S110 nRF51822

Bluetooth® low energy Peripheral SoftDevice

SoftDevice Specification v1.3

Key Features

- *Bluetooth*® 4.1 compliant low energy single-mode protocol stack
    - Link layer
    - L2CAP, ATT, and SM protocols
    - GATT, GAP, and L2CAP
    - Concurrent Peripheral and Broadcaster roles
    - GATT Client and Server
    - Full SMP support including MITM and OOB pairing
- Complementary nRF51 SDK including *Bluetooth* profiles and example applications
- 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 multiprotocol operation concurrent with *Bluetooth* low energy connections and non-concurrently
    - Concurrent multiprotocol timeslot API
    - Alternate protocol stack running in application space

Applications

- Computer peripherals and I/O devices
    - Mouse
    - Keyboard
    - Multi-touch trackpad
- Interactive entertainment devices
    - Remote control
    - 3D glasses
    - Gaming controller
- Personal Area Networks
    - Health and fitness sensor and monitor devices
    - Medical devices
    - Key fobs and wrist watches
- Remote control toys
- Home automation

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
June 2014	1.3	Updated for S110 SoftDevice v7.0.0. **Added:** • ***Chapter 10.6 "External requirements"*** on page 35 **Updated:** • Key Features on front page • ***Section 1.1 "Documentation"*** on page 5 • ***Section 2.2 "Multiprotocol support"*** on page 6 • ***Section 3.1 "Profile and service support"*** on page 8 • ***Section 3.2 "Bluetooth low energy features"*** on page 9 • ***Chapter 4 "SoC library"*** on page 12 • ***Chapter 5 "SoftDevice Manager"*** on page 13 • ***Chapter 6 "Flash memory API"*** on page 14 • ***Section 8.6 "Multiprotocol timeslot API"*** on page 21 • ***Section 9.1 "Master Boot Record (MBR)"*** on page 30 • ***Section 10.1 "Memory resource map and usage"*** on page 31
April 2014	1.3A	Updated for S110 SoftDevice v7.0.0. alpha **Added:** • ***Chapter 8 "Concurrent Multiprotocol Timeslot API"*** on page 19 • ***Section 9.1 "Master Boot Record (MBR)"*** on page 30 **Updated:** • Key Features on front page • ***Section 2.2 "Multiprotocol support"*** on page 6 • ***Table 3 "GAP features in the BLE stack"*** on page 9 • ***Table 9 "Proprietary features in the BLE stack"*** on page 11 • ***Table 27 "Additional latency due to SoftDevice processing"*** on page 36 • ***11.4 "Performance with Flash memory API and Concurrent Multiprotocol Timeslot API"*** on page 40
November 2013	1.2	Updated for S110 v6.0.0 release. Added ***Chapter 6 "Flash memory API"*** on page 14; Added ***Chapter 9 "Bootloader"*** on page 29 Updated ***Table 1*** on page 8; Updated ***Table 4*** on page 10; Updated ***Table 10*** on page 12; Updated ***Chapter 7 "Radio Notification"*** on page 15; Updated ***Table 17*** on page 19.

Date	Version	Description
March 2013	1.1	Updated for changes made as of S110 v5.0.0; Changed ***Section 9.2 "Processor availability"*** on page 37 and ***Section 13 "BLE power profiles"*** on page 42; Changed ***Table 27*** on page 37; Added ***Table 28*** on page 38; Changed ***Table 30*** on page 40; Changed ***Figure 16*** on page 43 and ***Figure 17*** on page 44.
February 2013	1.0	Changed Memory resource requirements in ***Table 16*** on page 19; Added ***Section 9.3 "Application signals - software interrupts"*** on page 21; Updated ***Chapter 9 "BLE performance"*** on page 36 and added ***Section 9.3 "Data throughput"*** on page 40; Updated diagrams in ***Chapter 13 "BLE power profiles"*** on page 42; Added ***Chapter 14 "SoftDevice identification and revision scheme"*** on page 45; Updated ***Chapter 14.1 "Notification of SoftDevice revision updates"*** on page 46.
September 2012	0.6	First release.

1 Introduction

The S110 SoftDevice is a *Bluetooth*® low energy (BLE) Peripheral protocol stack solution. It integrates a low energy controller and host, and provides a full and flexible application programming interface (API) for building *Bluetooth* low energy System on Chip (SoC) solutions.

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 14.1 "Notification of SoftDevice revision updates"*** on page 46 for more information. To find information on any limitations or omissions, please refer to the SoftDevice release notes, which contain a detailed summary of the release status.

1.1 Documentation

Below is a list of the core documentation for the 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.
nRF51822 Product Specification (PS)	Contains a description of the hardware, modules, and electrical specifications specific to the nRF51822 chip.
nRF51822 Product Anomaly Notification (PAN)	Contains information on anomalies related to the nRF51822 chip.
Bluetooth Core Specification	The *Bluetooth Core Specification* version 4.1, Volumes 1, 3, 4, and 6 describes *Bluetooth* terminology which is used throughout the SoftDevice Specification.

2 Product overview

This section provides an overview of the SoftDevice.

2.1 SoftDevice

The SoftDevice is a precompiled and linked binary software that integrates a *Bluetooth* 4.1 low energy protocol stack on the nRF51x22 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 *Bluetooth* low energy applications with the SoftDevice.

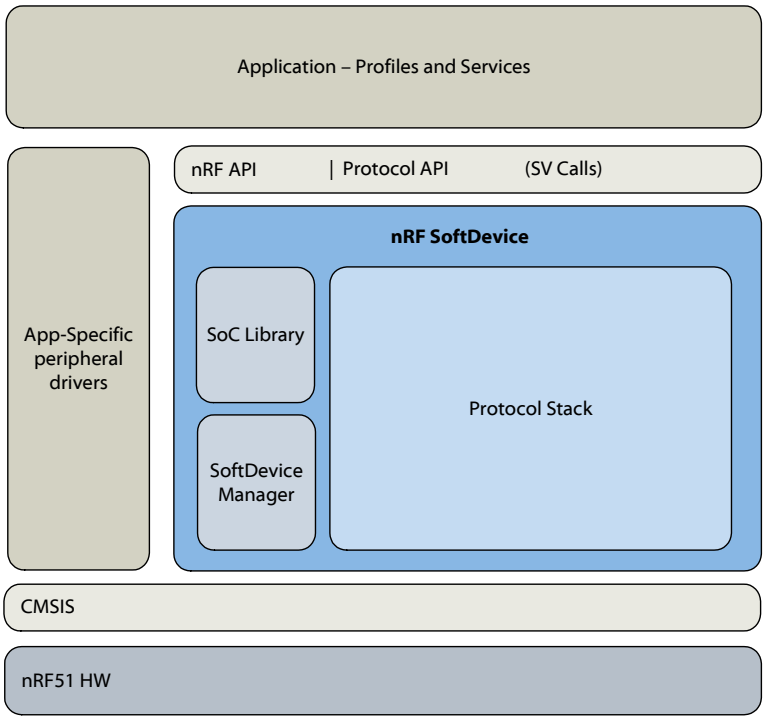

Figure 1 *System on Chip application with the SoftDevice*

The SoftDevice can be programmed onto compatible nRF51 Series chips during both development and production. 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 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 19.

3 *Bluetooth* low energy protocol stack

The *Bluetooth* 4.1 compliant low energy 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) complements 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).

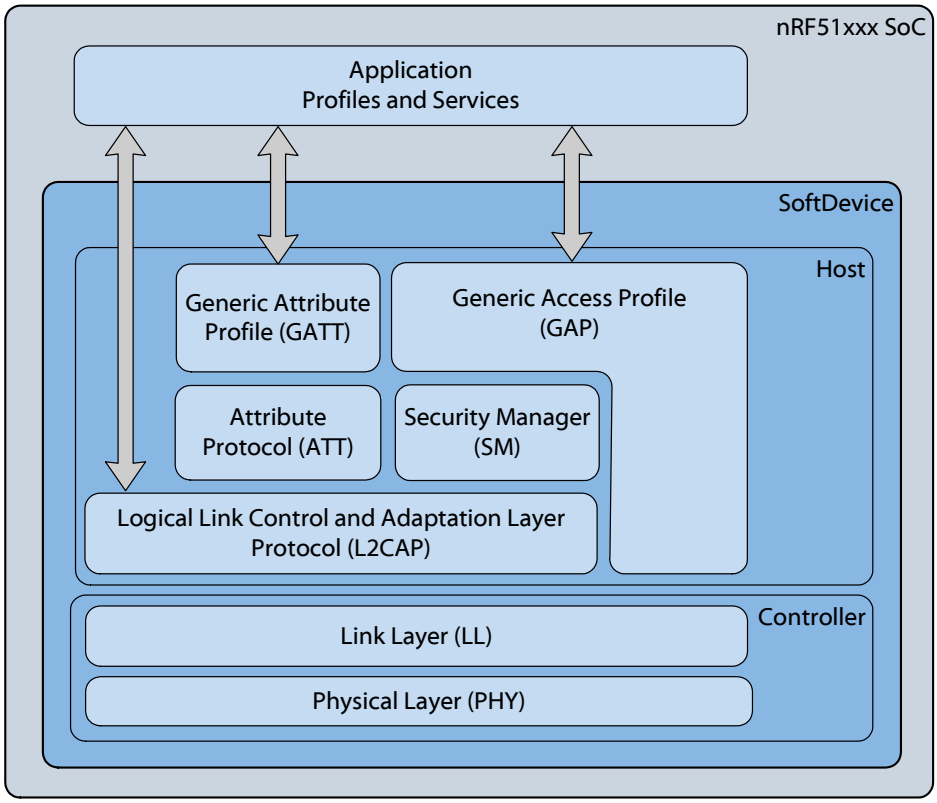

Figure 2 *SoftDevice stack architecture*

3.1 Profile and service support

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

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

Table 1 *Supported profiles and services*

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

3.2 *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. The SoftDevice implements GAP and GATT procedures and modes that are common to most profiles, such as the handling of discovery, connection, pairing, and bonding.

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.
Attribute Table population and access	Full flexibility to populate the Attribute Table at runtime, attribute removal is not supported.
Asynchronous and event driven	Thread-safe function and event model enforced by the architecture.
Vendor-specific (128 bit) UUIDs for proprietary profiles	Compact, fast, and memory efficient management of 128 bit UUIDs.
Packet flow control	Full application control over data buffers to ensure maximum throughput.

Table 2 *API features in the BLE stack*

GAP features	Description
Multi-role: Peripheral and Broadcaster	Broadcaster can run concurrently with a peripheral in a connection.
Multiple bond support	Keys and peer information stored in application space. No restrictions in stack implementation.
Security mode 1: Levels 1, 2, and 3	Support for all levels of SM 1.
User-defined Advertising data	Full control over advertising and scan response data for the application.

Table 3 *GAP features in the BLE stack*

GATT Features	Description
Full GATT Server	Including Service Changed Support
Support for authorization:	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	Efficient usage of key generation algorithms to minimize memory overheads.
Authenticated MITM (Man in the middle) protection	Protects the bonding procedure against malicious attackers.
Pairing methods: Just works, Passkey Entry, and Out of Band	Full control over the pairing algorithm for strict security requirements.

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

ATT Features	Description
Server protocol	
Client protocol	
Max MTU size 23 bytes	

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

L2CAP Features	Description
27 byte MTU size	
Low level L2CAP API access	Ability to send arbitrary L2CAP data from the application.

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	
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.
Full Privacy 1.1 support	Synchronous and low power solution for BLE enhanced privacy with hardware-accelerated address resolution for whitelisting.
Master Boot Record (MBR) for Device Firmware Update (DFU)	Enables over-the-air SoftDevice replacement, giving full SoftDevice update capability.

Table 9 *Proprietary features in the BLE stack*

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, see ***Chapter 8 "Concurrent Multiprotocol Timeslot API"*** on page 19.
Radio notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See ***Chapter 7 "Radio Notification"*** on page 15.
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 10 *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 selects between the following options: • RC oscillator • Crystal oscillator

Table 11 *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 active BLE connections.

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, but shorter write sizes have higher probability of success.
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 12 *Behavior with BLE 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 to the application notifying when the radio is in use. The signal is sent using software interrupt, as specified in ***Table 24*** on page 34.

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.

Figure 3 shows the active signal in relation to the Radio Event.

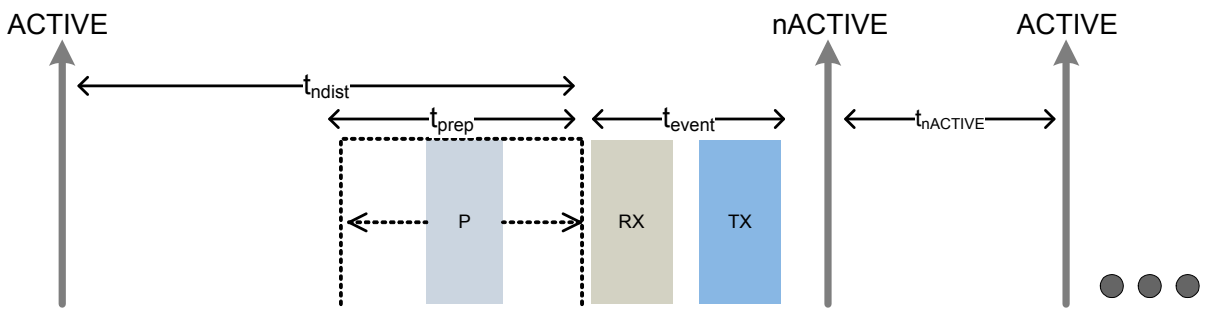

Figure 3 *BLE Radio Notification*

Many packets can be sent and received in one Radio Event. Radio Notification events will be as shown in ***Figure 4***.

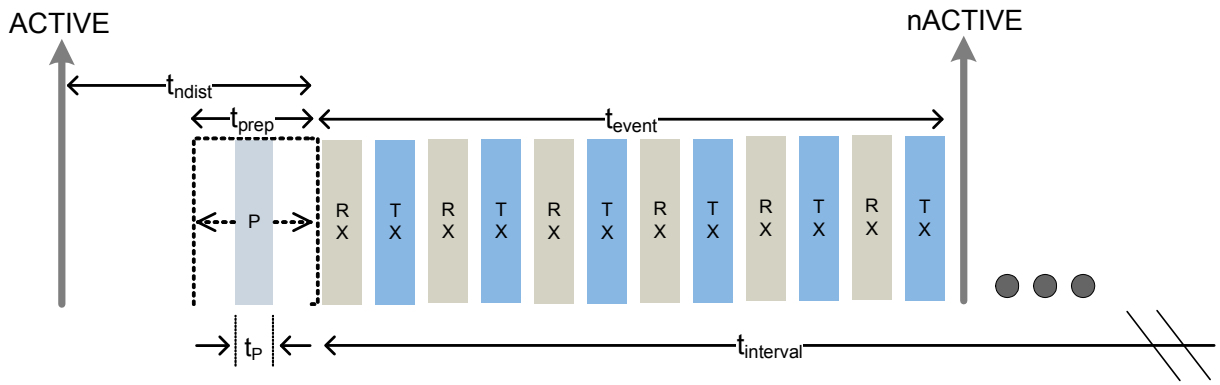

Figure 4 *BLE Radio Notification, multiple packet transfers*

Table 13 describes the notation used in ***Figure 3*** and ***Figure 4*** on page 15.

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 3*** on page 15. See also ***Table 15*** on page 18.

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 to 2250 - 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*** on page 16, the amount of CPU time available between ACTIVE and a Radio Event is:

$$t_{ndist} - t_P$$

Shown below 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:

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

The stack will limit the length of a Radio Event (t_{event}), thereby reducing the maximum packets exchanged, to accommodate the selected t_{ndist}. ***Figure 5*** shows consecutive Radio Events with Radio Notification and illustrates the limitation in t_{event} which may be required to ensure t_{ndist} is preserved.

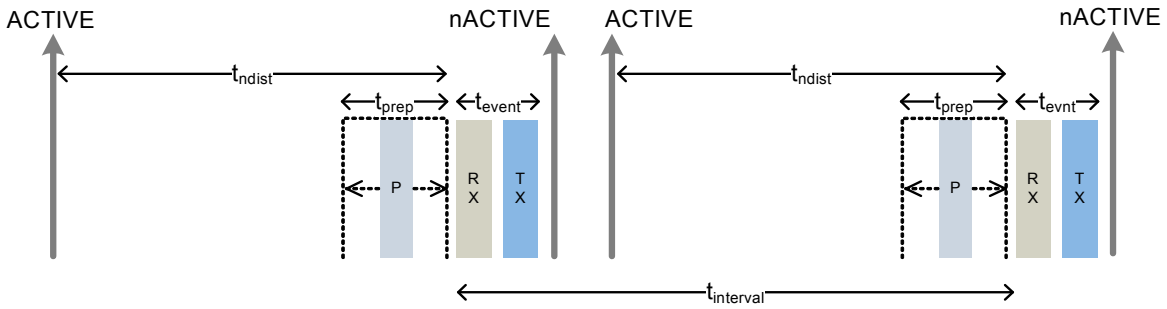

Figure 5 *Consecutive Radio Events with BLE Radio Notification*

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 Concurrent Multiprotocol Timeslot API

The Multiprotocol Timeslot API allows an application developer to safely schedule 2.4 GHz proprietary Radio usage while *Bluetooth* low energy is in use by the device. This allows the nRF51 device to be part of a BLE network 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 22 "Hardware access type definitions"*** on page 32 and ***Table 23 "Peripheral protection and usage by SoftDevice"*** on page 33.

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 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. Finally, the application may 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. To minimize the influence of the use of the Multiprotocol Timeslot API on other activities, using normal priority should be considered best practice. 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 15 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 Multiprotocol Timeslot API should be used with regard for the application configuration of the SoftDevice protocol. 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.

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 16 *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 "BLE peripheral performance"*** on page 38.

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 17 *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 "BLE peripheral performance"*** on page 38.

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 *nRF51 Reference Manual, chapter 16, 2.4 GHz radio (RADIO)*.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_TIMER0	Timer interrupt, for more information, see *nRF51 Reference Manual, chapter 17, Timer/counter (TIMER)*.
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 18 *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 19 *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 Examples

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

8.7.1 Complete session example

Figure 6 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 6***) 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.

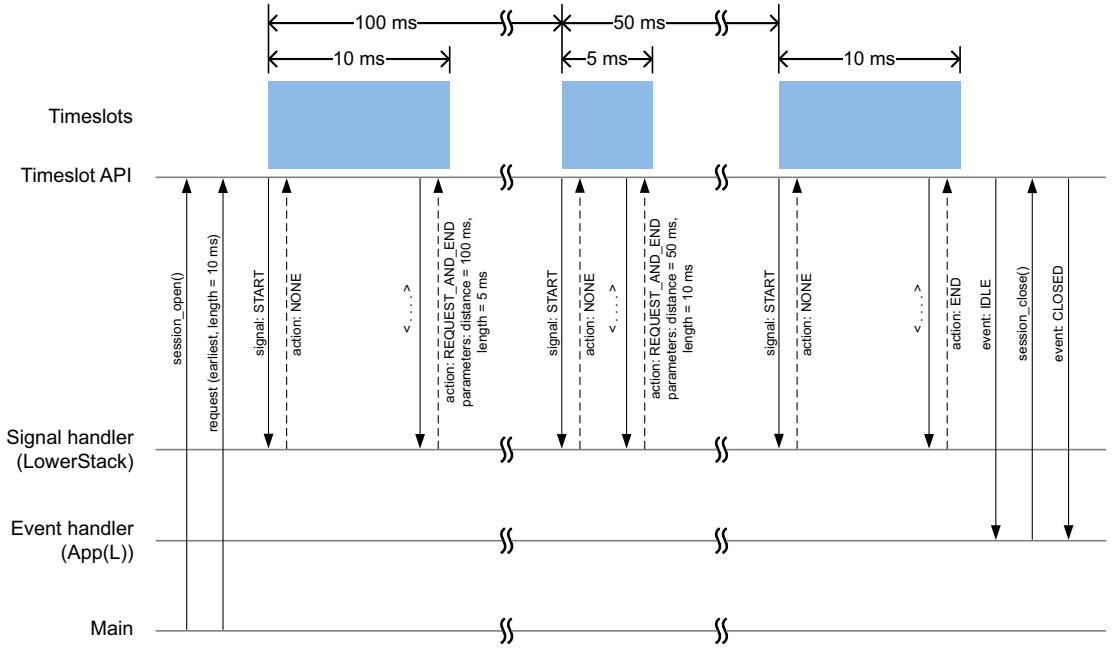

Figure 6 *Complete session example*

8.7.2 Blocked timeslot example

Figure 7 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 7***, the application signal handler returns a REQUEST_AND_END action to request a new timeslot. The new timeslot cannot be scheduled as requested, due to 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 requests succeeds (it does not collide with anything), and a new timeslot is scheduled.

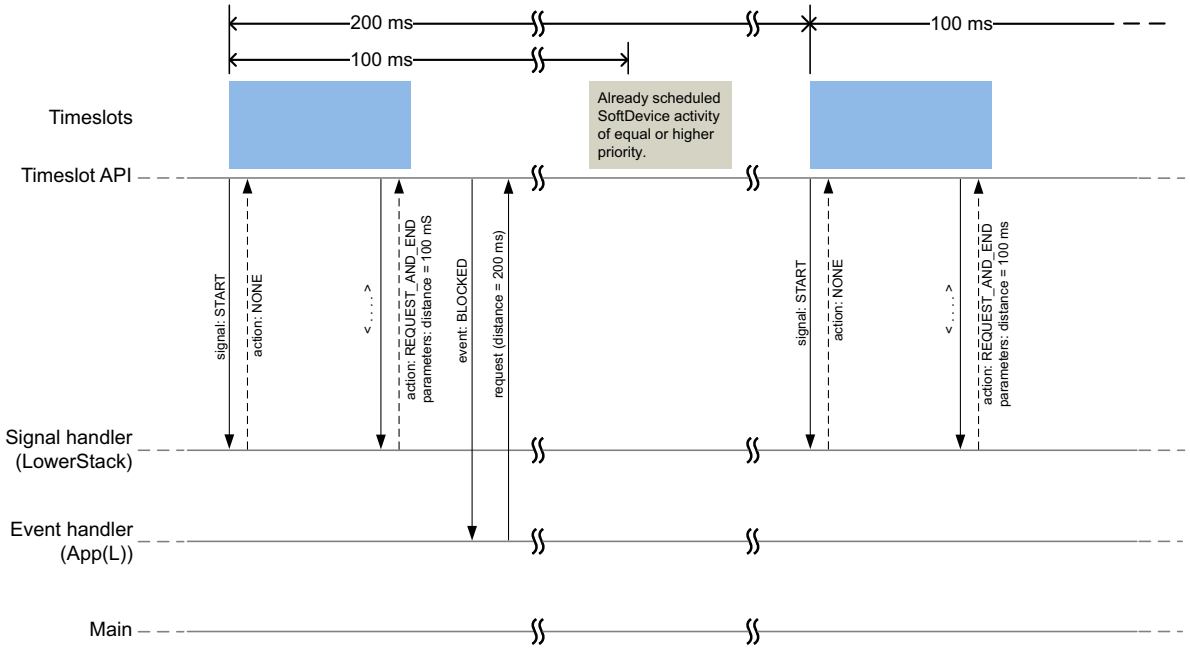

Figure 7 *Blocked timeslot example*

8.7.3 Canceled timeslot example

Figure 8 on page 27 shows a situation in the middle of a session where a requested and scheduled application timeslot is being revoked. The upper part of ***Figure 8*** on page 27 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 8*** on page 27 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 requests succeeds (it does not collide with anything), and a new timeslot is scheduled.

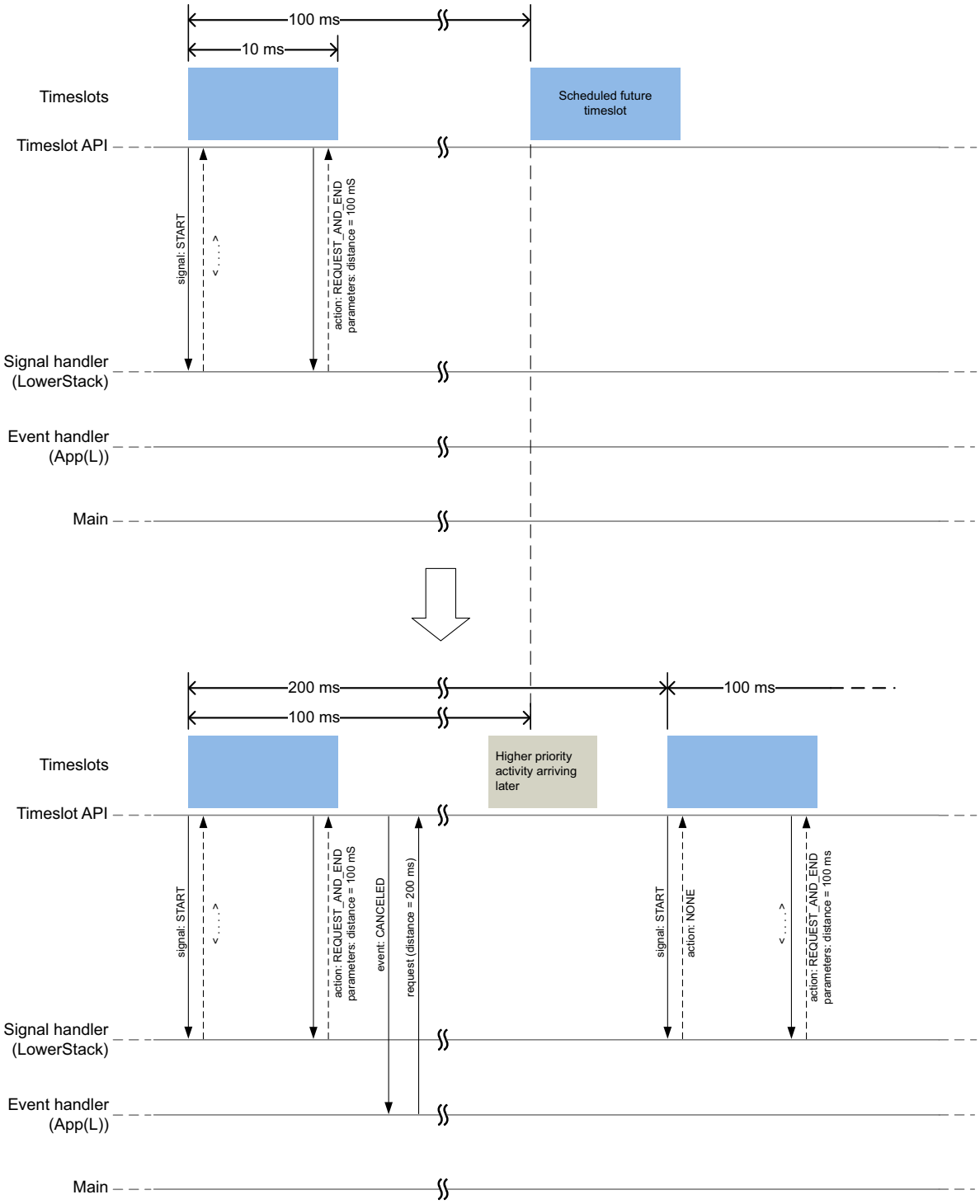

Figure 8 *Canceled timeslot example*

8.7.4 Timeslot extension example

Figure 9 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 9***, 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.

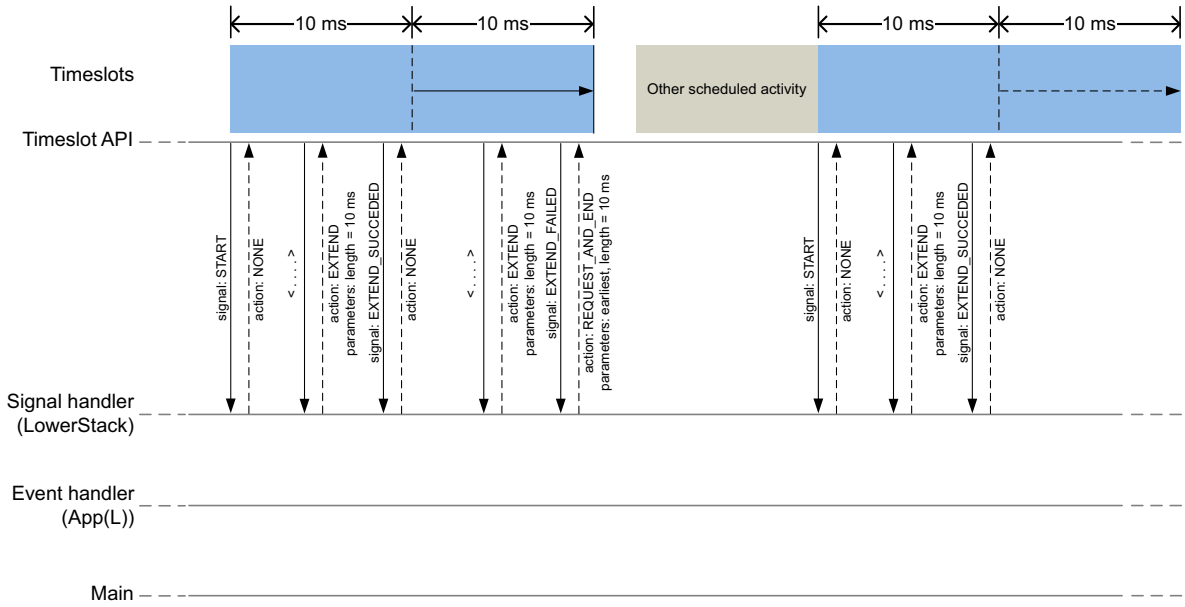

Figure 9 *Timeslot extension example*

9 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.

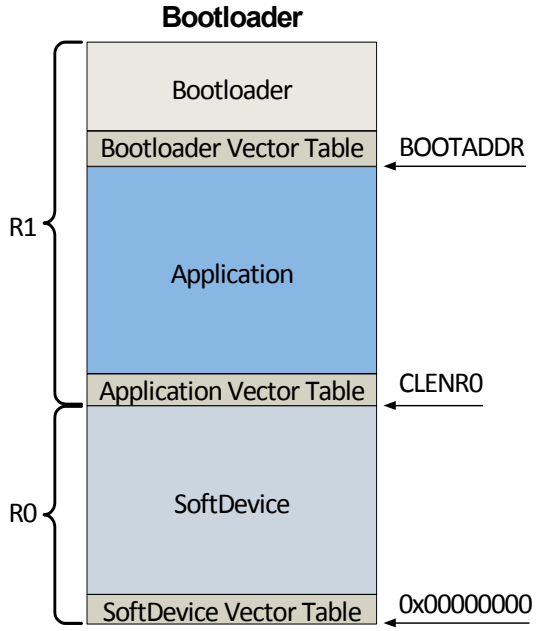

Figure 10 *R0 (SoftDevice) and R1 (Application + Bootloader).*

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*

9.1 Master Boot Record (MBR)

The Master Boot Record (MBR) makes it possible to update the SoftDevice and Bootloader. The MBR is first placed in flash memory where the System Vector table resides. All exceptions (reset, hard fault, interrupts SVC), are processed first by the MBR. The MBR is not updated between versions of the SoftDevice, meaning during an update process, the MBR is never erased. The MBR ensures safe restart of the copy process if an unexpected reset occurs.

The Bootloader is responsible for getting a new SoftDevice and/or bootloader stored in the application area. When the Bootloader wants to update the SoftDevice, it calls the copy function in the MBR which writes the new SoftDevice over the existing SoftDevice. The Bootloader is responsible for handling unexpected resets and restarts the copy process after resets.

If the Bootloader wants to update the bootloader itself, it invokes a "copy bootloader" function in the MBR. In this case the MBR handles unexpected resets during the copy process and guarantees the copy process will resume after an unexpected reset. On completion the new bootloader will be started.

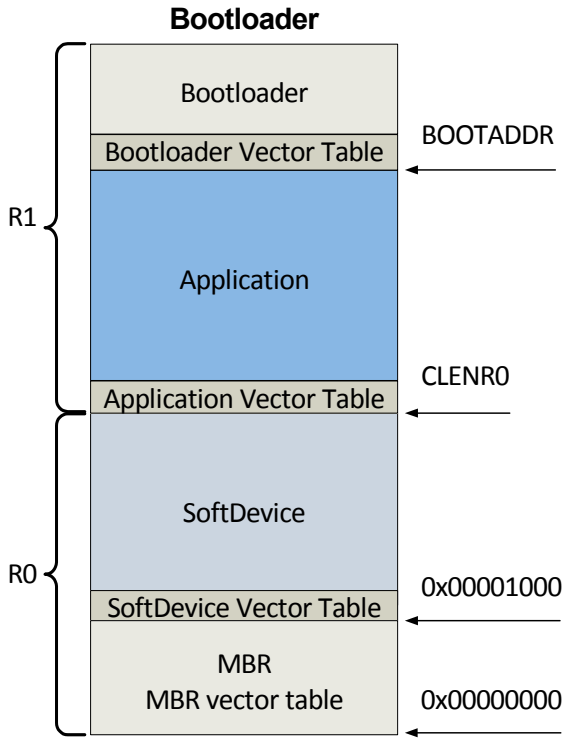

Figure 11 *Master Boot Record*

On every reset the MBR will start the Bootloader and forward all interrupts to the Bootloader without invoking the SoftDevice. The reason for this is because the MBR cannot know if the SoftDevice is corrupt or not. The Bootloader can start the SoftDevice by calling a function in the MBR, which causes the MBR to forward interrupts to the SoftDevice. After this, the Bootloader can use the SoftDevice as a normal application.

10 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.

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 12*** below. Memory resource requirements, both when the SoftDevice is enabled and disabled, are shown in ***Table 21*** on page 32.

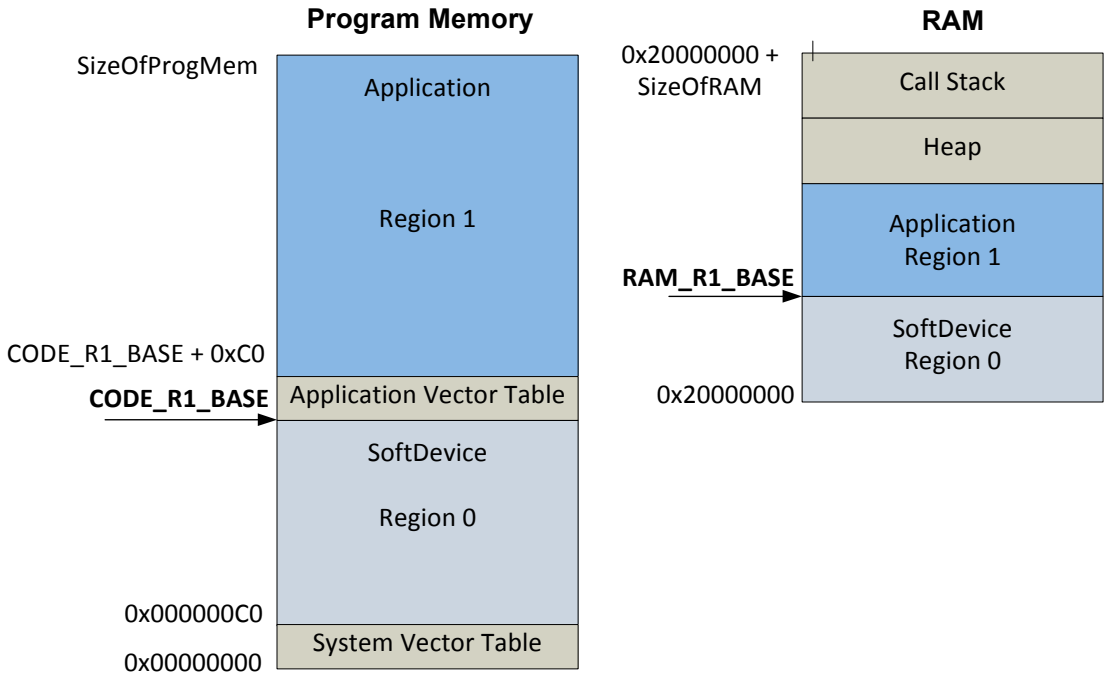

Figure 12 *Memory resource map*

Flash	S110 Enabled	S110 Disabled
Amount	88 kB[1]	88 kB
CODE_R1_BASE	0x00016000	0x00016000
RAM	**S110 Enabled**	**S110 Disabled**
Amount	8 kB	8 bytes
RAM_R1_BASE	0x20002000	0x20000008
Call stack[2]	**S110 Enabled**	**S110 Disabled**
Maximum usage	1536 bytes	0 kB
Heap	**S110 Enabled**	**S110 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 21 *S110 Memory resource requirements*

10.2 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 33 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	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 P0	Open	Open
NA	0xE000E100	NVIC	Restricted[2]	Open

1. Blocked only when radio notification signal is enabled. See ***Table 24*** on page 34 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*

10.3 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*

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 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	-	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 25 *PPI channel and group availability*

10.5 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*

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. The maximum application interrupt latency is dependent on protocol stack activity as described in ***Section 11.2 "Processor availability"*** on page 37.

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 27 *Additional latency due to SoftDevice processing*

See ***Table 23*** on page 33 for open, blocked, and restricted peripherals.

11.2 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 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 13*** illustrates UpperStack activity (API calls) and LowerStack activity (Protocol events) and the time reserved/not reserved for those interrupts.

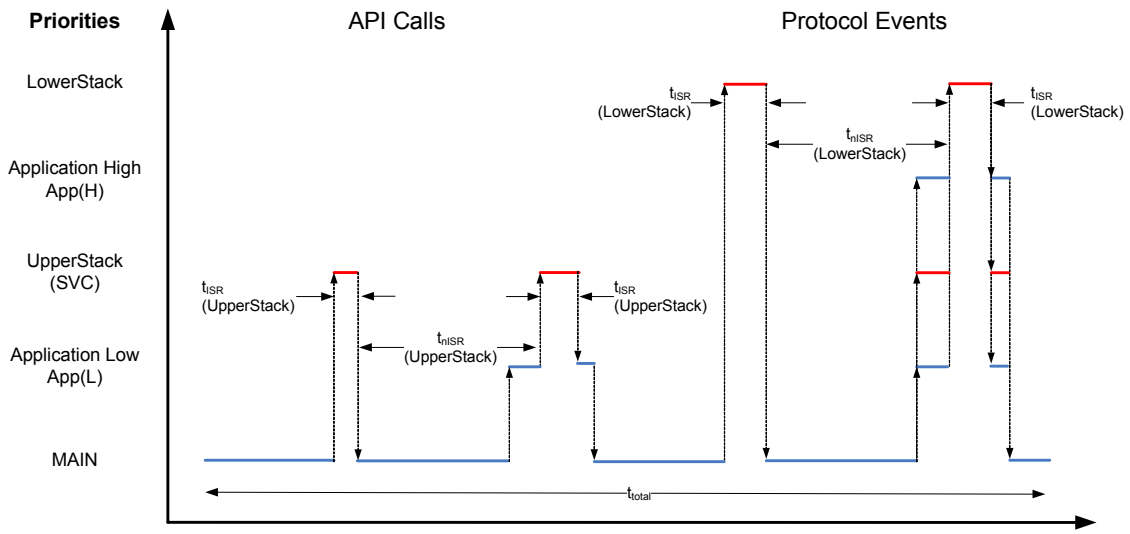

Figure 13 *UpperStack and LowerStack activity*

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

Table 28 *SoftDevice interrupt latency definitions*

11.3 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.

11.3.1 BLE peripheral advertising

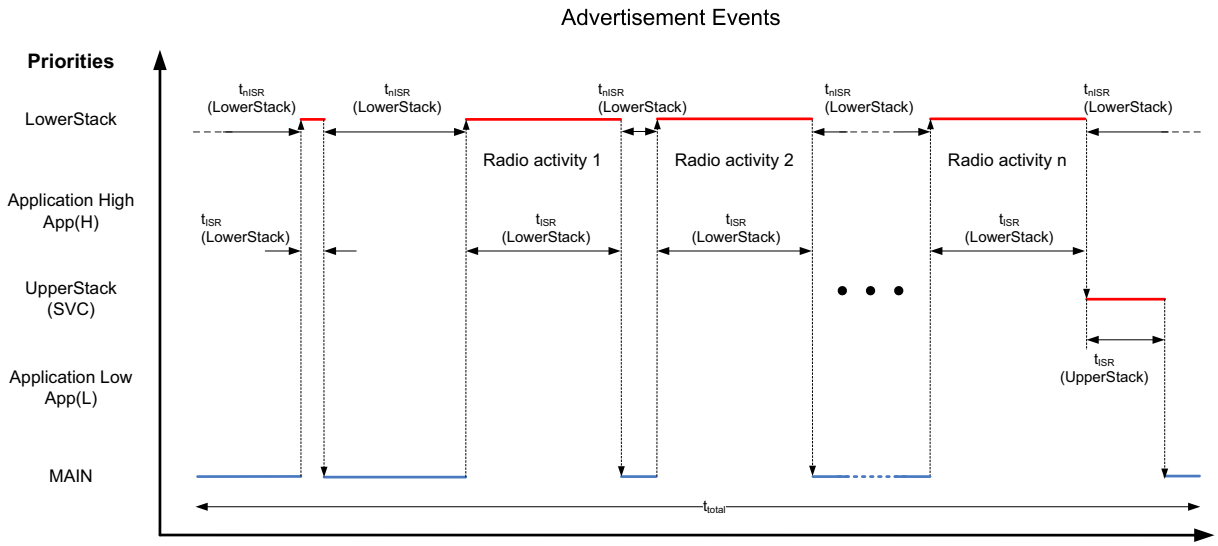

Figure 14 *Advertising*

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 29 *SoftDevice interrupt latency LowerStack for an advertising event*

11.3.2 BLE peripheral connection

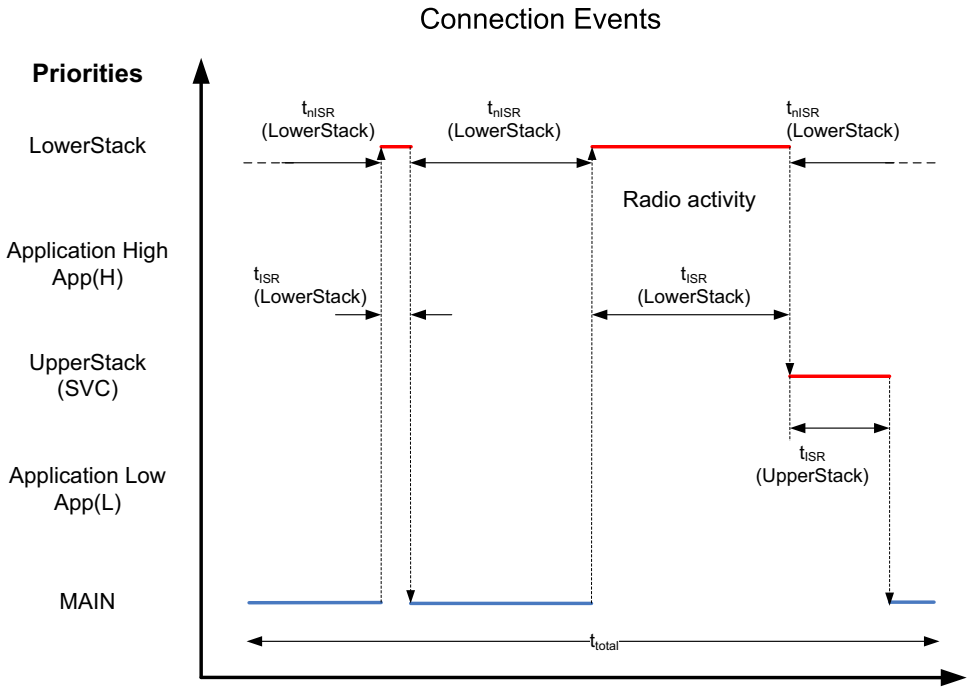

Figure 15 *Connection*

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 30 *SoftDevice interrupt latency LowerStack for a connection event*

The data in ***Table 30*** 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.

11.3.3 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 31 *SoftDevice interrupt latency - UpperStack*

11.3.4 CPU utilization in connection

Table 32 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 access the GATT attributes 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 32 *Processor usage and remaining availability for example BLE connection configurations*

11.4 Performance with Flash memory API and Concurrent Multiprotocol Timeslot API

The LowerStack interrupt is also used by the Flash memory API processing and by the Concurrent Multiprotocol Timeslot 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.

12 BLE data throughput

The maximum data throughput limits in ***Table 33*** 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 33 *L2CAP and GATT maximum data throughput*

13 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.

13.1 Connection event

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

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

Table 34 Connection event

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

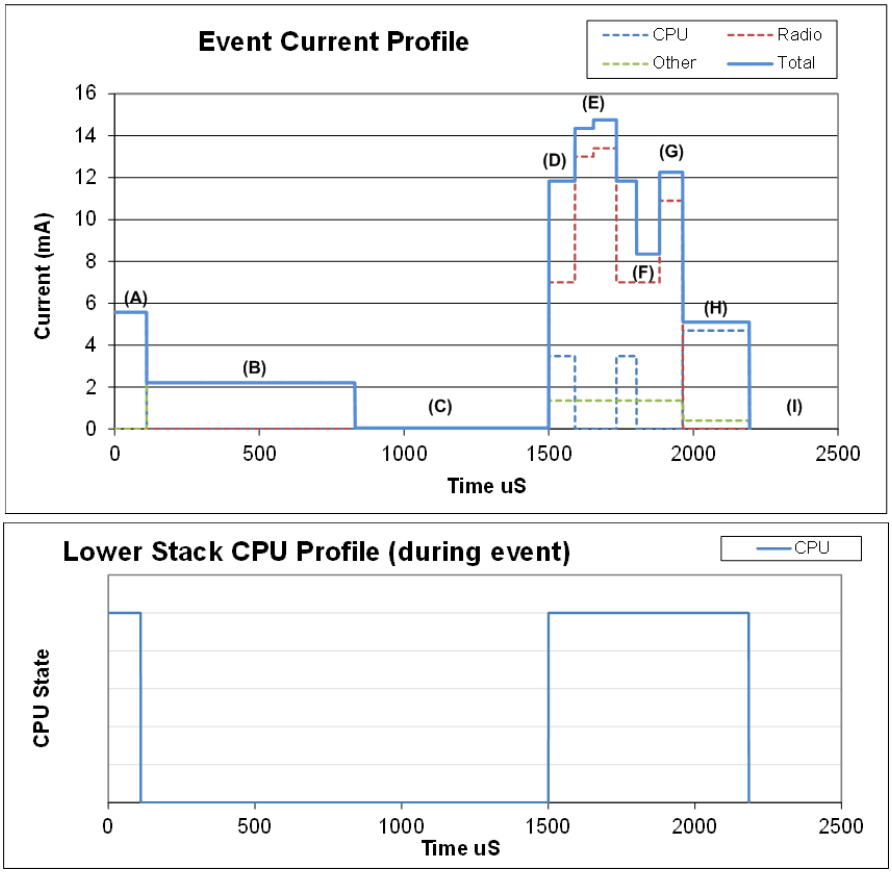

Figure 16 *Connection event*

13.2 Advertising event

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

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

Table 35 Advertising event

Note: I_{RC32k} should be substituted for I_{X32k} when using the 32.768k R_{COSC}.

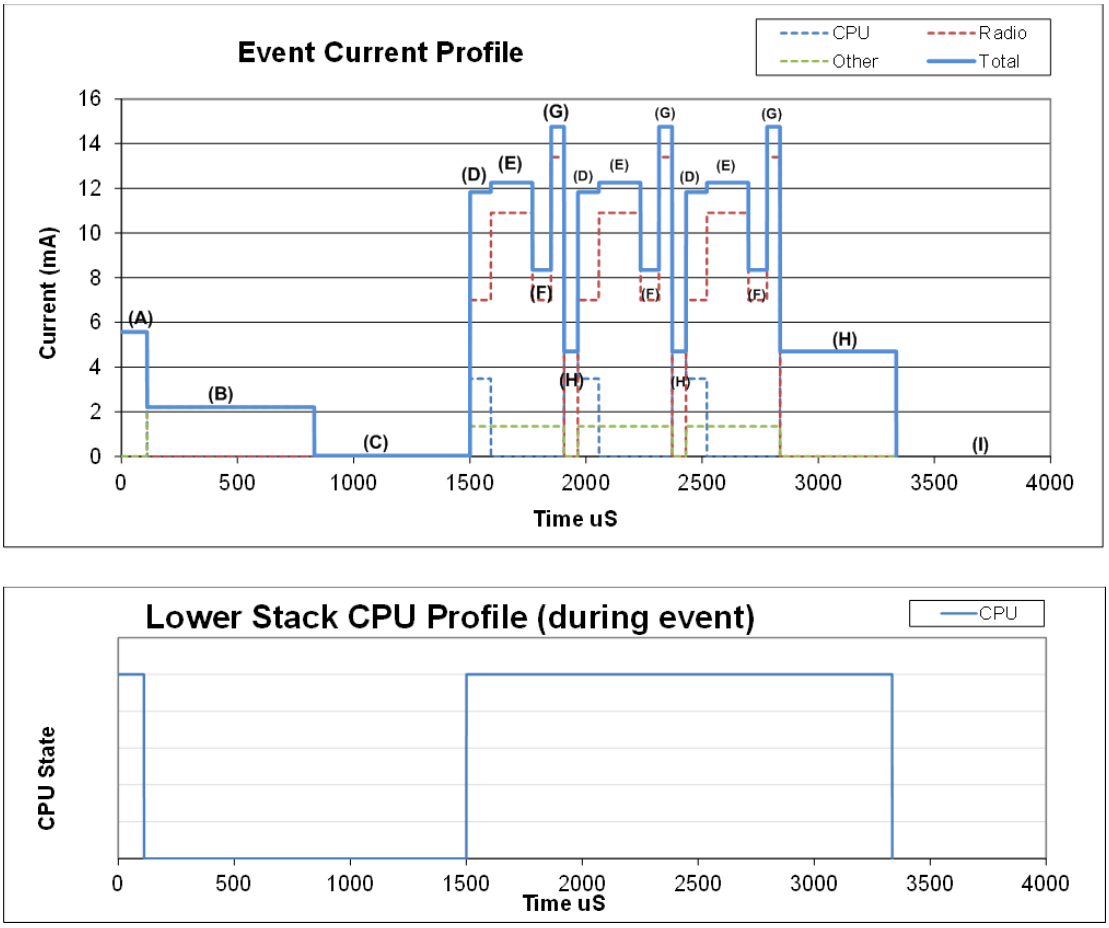

Figure 17 *Advertising event*

14 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 36***.

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 37***.

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 36 *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 37 *SoftDevice revision examples*

The test qualification levels are outlined in ***Table 38.***

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 38 *Test qualification levels*

14.1 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.