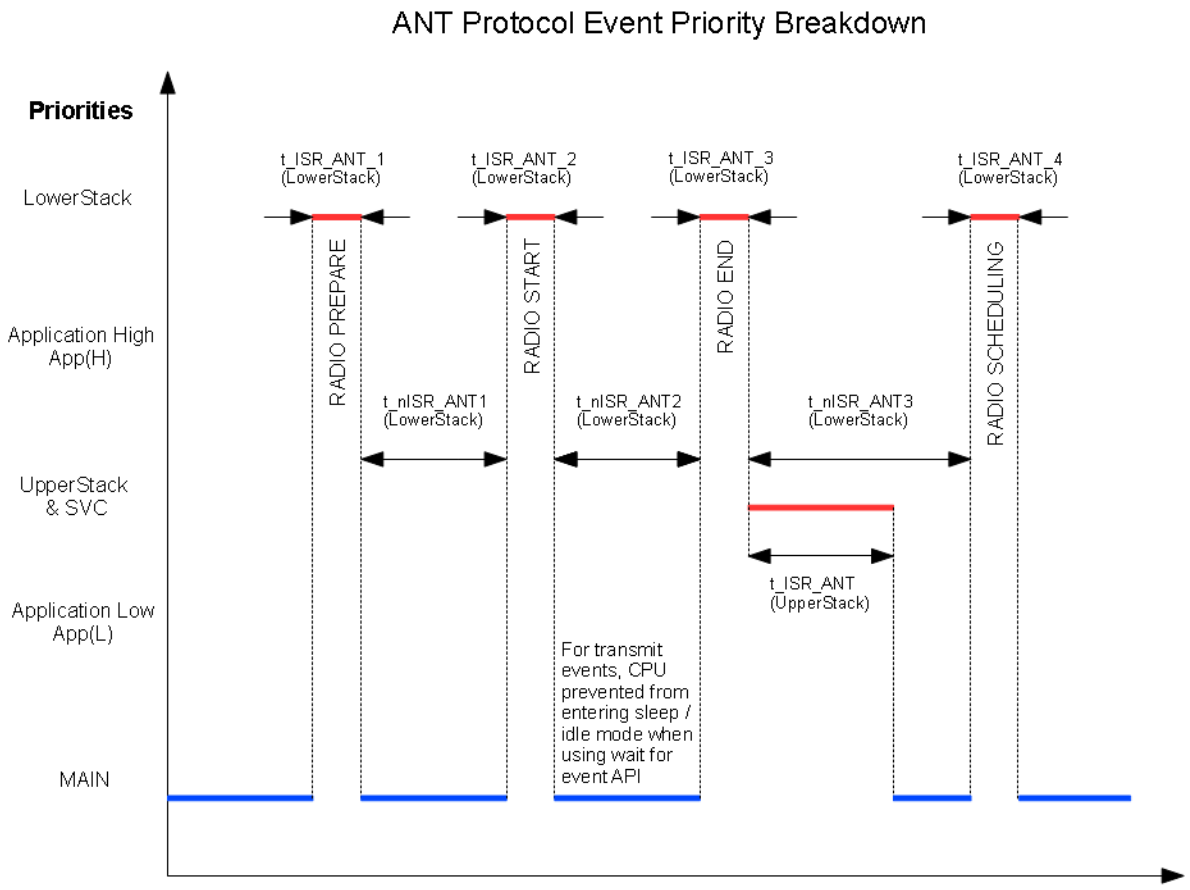

Figure 11 *ANT protocol event*

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.

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 18***. High priority application interrupts will directly affect ANT STACK PROCESSING overhead time (t_ISR_ANT).

Parameter	Description	Min.	Typ.	Max.
	LowerStack			
$t_{ISR\_ANT\_1}$ (LowerStack)	High priority process – RADIO PREPARE.			167 µs
$t_{nISR\_ANT\_1}$ (LowerStack)	Free processing time during RADIO PREPARE and RADIO START.	78 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 RADIO 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 RADIO SCHEDULING process.[1]	189 µs		
$t_{ISR\_ANT\_4}$ (LowerStack)	High priority process – RADIO SCHEDULING.		95 µs	
	UpperStack			
t_{ISR_ANT} (UpperStack)	Low priority stack process.[1]			309 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 18 *SoftDevice interrupt latency LowerStack/UpperStack for an ANT protocol event*

11.3.2 ANT API calls

The timing for API call handling in the UpperStack is described in ***Table 19***.

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 trigger the UpperStack interrupt.

Table 19 *SoftDevice interrupt latency UpperStack for ANT API calls*

11.4 Performance with Flash memory API

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

12 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.

12.1 Master Channel

Radio transmit and receive event occurrence for a single ANT master channel instance is shown in ***Figure 12***.

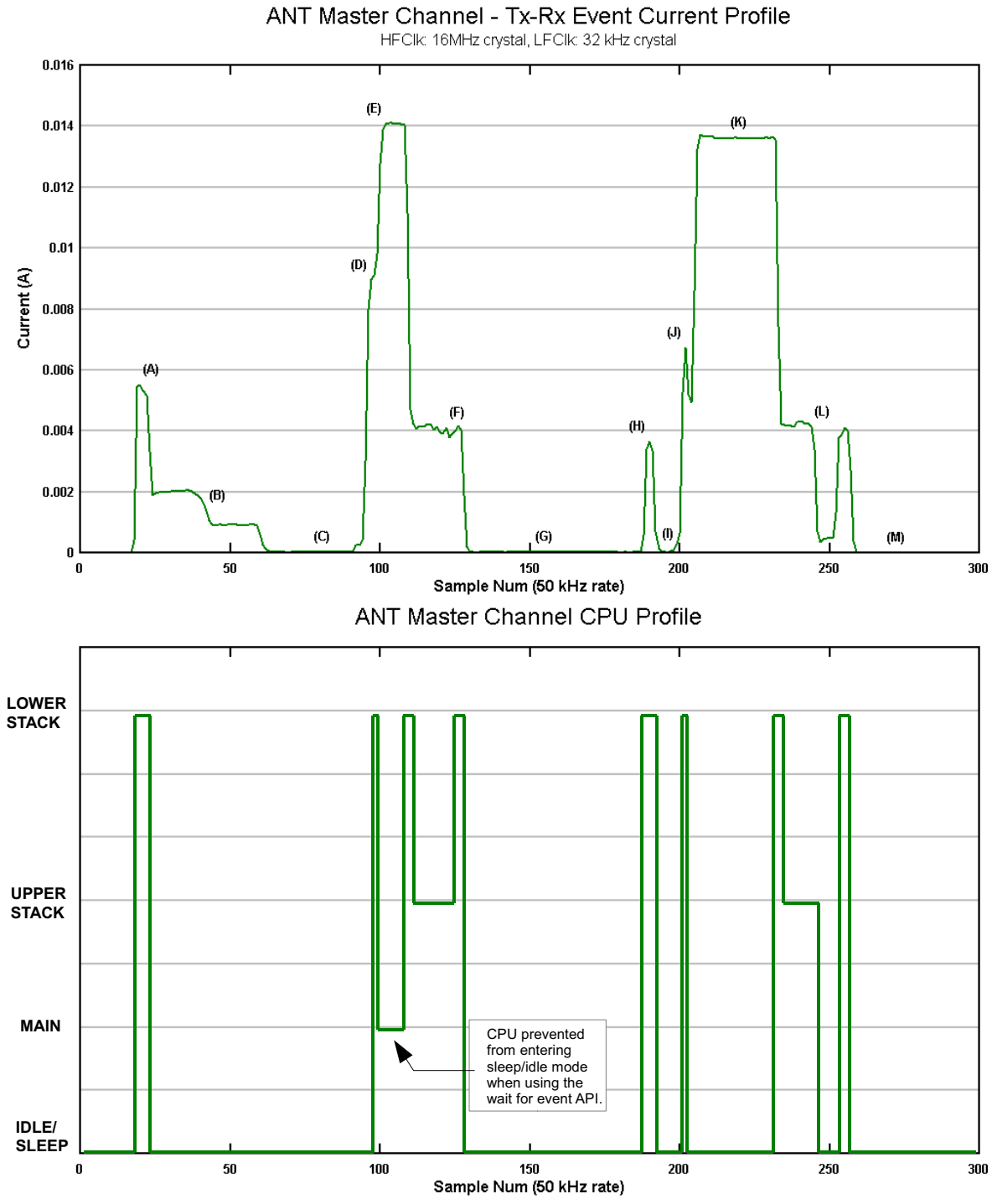

Figure 12 *ANT Master Transmit Channel*

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 20 *ANT Master Transmit Channel*

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

12.2 Slave Receive Channel

A radio receive event occurrence for a single ANT slave channel instance are shown in ***Figure 13***.

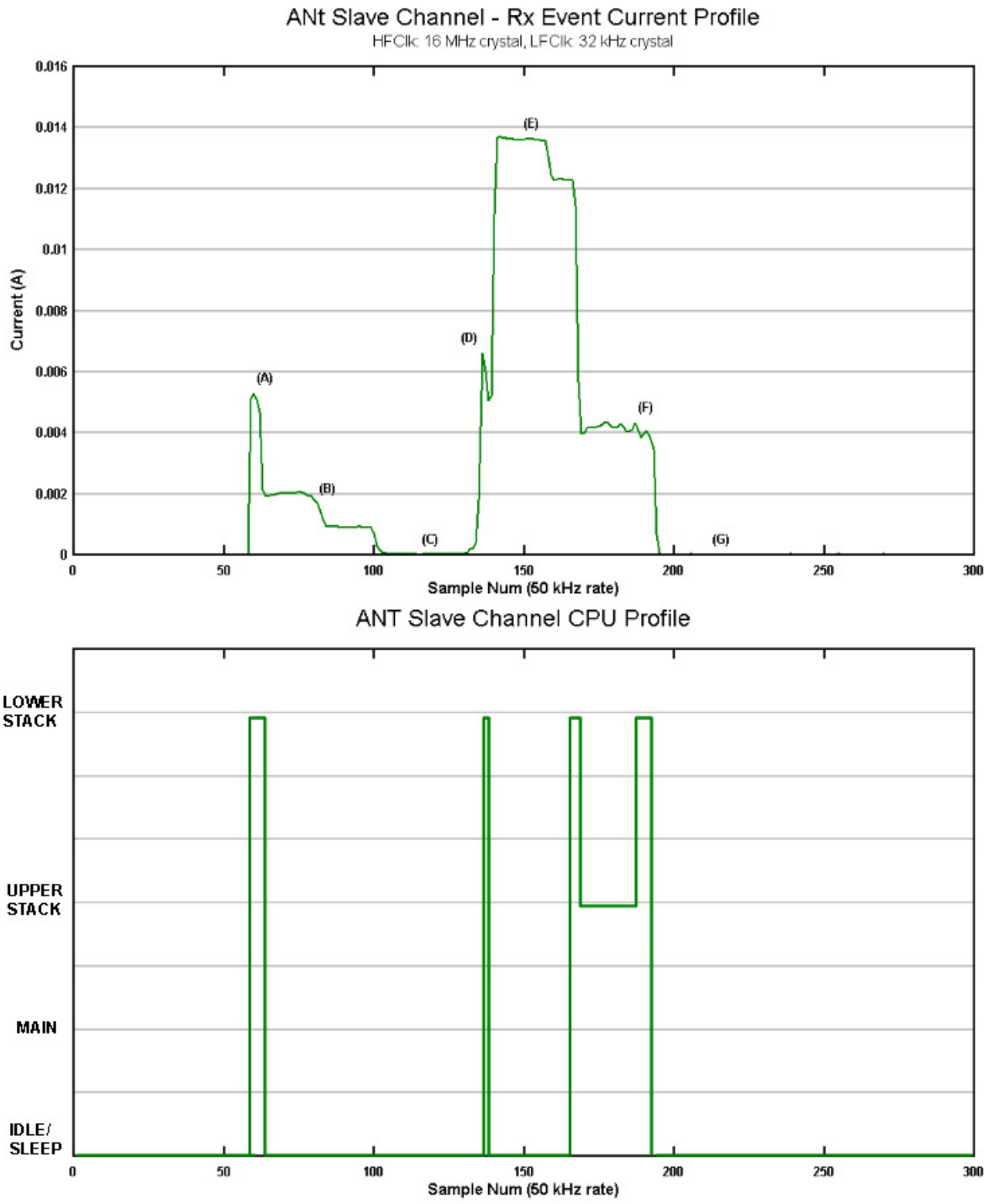

Figure 13 *ANT Slave Receive Channel*

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 21** 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.

13 SoftDevice identification and revision scheme

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

For revisions of the SoftDevice that are production qualified, the version string consists of major, minor, and revision numbers only, as described in ***Table 22***.

For revisions of the SoftDevice that 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 23***.

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 22 *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 23 *SoftDevice revision examples*

The test qualification levels are outlined in ***Table 24.***

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 24 *Test qualification levels*

13.1 MBR distribution and revision scheme

The MBR is distributed in each SoftDevice hex file. The version of the MBR distributed with the SoftDevice will be published in the release notes for the SoftDevice and uses the same major, minor and revision numbering scheme as described here.

13.2 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.