S113 SoftDevice

SoftDevice Specification

v1.1

NORDIC® SEMICONDUCTOR

Contents

Revision history

Date	Version	Description
September 2019	1.1	Updated for SoftDevice S113 version 7.0.1. Updated: • Bluetooth version support raised from 5.0 to 5.1.
July 2019	1.0	First release

Previous versions

PDF files for relevant previous versions are available here:

- S113 SoftDevice Specification v1.0 (deprecated)

1 S113 SoftDevice

The S113 SoftDevice is a *Bluetooth*® Low Energy peripheral protocol stack solution. It supports up to four peripheral connections with an additional broadcaster role running concurrently. The S113 SoftDevice integrates a Bluetooth Low Energy Controller and Host, and provides a full and flexible API for building Bluetooth Low Energy nRF52 System on Chip solutions.

Key features	Applications

• Bluetooth 5.1 compliant single-mode Bluetooth Low Energy protocol stack
  • Up to four peripheral connections and one broadcaster running concurrently
  • Configurable number of connections and connection properties
  • Configurable attribute table size
  • Custom UUID support
  • Link layer supporting LE 1M PHY and LE 2M PHY
  • LL Privacy
  • LE Data Packet Length Extension
  • ATT and SM protocols
  • L2CAP with LE Credit-based Flow Control
  • LE Secure Connections pairing model
  • GATT and GAP APIs
  • GATT Client and Server
  • Configurable ATT MTU
• Complementary nRF5 SDK including Bluetooth profiles and example applications
• Master boot record for over-the-air device firmware update
  • SoftDevice, application, and bootloader can be updated separately
• 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®- M4 project configuration for application development
• Support for concurrent and non-concurrent multiprotocol operation
  • Concurrent with the Bluetooth stack using Radio Timeslot API
  • Alternate protocol stack in application space
• Support for control of external power amplifiers and low noise amplifiers
	• Sports and fitness devices
	  • Sports watches
	  • Bike computers
	  • Fitness machines
	• Personal area networks
	  • Health and fitness sensor and monitoring devices
	  • Medical devices
	  • Key fobs and wrist watches
	• Home automation
	• AirFuel wireless charging
	• Remote control toys
	• Computer peripherals and I/O devices
	  • Mice
	  • Keyboards
	  • Multi-touch trackpads
	• Interactive entertainment devices
	  • Remote controls
	  • Gaming controllers

2 Documentation

Additional recommended reading for developing applications using the SoftDevice on an nRF52 *System on Chip (SoC)* includes the product specification, errata, compatibility matrix, and the Bluetooth Core Specification.

Documentation	Description
nRF52 Series found on Infocenter	• Product Specification: Contains a description of the hardware, peripherals, and electrical specifications specific to the *Integrated Circuit (IC)* • Errata: Contains information on anomalies related to the *IC* • Compatibility Matrix: Contains information on the compatibility between *IC* revisions, SoftDevices and SoftDevice Specifications, *Software Development Kit (SDK)*s, development kits, documentation, and *Qualified Design Identification (QDID)*s
Bluetooth Core Specification	The Bluetooth Core Specification version 5.1, Volumes 1, 3, 4, and 6, describe Bluetooth terminology which is used throughout the SoftDevice Specification

Table 1: Additional documentation

3 Product overview

The S113 SoftDevice is a precompiled and linked binary image implementing a Bluetooth 5.1 Low Energy protocol stack for the nRF52 Series of *SoC*s.

See the relevant compatibility matrix in Table 1: Additional documentation on page 7 for SoftDevice/*IC* compatibility information.

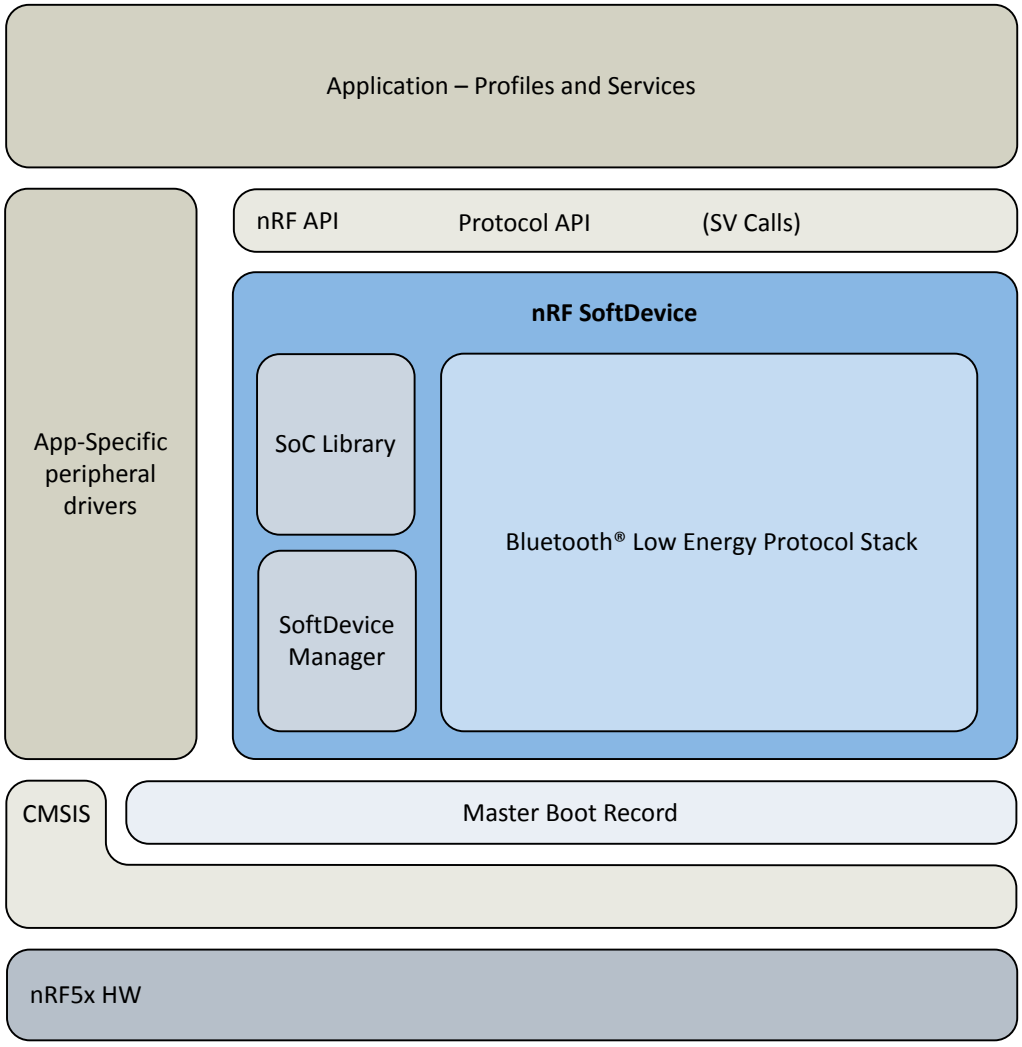

Figure 1: SoC application with the SoftDevice

Figure 1: SoC application with the SoftDevice on page 8 shows the nRF52 series software architecture. It includes the standard ARM *Cortex Microcontroller Software Interface Standard (CMSIS)* interface for nRF52 hardware, the MBR, profile and application code, application specific peripheral drivers, and a firmware module identified as a SoftDevice.

A SoftDevice consists of three main components:

- *SoC* Library: implementation and nRF *Application Programming Interface (API)* for shared hardware resource management (application coexistence)
- *SoftDevice Manager (SDM)*: implementation and nRF *API* for SoftDevice management (enabling/disabling the SoftDevice, etc.)
- Bluetooth 5.1 Low Energy protocol stack: implementation of protocol stack and *API*

The *API* is a set of standard C language functions and data types provided as a series of header files that give the application complete compiler and linker independence from the SoftDevice implementation. For more information, see Application programming interface on page 10.

The SoftDevice enables the application developer to develop their code as a standard ARM Cortex -M4 project without having the need to integrate with proprietary *IC* vendor software frameworks. This means that any ARM Cortex -M4-compatible toolchain can be used to develop Bluetooth Low Energy applications with the SoftDevice.

The SoftDevice can be programmed onto compatible nRF52 Series *IC*s during both development and production.

4 Application programming interface

The SoftDevice *API* is available to applications as a C programming language interface based on *Supervisor Call (SVC)*s and defined in a set of header files.

All variants of SoftDevices with the same version number are *API* compatible (see SoftDevice identification and revision scheme on page 78). In addition to a Protocol *API* enabling wireless applications, there is an nRF *API* that exposes the functionality of both the *SDM* and the *SoC* library.

> **Note:** When the SoftDevice is disabled, only a subset of the SoftDevice *API*s is available to the application (see SoftDevice API). For more information about enabling and disabling the SoftDevice, see SoftDevice enable and disable on page 12.

*SVC*s are software triggered interrupts conforming to a procedure call standard for parameter passing and return values. Each SoftDevice *API* call triggers a *SVC* interrupt. The SoftDevice *SVC* interrupt handler locates the correct SoftDevice function, allowing applications to compile without any *API* function address information at compile time. This removes the need for the application to link the SoftDevice. The header files contain all information required for the application to invoke the *API* functions with standard programming language prototypes. This *SVC* interface makes SoftDevice *API* calls thread-safe: they can be invoked from the application's different priority levels without additional synchronization mechanisms.

> **Note:** SoftDevice *API* functions can only be called from a lower interrupt priority level (higher numerical value for the priority level) than the *SVC* priority. For more information, see Interrupt priority levels on page 64.

4.1 Events - SoftDevice to application

Software triggered interrupts in a reserved IRQ are used to signal events from the SoftDevice to the application. The application is then responsible for handling the interrupt and for invoking the relevant SoftDevice functions to obtain the event data.

The application must respond to and process the SoftDevice events to ensure the SoftDevice functions properly. If events for Bluetooth Low Energy control procedures are not serviced, the procedures may time out and result in a link disconnection. If data received by the SoftDevice from the peer is not fetched in time, the internal SoftDevice data buffers may become full and no more data can be received.

For further details on how to implement the handling of these events, see the nRF5 Software Development Kit (nRF5 SDK) documentation.

4.2 Error handling

All SoftDevice *API* functions return a 32-bit error code. The application must check this error code to confirm whether a SoftDevice *API* function call was successful.

Unrecoverable failures (faults) detected by the SoftDevice will be reported to the application by a registered, fault handling callback function. A pointer to the fault handler must be provided by the application upon SoftDevice initialization. The fault handler is then used to notify of unrecoverable errors, and the type of error is indicated as a parameter to the fault handler.

The following types of faults can be reported to the application through the fault handler:

- SoftDevice assertions

- Attempts by the application to perform unallowed memory accesses, either against SoftDevice memory protection rules or to protected peripheral configuration registers at runtime

The fault handler callback is invoked by the SoftDevice in HardFault context with all interrupts disabled.

5 SoftDevice Manager

The *SDM API* allows the application to manage the SoftDevice on a top level. It controls the SoftDevice state and configures the behavior of certain SoftDevice core functionality.

When enabling the SoftDevice, the *SDM* configures the following:

- The LFCLK source. See Clock source on page 12.
- The interrupt management. See SoftDevice enable and disable on page 12.
- The embedded protocol stack.

In addition, it enables the SoftDevice RAM and peripheral protection. See Memory isolation and runtime protection on page 13.

Detailed documentation of the *SDM API* is made available with the *SDK*s.

5.1 SoftDevice enable and disable

When the SoftDevice is not enabled, the Protocol *API* and parts of the *SoC* library *API* are not available to the application.

When the SoftDevice is not enabled, most of the *SoC*'s resources are available to the application. However, the following restrictions apply:

- *SVC* numbers 0x10 to 0xFF are reserved.
- SoftDevice program (flash) memory is reserved.
- A few bytes of RAM are reserved. See Memory resource map and usage on page 53 for more details.

Once the SoftDevice has been enabled, more restrictions apply:

- Some RAM will be reserved. See Memory isolation and runtime protection on page 13 for more details.
- Some peripherals will be reserved. See Hardware peripherals on page 18 for more details.
- Some of the peripherals that are reserved will have a *SoC* library interface.
- Interrupts from the reserved SoftDevice peripherals will not be forwarded to the application. See Interrupt forwarding to the application on page 63 for more details.
- The reserved peripherals are reset upon SoftDevice disable.
- `nrf_nvic_` functions must be used instead of *CMSIS* `NVIC_` functions for safe use of the SoftDevice.
- SoftDevice activity in high priority levels may interrupt the application, increasing the maximum interrupt latency. For more information, see Interrupt model and processor availability on page 63.

5.2 Clock source

The SoftDevice can use one of two available LFCLK sources: the internal RC Oscillator, or external Crystal Oscillator.

The application must provide the selected clock source and some clock source characteristics, such as accuracy, when it enables the SoftDevice. The *SDM* is responsible for configuring the LFCLK source and for keeping it calibrated when the RC oscillator is the selected clock source.

If the SoftDevice is configured with the internal RC oscillator clock option, periodic clock calibration is required to adjust the RC oscillator frequency. Additional calibration is required for temperature changes

of more than 0.5°C. See the relevant product specification (Table 1: Additional documentation on page 7) for more information. The SoftDevice will perform this function automatically. The application may choose how often the SoftDevice will make a measurement to detect temperature change. The application must consider how frequently significant temperature changes are expected to occur in the intended environment of the end product. It is recommended to use a temperature polling interval of 4 seconds, and to force clock calibration every second interval (`.rc_ctiv=16`, `.rc_temp_ctiv=2`).

Extended RC calibration is enabled by default when the RC oscillator is used. In this feature, the SoftDevice as a peripheral can detect if the clock has drifted and then calibrate the RC oscillator if necessary. This calibration is in addition to the periodic calibration. If using only peripheral connections, the periodic calibration can then be configured with a much longer interval as the peripheral will be able to detect and adjust automatically to clock drift and calibrate when required. When the Extended RC calibration is enabled, the SoftDevice as a peripheral will try to increase the receive window if two consecutive packets are not received. If it turns out that the packets were missed due to clock drift, the RC oscillator calibration is started. Extended RC calibration can be disabled using the BLE option API.

5.3 Power management

The SoftDevice implements a simple to use SoftDevice Power API for optimized power management.

The application must use this *API* when the SoftDevice is enabled to ensure correct function. When the SoftDevice is disabled, the application must use the hardware abstraction (*CMSIS*) interfaces for power management directly.

When waiting for application events using the *API*, the CPU goes to an IDLE state whenever the SoftDevice is not using the CPU, and interrupts handled directly by the SoftDevice do not wake the application. Application interrupts will wake the application as expected. When going to system OFF, the *API* ensures the SoftDevice services are stopped before powering down.

5.4 Memory isolation and runtime protection

The SoftDevice data memory and peripherals can be sandboxed and runtime protected to prevent the application from interfering with the SoftDevice execution, ensuring robust and predictable performance. Sandboxing is available only if the SoftDevice is run on a chip with a *Memory Watch Unit (MWU)*.

Sandboxing[1] and runtime protection can allow memory access violations to be detected at development time. This ensures that developed applications will not inadvertently interfere with the correct functioning of the SoftDevice.

Sandboxing is enabled by default when the SoftDevice is enabled, and disabled when the SoftDevice is disabled. When enabled, SoftDevice RAM and peripheral registers are protected against write access by the application. The application will have read access to SoftDevice RAM and peripheral registers.

The program memory is divided into two regions at compile time. The SoftDevice Flash Region is located between addresses 0x00000000 and `APP_CODE_BASE - 1` and is occupied by the SoftDevice. The Application Flash Region is located between the addresses `APP_CODE_BASE` and the last valid address in the flash memory and is available to the application.

The RAM is split into two regions, which are defined at runtime, when the SoftDevice is enabled. The SoftDevice RAM Region is located between the addresses 0x20000000 and `APP_RAM_BASE - 1` and is used by the SoftDevice. The Application RAM Region is located between the addresses `APP_RAM_BASE` and the top of RAM and is available to the application.

The following figure presents an overview of the regions.

[1] A sandbox is a set of memory access restrictions imposed on the application.

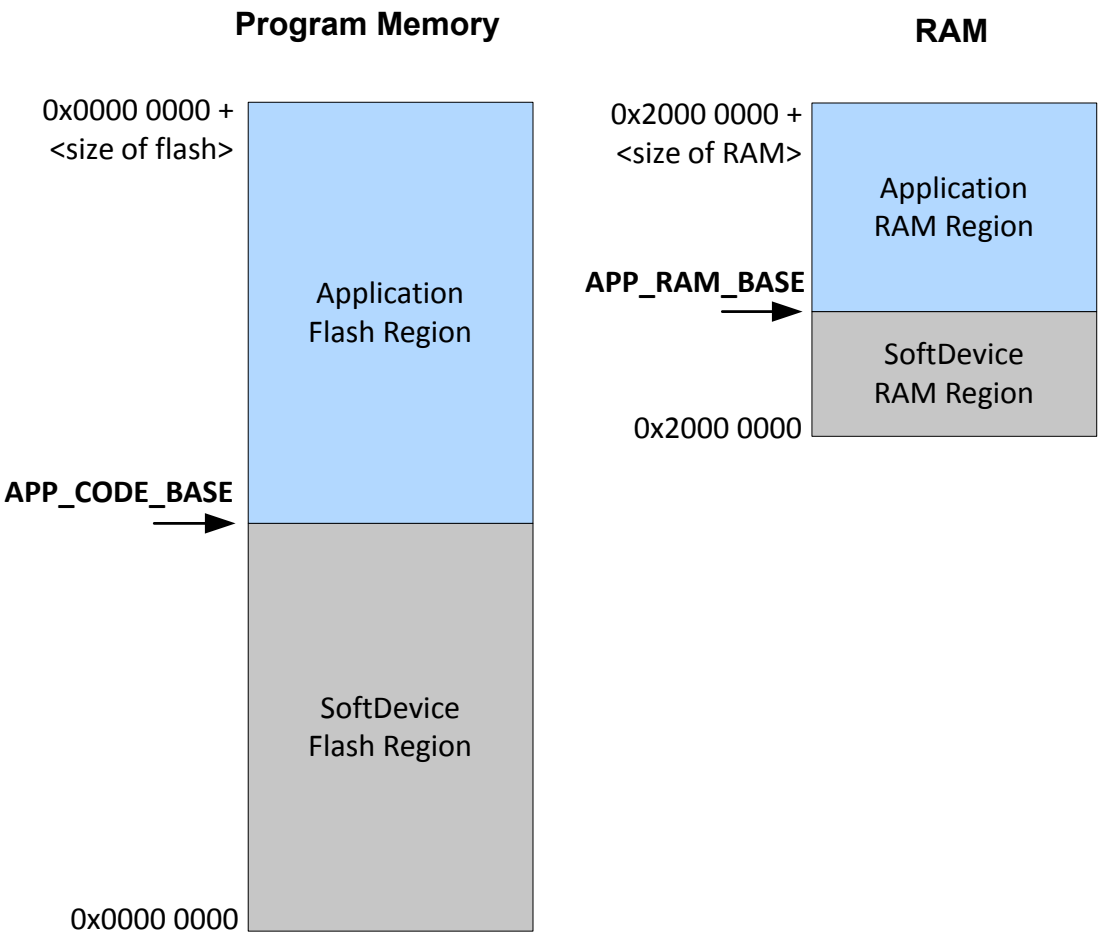

Figure 2: Memory region designation

The SoftDevice uses a fixed amount of flash (program) memory. By contrast, the size of the SoftDevice RAM Region depends on whether the SoftDevice is enabled or not, and on the selected Bluetooth Low Energy protocol stack configuration. See Role configuration on page 56 for more details.

The amount of flash and RAM available to the application is determined by region size (kilobytes or bytes) and the `APP_CODE_BASE` and `APP_RAM_BASE` addresses which are the base addresses of the application code and RAM, respectively. The application code must be located between `APP_CODE_BASE` and <size of flash>. The application variables must be allocated in an area inside the Application RAM Region, located between `APP_RAM_BASE` and <size of RAM>. This area shall not overlap with the allocated RAM space for the call stack and heap, which is also located inside the Application RAM Region.

The program code address range of an example application:

`APP_CODE_BASE` ≤ Program ≤ <size of flash>

RAM address range of example application assuming call stack and heap location as shown in Figure 18: Memory resource map on page 54:

`APP_RAM_BASE` ≤ RAM ≤ (0x2000 0000 + <size of RAM>) - (<Call Stack> + <Heap>)

Sandboxing protects the SoftDevice RAM Region so that it cannot be written to by the application at runtime. Violation of sandboxing rules, for example, an attempt to write to the protected SoftDevice memory, will result in the triggering of a fault (unrecoverable error handled by the application). See Error handling on page 10 for more information.

When the SoftDevice is disabled, all RAM, with the exception of a few bytes, is available to the application. See Memory resource map and usage on page 53 for more details. When the SoftDevice is enabled, RAM up to `APP_RAM_BASE` will be used by the SoftDevice and will be write protected.

The typical location of the call stack for an application using the SoftDevice is in the upper part of the Application RAM Region, so the application can place its variables from the end of the SoftDevice RAM Region (`APP_RAM_BASE`) to the beginning of the call stack space.

Note:

- The location of the call stack is communicated to the SoftDevice through the contents of the *Main Stack Pointer (MSP)* register.
- Do not change the value of *MSP* dynamically (i.e. never set the *MSP* register directly).
- The RAM located in the SoftDevice RAM Region will be overwritten once the SoftDevice is enabled.
- The SoftDevice RAM Region will not be cleared or restored to default values after disabling the SoftDevice, so the application must treat the contents of the region as uninitialized memory.

6 System on Chip library

The coexistence of the Application and SoftDevice with safe sharing of common *SoC* resources is ensured by the *SoC* library.

The features described in the following table are implemented by the *SoC* library and can be used for accessing the shared hardware resources when the SoftDevice is enabled.

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	Wrapper functions for the *CMSIS* NVIC functions provided by ARM. **Note:** To ensure reliable usage of the SoftDevice you must use the wrapper functions when the SoftDevice is enabled.
Rand	Provides random numbers from the hardware random number generator.

Power	Access to POWER block configuration:
	• 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:
	  • DISABLED
	  • ENABLED

Clock	Access to CLOCK block configuration. 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 application event.
PPI	Configuration interface for *Programmable Peripheral Interconnect (PPI)* channels and groups reserved for an application.[2]
Radio Timeslot API	Schedule other radio protocol activity, or periods of radio inactivity. For more information, see Concurrent multiprotocol implementation using the Radio Timeslot API on page 24.
Radio Notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See Radio Notification signals on page 42.
Block Encrypt (ECB)	Safe use of 128-bit AES encrypt HW accelerator
Event API	Fetch asynchronous events generated by the *SoC* library.

Feature	Description
Flash memory API	Application access to flash write, erase, and protect. Can be safely used during all protocol stack states.[2] See Flash memory API on page 22.
Temperature	Application access to the temperature sensor

Table 2: SoC features

[2] This can also be used when the SoftDevice is disabled.

7 System on Chip resource requirements

This section describes how the SoftDevice, including the MBR, uses the *SoC* resources. The SoftDevice requirements are shown for when the SoftDevice is enabled and disabled.

The SoftDevice and MBR (see Master boot record and bootloader on page 49) are designed to be installed on the nRF *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 *SoC* including RAM and hardware peripherals like the radio. For the amount of RAM required by the SoftDevice, see SoftDevice memory usage on page 53.

7.1 Hardware peripherals

The SoftDevice requires certain hardware peripherals to function correctly. The availability of these hardware peripherals to the application depends on whether the SoftDevice is enabled or disabled.

The access types listed in the following table are used to categorize the availability of the hardware peripherals to the application. The application has access to most hardware peripherals. The exceptions are listed in Table 4: Hardware peripherals with limited availability to the application on page 18.

Access type	Definition
Restricted	The hardware peripheral is used by the SoftDevice and is outside the application sandbox. When the SoftDevice is enabled, it shall only be accessed through the SoftDevice API. Through this *API*, the application has limited access.
Blocked	The hardware peripheral is used by the SoftDevice and is outside the application sandbox. The application has no access. Interrupts from blocked peripherals are forwarded to the SoftDevice by the MBR and are not available to the application, even inside a Radio Timeslot API timeslot.
Open	The hardware peripheral is not used by the SoftDevice. The application has full access.

Table 3: Hardware access type definitions

Instance	Access SoftDevice enabled	Access SoftDevice disabled
CLOCK	Restricted	Open
POWER	Restricted	Open
BPROT	Restricted	Open

Instance	Access SoftDevice enabled	Access SoftDevice disabled
RADIO	Blocked[3]	Open
TIMER0	Blocked[3]	Open
RTC0	Blocked	Open
TEMP	Restricted	Open
RNG	Restricted	Open
ECB	Restricted	Open
CCM	Blocked[4]	Open
AAR	Blocked[4]	Open
EGU1/SWI1/Radio Notification	Restricted[5]	Open
EGU5/SWI5	Blocked	Open
NVMC	Restricted	Open
MWU	Restricted[6]	Open
FICR	Blocked	Blocked
UICR	Restricted	Open
NVIC	Restricted[7]	Open

Table 4: Hardware peripherals with limited availability to the application

Note: Some of the peripherals in this table are not present on all devices. For a complete overview of the peripherals on a device, see the relevant product specification (Table 1: Additional documentation on page 7).

7.2 Application signals – software interrupts

Software interrupts are used by the SoftDevice to signal events to the application.

[3] The peripheral is available to the application through the Radio Timeslot API. See Concurrent multiprotocol implementation using the Radio Timeslot API on page 24.

[4] The peripheral is available to the application during a Radio Timeslot API timeslot. See Concurrent multiprotocol implementation using the Radio Timeslot API on page 24.

[5] Blocked only when Radio Notification signal is enabled. See Application signals – software interrupts on page 19 for SWI allocation.

[6] See sections Memory isolation and runtime protection on page 13 and Peripheral runtime protection on page 21 for limitations on the use of MWU when the SoftDevice is enabled.

[7] Not protected. For robust system function, the application program must comply with the restriction and use the SoftDevice NVIC API for configuration when the SoftDevice is enabled.

SWI	Peripheral ID	Interrupt priority	SoftDevice Signal
0	20	-	Unused by the SoftDevice and available to the application.
1	21	6	Radio Notification. The interrupt priority can optionally be configured through the SoftDevice NVIC API.
2	22	6	SoftDevice Event Notification. The interrupt priority can optionally be configured through the SoftDevice NVIC API. The interrupt will be set to PENDING state by the SoftDevice on SoftDevice Event Notification, but also the application may set it to PENDING state.
3	23	-	Unused by the SoftDevice and available to the application.
4	24	-	Reserved for future use.
5	25	4	SoftDevice processing - not user configurable.

Table 5: Allocation of software interrupt vectors to SoftDevice signals

7.3 Programmable peripheral interconnect

A set of *PPI* channels and groups may be configured using the *PPI API* in the *SoC* library.

This *API* is available both when the SoftDevice is disabled and when it is enabled. It is also possible to configure the *PPI*s using the *CMSIS* directly when the SoftDevice is disabled.

When the SoftDevice is disabled, all *PPI* channels and groups are available to the application.

When the SoftDevice is enabled, some of the *PPI* channels and groups are reserved by the SoftDevice. The application must therefore not change the configuration of these *PPI* channels or groups when the SoftDevice is enabled. Failing to comply with this will cause the SoftDevice to not operate properly.

The *PPI* channels and groups that are reserved by the SoftDevice when enabled are defined in `nrf_soc.h`.

7.4 SVC number ranges

Application programs and SoftDevices use certain *SVC* numbers.

The table below 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 6: SVC number allocation

7.5 Peripheral runtime protection

To prevent the application from accidentally disrupting the protocol stack in any way, the application sandbox also protects the peripherals used by the SoftDevice.

Protected peripheral registers are readable by the application. An attempt to perform a write to a protected peripheral register will result in a Hard Fault. See Error handling on page 10 for more details on faults due to unallowed memory access. The peripherals are only protected when the SoftDevice is enabled; otherwise, they are available to the application. See Table 4: Hardware peripherals with limited availability to the application on page 18 for an overview of the peripherals with access restrictions due to the SoftDevice.

7.6 External and miscellaneous requirements

For correct operation of the SoftDevice, it is a requirement that the crystal oscillator (HFXO) startup time is less than 1.5 ms.

The external clock crystal and other related components must be chosen accordingly. Data for the crystal oscillator input can be found in the relevant *SoC* product specification (Table 1: Additional documentation on page 7).

When the SoftDevice is enabled, the SEVONPEND flag in the SCR register of the CPU shall only be changed from main or low interrupt level (priority not higher than 4). Otherwise, the behavior of the SoftDevice is undefined and the SoftDevice might malfunction.

8 Flash memory API

The *SoC* flash memory *API* provides the application with flash write, flash erase, and flash protect support through the SoftDevice. Asynchronous flash memory operations can be safely performed during active Bluetooth Low Energy connections using the Flash memory *API* of the *SoC* library.

The flash memory accesses are scheduled to not disturb radio events. See Flash API timing on page 61 for details. If the protocol radio events are in a critical state, flash memory accesses may be delayed for a long period resulting in a time-out event. In this case, NRF_EVT_FLASH_OPERATION_ERROR will be returned in the application event handler. If this happens, retry the flash memory operation. Examples of typical critical phases of radio events include connection setup, connection update, disconnection, and impending supervision time-out.

The probability of successfully accessing the flash memory decreases with increasing scheduler activity (i.e. radio activity and timeslot activity). With long connection intervals, there will be a higher probability of accessing flash memory successfully. Use the guidelines in Table 7: Behavior with Bluetooth Low Energy traffic and concurrent flash write/erase on page 22 to improve the probability of flash operation success.

A flash write must be made in chunks smaller or equal to the flash page size. Make flash writes in as small chunks as possible to increase the probability of success and reduce the chance of affecting Bluetooth Low Energy performance. The table below assumes a flash write size of four bytes. LE 1M PHY is assumed unless another PHY is specified.

The time required to do a flash memory operation using the flash memory API depends on which *IC* is being used. For the exact timing numbers, see the relevant product specification (Table 1: Additional documentation on page 7). In the table below, a flash page erase is assumed to last for 90 ms.

Bluetooth Low Energy activity	Flash write/erase
High Duty cycle directed advertising	Does not allow flash operation while advertising is active (maximum 1.28 seconds). In this case, retrying flash operation will only succeed after the advertising activity has finished.
All possible Bluetooth Low Energy roles running concurrently (connections as a Peripheral and Advertiser)	Low to medium probability of flash operation success Probability of success increases with: • Configurations with shorter event lengths • Lower data traffic • Increase in connection interval and advertiser interval
1 connection as a Peripheral The active connection fulfills the following criteria: • Supervision time-out > 6 x connection interval • Connection interval ≥ 25 ms	High probability of flash operation success

Bluetooth Low Energy activity	Flash write/erase
4 connections as a Peripheral All active connections fulfill the following criteria: • Supervision time-out > 6 x connection interval • Connection interval ≥ 115 ms	Medium to high probability of flash operation success. The scheduling of connections as Peripheral is done by the peer devices. The Peripheral does not influence this scheduling, which means that the connection events may collide and result in flash operations being blocked. With multiple connections as Peripheral, choose connection intervals and connection event lengths in a way that leaves enough free time to handle collisions and other activities.
Connectable Undirected Advertising Nonconnectable Advertising Scannable Advertising Connectable Low Duty Cycle Directed Advertising	High probability of flash operation success
No Bluetooth Low Energy activity	Flash operation will always succeed

Table 7: Behavior with Bluetooth Low Energy traffic and concurrent flash write/erase

9 Multiprotocol support

Multiprotocol support allows developers to implement their own 2.4 GHz proprietary protocol in the application both when the SoftDevice is not in use (non-concurrent) and while the SoftDevice protocol stack is in use (concurrent). For concurrent multiprotocol implementations, the Radio Timeslot API allows the application protocol to safely schedule radio usage between Bluetooth Low Energy events.

9.1 Non-concurrent multiprotocol implementation

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. The SoftDevice may be disabled and enabled without resetting the application in order to switch between a proprietary protocol stack and Bluetooth communication.

9.2 Concurrent multiprotocol implementation using the Radio Timeslot API

The Radio Timeslot API allows the nRF52 device to be part of a network using the SoftDevice protocol stack and an alternative network of wireless devices at the same time.

The Radio Timeslot (or, simply Timeslot) feature gives the application access to the radio and other restricted peripherals during defined time intervals, denoted as timeslots. The Timeslot feature achieves this by cooperatively scheduling 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 of the SoftDevice. It can also be used to suppress SoftDevice radio activity and to 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 request is granted, the application has exclusive and real-time access to the normally blocked RADIO, TIMER0, CCM, and AAR peripherals and can use these freely for the duration (length) of the timeslot. See Table 3: Hardware access type definitions on page 18 and Table 4: Hardware peripherals with limited availability to the application on page 18.

9.2.1 Request types

There are two types of Radio Timeslot requests, *earliest possible* Timeslot requests and *normal* Timeslot requests.

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.

> **Note:** The first request in a session must always be *earliest possible* to create the timing reference point for later timeslots.

Timeslots may also be requested at a given time (*normal*). 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.

The application may also request to extend an ongoing 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.

9.2.2 Request priorities

Radio Timeslots can be requested at either high or normal priority, indicating how important it is for the application to access the specified peripherals. A Timeslot request can only be blocked or canceled due to an overlapping SoftDevice activity that has a higher scheduling priority.

9.2.3 Timeslot length

A Radio Timeslot is requested for a given length. Ongoing timeslots have the possibility to be extended.

The length of the timeslot is specified by the application in the Timeslot request and ranges from 100 µs to 100 ms. Longer continuous timeslots can be achieved by requesting to extend the current timeslot. A timeslot may be extended multiple times, as long as its duration does not extend beyond the time limits set by other SoftDevice activities, and up to a maximum length of 128 seconds.

9.2.4 Scheduling

The SoftDevice includes a scheduler which manages radio timeslots and priorities and sets up timers to grant timeslots.

Whether a Timeslot request is granted and access to the peripherals is given is determined by the following factors:

- The time the request is made
- The exact time in the future the timeslot is requested for
- The desired priority level of the request
- The length of the requested timeslot

Timeslot API timing on page 61 explains how timeslots are scheduled. Timeslots requested at high priority will cancel other activities scheduled at lower priorities in case of a collision. 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 Radio Timeslot interface as for SoftDevice internal activities. See section Radio Notification signals on page 42 for more information. If Radio Notifications are enabled, Radio Timeslots will be notified.

9.2.5 High-frequency clock configuration

The application can request the SoftDevice to guarantee that the HFCLK source is set to the external crystal and that it is ramped up and stable before the start of the timeslot.

If the application requests the SoftDevice to have the external high-frequency crystal ready by the start of the timeslot, the SoftDevice will handle all the enabling and disabling of the crystal. The application does not need to disable the crystal at the end of the timeslot. The SoftDevice will disable the crystal after the end of the timeslot unless the SoftDevice needs to use it within a short period of time after the end of the timeslot. In that case, the SoftDevice will leave the crystal running.

If the application does not request the SoftDevice to have the external high-frequency crystal ready by the start of the timeslot, then the application must not use the RADIO during the timeslot and must take into consideration that the HFCLK source is inaccurate during the timeslot unless the application itself makes

sure that the crystal is ramped up and ready at the start of the timeslot. If the application starts the crystal before or during the timeslot, it is the responsibility of the application to disable it again.

9.2.6 Performance considerations

The Radio Timeslot API shares core peripherals with the SoftDevice, and application-requested timeslots are scheduled along with other SoftDevice activities. Therefore, the use of the Timeslot feature may influence the performance of the SoftDevice.

The configuration of the SoftDevice should be considered when using the Radio Timeslot API. A configuration which uses more radio time for native protocol operation will reduce the available time for serving timeslots and result in a higher risk of scheduling conflicts.

All Timeslot requests should use the lowest priority to minimize disturbances to other activities. See Table 28: Scheduling priorities on page 58 for the scheduling priorities of the different 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 by using normal priority. By using the highest priority available to the Timeslot API, non-critical SoftDevice radio protocol traffic may be affected. The SoftDevice radio protocol has access to higher priority levels than the application. These levels will be used for important radio activity, for instance when the device is about to lose a connection.

See Scheduling on page 57 for more information on how priorities work together with other modules like the Bluetooth Low Energy protocol stack, the Flash API etc.

Timeslots should be kept as short as possible in order to minimize the impact on the overall performance of the device. Requesting a short timeslot will make it easier for the scheduler to fit in between other scheduled activities. The timeslot may later be extended. This will not affect other sessions, as it is only possible to extend a timeslot if the extended time is unreserved.

It is important to ensure that a timeslot has completed its outstanding operations before the time it is scheduled to end (based on its starting time and requested length); otherwise, the SoftDevice behavior is undefined and may result in an unrecoverable fault.

9.2.7 Radio Timeslot API

This section describes the calls, events, signals, and return actions of the Radio Timeslot API.

A Timeslot session is opened and closed using *API* calls. Within a session, there is a *API* call to request timeslots. For communication back to the application, the Timeslot feature will generate events and signals. The generated events are handled by the normal application event handler, while the Timeslot signals 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.

9.2.7.1 API calls

The S113 SoftDevice provides *API* functions for handling radio timeslots.

The *API* functions are defined in the following table.

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

Table 8: API calls

9.2.7.2 Radio Timeslot events

Events come from the SoftDevice scheduler and are used for Radio Timeslot session management.

Events are received in the application event handler callback function, which will typically be run in an application interrupt. For more information, see Events - SoftDevice to application on page 10. The events are defined in the following table.

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 canceled due to overlapping activity of higher priority.
NRF_EVT_RADIO_SIGNAL_ CALLBACK_INVALID_RETURN	Signal handler: The last signal handler return value contained invalid parameters and the timeslot was ended.

Table 9: Radio Timeslot events

9.2.7.3 Radio Timeslot signals

Signals come from the peripherals and arrive within a Radio Timeslot.

Signals are received in a signal handler callback function that the application must provide. The signal handler runs in interrupt priority level 0, which is the highest priority in the system, see section Interrupt priority levels on page 64.

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 nRF52 Reference Manual.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_TIMER0	Timer interrupt. For more information, see chapter Timer/counter (TIMER) in the nRF52 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 10: Radio Timeslot signals

9.2.7.4 Signal handler return actions

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

Signal	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 11: Signal handler action return values

9.2.7.5 Ending a timeslot in time

The application is responsible for keeping track of timing within the Radio Timeslot and for ensuring that the application's use of the peripherals does not last for longer than the granted timeslot length.

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.

Note: The scheduler uses the LFCLK source for time calculations when scheduling events. If the application uses a TIMER (sourced from the current HFCLK source) to calculate and signal the end of a timeslot, it must account for the possible clock drift between the HFCLK source and the LFCLK source.

9.2.7.6 Signal handler considerations

The signal handler runs at interrupt priority level 0, which is the highest priority. Therefore, it cannot be interrupted by any other activity.

Since the signal handler runs at a higher interrupt priority (lower numerical value for the priority level) than the *SVC* calls (see Interrupt priority levels on page 64), *SVC* calls are not available in the signal handler.

Note: 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 real time signal handling. When the application has handled the signal, it can exit the signal handler and wait for the next signal if it wants to do other (less time critical) processing at lower interrupt priority (higher numerical value for the priority level) while waiting.

9.3 Radio Timeslot API usage scenarios

In this section, several Radio Timeslot API usage scenarios are provided with descriptions of the sequence of events within them.

9.3.1 Complete session example

This section describes a complete Radio Timeslot session.

Figure 3: Complete Radio Timeslot session example on page 30 shows a complete Timeslot session. In this case, only timeslot requests from the application are being scheduled, and 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 possible*). The SoftDevice schedules the timeslot. At the start of the timeslot, the SoftDevice calls the application signal handler 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 indicating 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.

All subsequent timeslots are 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. Therefore, 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.

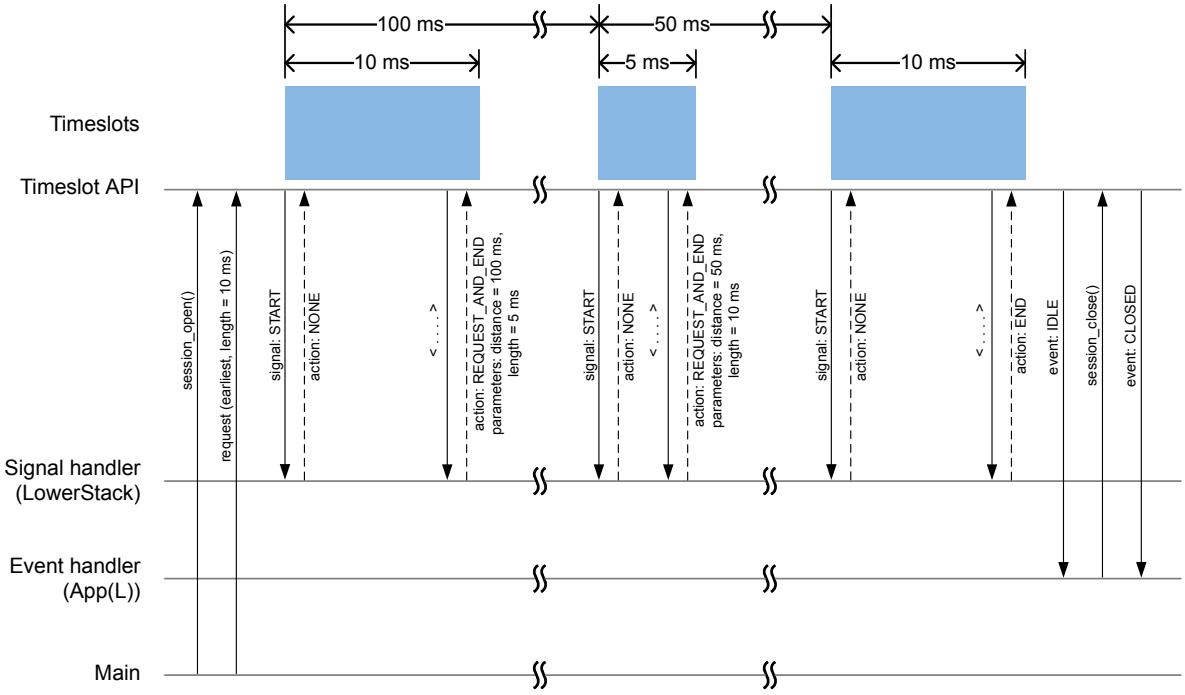

Figure 3: Complete Radio Timeslot session example

LowerStack denotes the interrupt level for SoftDevice *API* calls and non-time-critical processing, and App(L) denotes the selected low-priority application interrupt level. See Interrupt priority levels on page 64 for the available interrupt levels.

9.3.2 Blocked timeslot scenario

Radio Timeslot requests may be blocked due to an overlap with activities already scheduled by the SoftDevice.

Figure 4: Blocked timeslot scenario on page 31 shows a situation in the middle of a session where a requested timeslot cannot be scheduled. At the end of the first timeslot illustrated here, 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 for a later point in time. This request succeeds (it does not collide with anything), and a new timeslot is eventually scheduled.

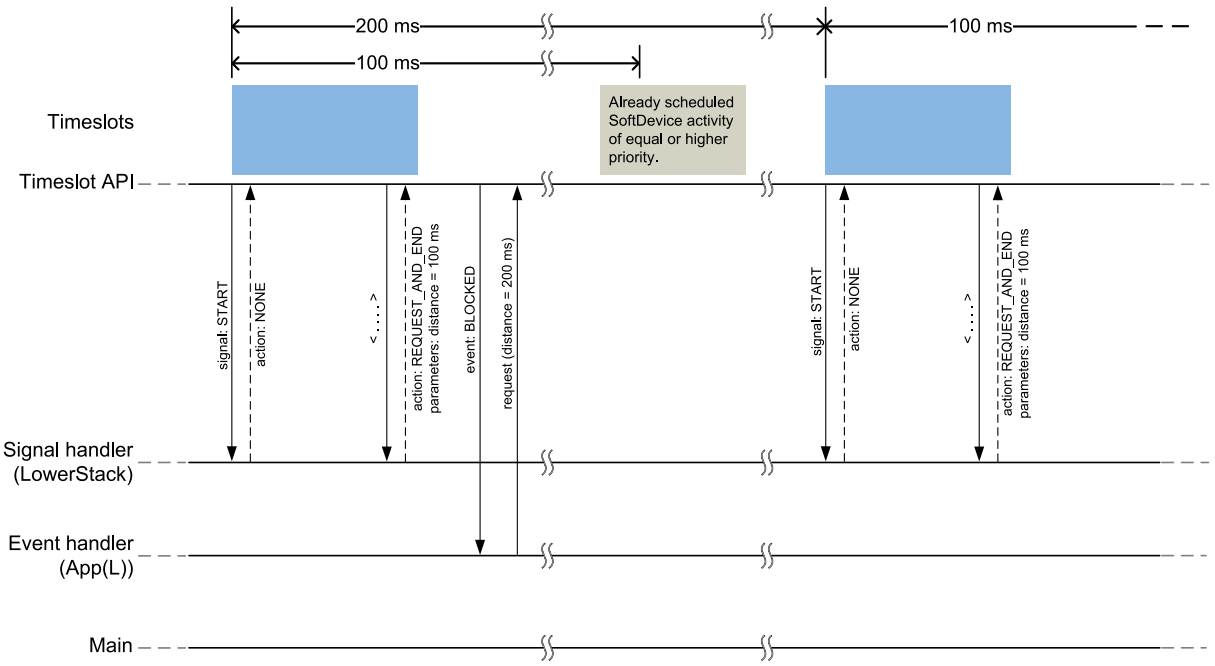

Figure 4: Blocked timeslot scenario

9.3.3 Canceled timeslot scenario

Situations may occur in the middle of a session where a requested and scheduled application radio timeslot is being revoked.

Figure 5: Canceled timeslot scenario on page 32 shows a situation in the middle of a session where a requested and scheduled application timeslot is being revoked. The upper part of the figure 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 started yet, as its starting time is in the future. The lower part of the figure shows the situation some time later.

In the meantime, the SoftDevice has requested some reserved time for a higher priority activity that overlaps with the scheduled application timeslot. To accommodate the higher priority request, the application timeslot is removed from the schedule and, instead, the higher priority SoftDevice activity is scheduled. The application is notified about this by a CANCELED event to the application event handler. The application then makes a new request at a later point in time. That request succeeds (it does not collide with anything), and a new timeslot is eventually scheduled.

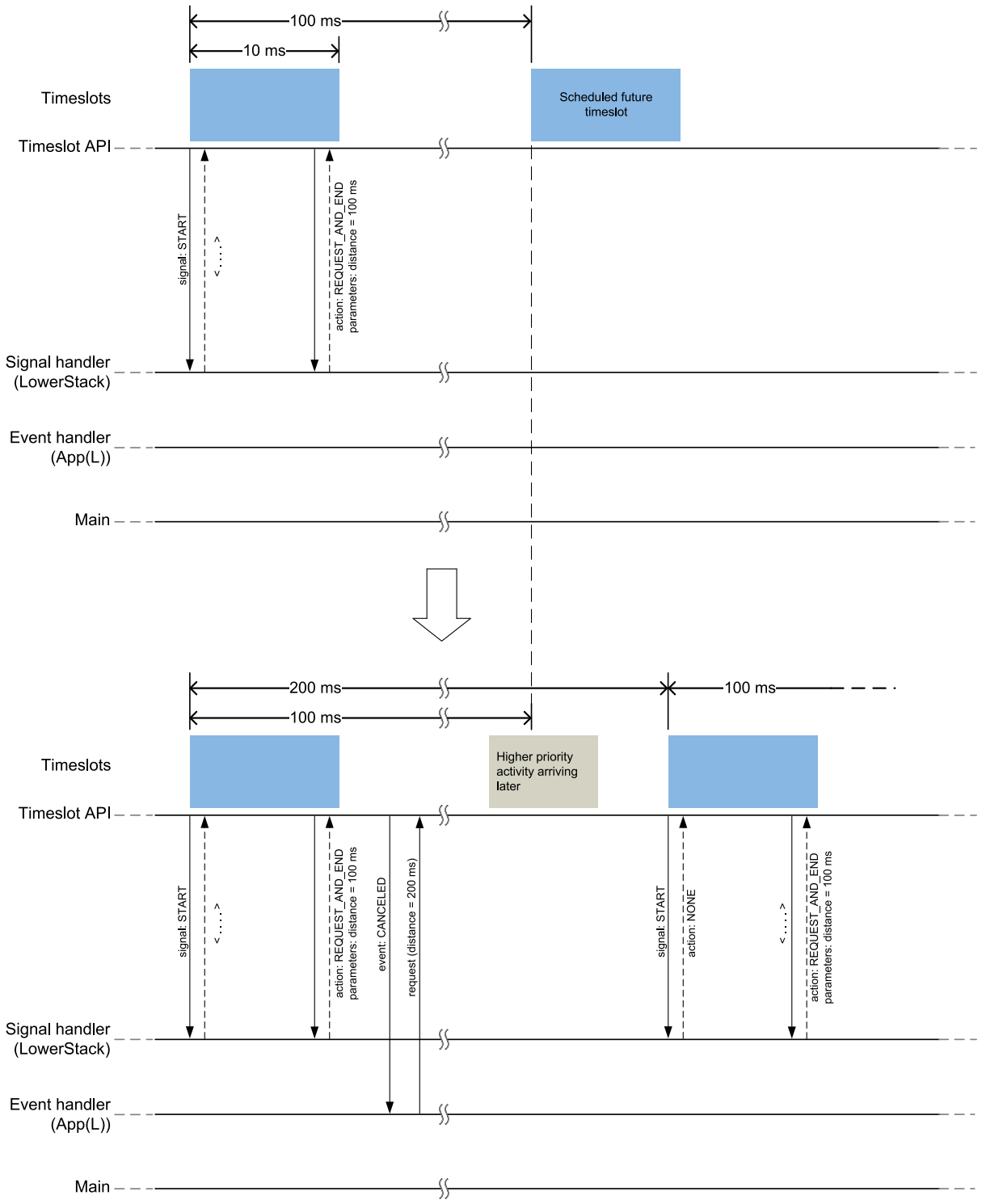

Figure 5: Canceled timeslot scenario

9.3.4 Radio Timeslot extension example

An application can use Radio 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 timeslot in Figure 6: Radio Timeslot extension example on page 33, 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 to extend the timeslot fails, as a further extension would cause a collision with a SoftDevice activity that has been scheduled. Therefore, the application

makes 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.

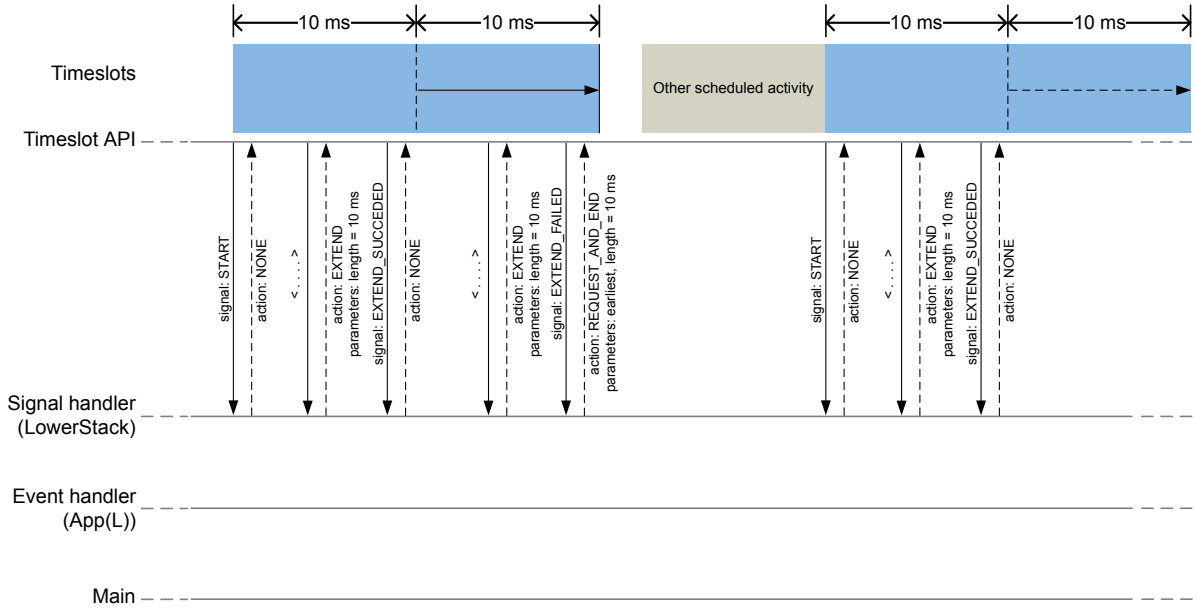

Figure 6: Radio Timeslot extension example

10 Bluetooth Low Energy protocol stack

The Bluetooth 5.1 compliant Host and Controller implemented by the SoftDevice are fully qualified with multirole support (Peripheral and Broadcaster).

The SoftDevice allows applications to implement standard Bluetooth Low Energy profiles as well as proprietary use case implementations. The *API* is defined above the *Generic Attribute Protocol (GATT)*, *Generic Access Profile (GAP)*, and *Logical Link Control and Adaptation Protocol (L2CAP)*. Other protocols, such as the *Attribute Protocol (ATT)*, *Security Manager (SM)*, and *Link Layer (LL)*, are managed by the higher layers of the SoftDevice as shown in the following figure.

The nRF5 Software Development Kit (nRF5 SDK) complements the SoftDevice with Service and Profile implementations. Single-mode *SoC* applications are enabled by the Bluetooth Low Energy protocol stack and nRF52 Series *SoC*.

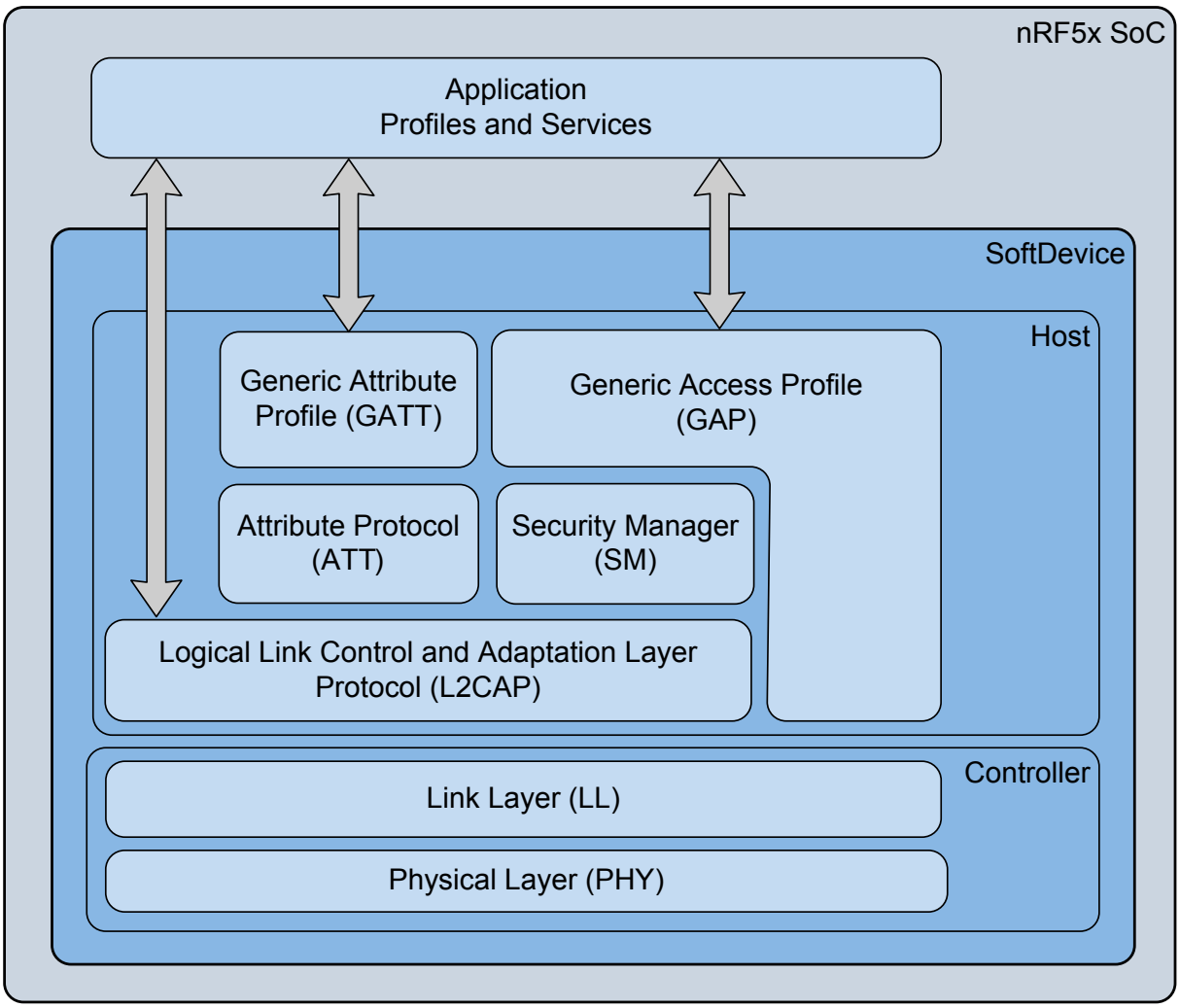

Figure 7: SoftDevice stack architecture

10.1 Profile and service support

This section lists the profiles and services adopted by the Bluetooth Special Interest Group at the time of publication of this document.

The SoftDevice supports a number of GATT based profiles which are listed in the following table. The SoftDevice also supports the use of proprietary profiles. The GATT profile specifications can be found on Bluetooth's website in GATT Specifications.

Adopted profile	Adopted services
Human Interface Device (HID) over *GATT*	*HID* Battery Device Information
Heart Rate	Heart Rate Device Information
Proximity	Link Loss Immediate Alert TX Power
Blood Pressure	Blood Pressure Device Information
Health Thermometer	Health Thermometer Device Information
Glucose	Glucose Device Information
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

Adopted profile	Adopted services
Weight Scale	Weight Scale Body Composition User Data Device Information
Continuous Glucose Monitoring	Continuous Glucose Monitoring Bond Management Device Information
Environmental Sensing	Environmental Sensing
Pulse Oximeter	Pulse Oximeter Device Information Bond Management Battery Current Time
Object Transfer	Object Transfer
Automation IO	Automation IO
	Indoor Positioning
Internet Protocol Support	
Fitness Machine Profile	Fitness Machine Device Information User Data
Reconnection Configuration Profile	Reconnection Configuration Service

Table 12: Supported profiles and services

Note: Examples for selected profiles and services are available in the nRF5 *SDK*. See the nRF5 SDK documentation for details.

10.2 Bluetooth Low Energy features

The Bluetooth Low Energy 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, data transfer, and bonding.

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

API features	Description
Interface to *GATT*/*GAP*	Consistency between *API*s including shared data formats
Attribute table sizing, population, and access	Full flexibility to size the attribute table at application compile time and to populate it at run time. 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
Application control of PHY	Full application control over the PHYs negotiated in connections
Application control of MTU size and packet length	Full application control of MTU size and packet length used in connections

Table 13: API features in the Bluetooth Low Energy stack

GAP features	Description
Multirole	Connectable advertiser or Broadcaster can run concurrently with peripheral connections.
Multiple bond support	Keys and peer information stored in application space. No restrictions in stack implementation.
Security Mode 1, Levels 1, 2, 3, and 4	Support for all levels of *SM* 1

Table 14: GAP features in the Bluetooth Low Energy stack

GATT features	Description
GATT Server	Support for one *ATT* server for all connections Includes configurable Service Changed support
Support for authorization	Enables control points Enables the application to provide fresh data Enables *GAP* authorization
GATT Client	Flexible data management options for packet transmission with either fine control or abstract management.
Implemented *GATT* Sub-procedures	Exchange MTU 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 Read Multiple Characteristic Values (Client only) 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 15: GATT features in the Bluetooth Low Energy stack

SM features	Description
Flexible key generation and storage for reduced memory requirements	Keys are stored directly in application memory to avoid unnecessary copies and memory constraints.
Authenticated *Man-in-the-Middle (MITM)* protection	Allows for per-link elevation of the encryption security level.
Pairing methods: Just works, Numeric Comparison, Passkey Entry, and Out of Band	*API* provides the application full control of the pairing sequences.

Table 16: SM features in the Bluetooth Low Energy stack

ATT features	Description
Server protocol	Fast and memory efficient implementation of the *ATT* server role
Client protocol	Fast and memory efficient implementation of the *ATT* client role
Configurable ATT_MTU size	Allows for per-link configuration of ATT_MTU size

Table 17: ATT features in the Bluetooth Low Energy stack

L2CAP features	Description
LE Credit-based Flow Control Mode	Configurable support for up to 64 channels on each link

Table 18: L2CAP features in the Bluetooth Low Energy stack

LL features	Description
Slave role Advertiser role	The SoftDevice supports multiple concurrent peripheral connections and an additional broadcaster or connectable advertiser. The connectable advertiser cannot be started when the number of available simultaneous connections has been reached, but the advertiser can still be started as a broadcaster.
Channel map configuration	Accepting update for the channel map for a peripheral connection.
LE Data Packet Length Extension (DLE)	Up to 251 bytes of *LL* data channel packet payload. Both central and peripheral roles are able to initiate a Data Length Update procedure and respond to a peer-initiated Data Length Update procedure.
LE 1M PHY LE 2M PHY	LE connections transmitting and receiving packets on all PHYs. Both symmetric connections (where the TX and RX PHYs are the same) and asymmetric connections (where the TX and RX PHYs are different) are supported. Peripheral role is able to initiate a PHY update procedure and respond to a peer-initiated PHY update procedure.
Encryption	
RSSI	Channel-specific signal strength measurements during advertising and peripheral connections.
LE Ping	
Privacy	The *LL* can generate and resolve resolvable private addresses in the advertiser.

Table 19: LL features in the Bluetooth Low Energy stack

Proprietary features	Description
TX Power control	Access for the application to change transmit power settings for the advertiser or a specific connection handle.
MBR for *Device Firmware Update (DFU)*	Enables over-the-air firmware replacement, including full SoftDevice update capability.

Table 20: Proprietary features in the Bluetooth Low Energy stack

10.3 Limitations on procedure concurrency

There are no limitations on the procedure concurrency for this SoftDevice.

10.4 Bluetooth Low Energy role configuration

The S113 SoftDevice stack supports concurrent operation in multiple Bluetooth Low Energy roles. The roles available can be configured when the S113 SoftDevice stack is enabled at runtime.

The SoftDevice provides a mechanism for enabling the number of peripheral roles the application can run concurrently. The SoftDevice can be configured with multiple connections as a Peripheral. The SoftDevice supports running one connectable Advertiser or Broadcaster concurrently with the Bluetooth Low Energy connections.

A connectable Advertiser can only be started if the number of connections is less than the maximum supported.

When the SoftDevice is enabled, it will allocate memory for the connections the application has requested. The SoftDevice will make sure that it has enough buffers to avoid buffer starvation within a connection event if the application processes the SoftDevice events immediately when they are raised.

The SoftDevice supports per connection bandwidth configuration by enabling the application to request an update of the connection interval and the length of the connection event. By default, connections are set to have an event length of 3.75 ms. This is sufficient for three packet pairs in a connection event with the default 27 octet-long *LL* payload for Data Channel PDUs.

The connection bandwidth can be increased by enabling Connection Event Length Extension. See Connection timing with Connection Event Length Extension on page 60 for more information. Enabling Connection Event Length Extension does not increase the size of the SoftDevice memory pools.

Bandwidth and multilink scheduling can affect each other. See Scheduling on page 57 for details. Knowledge about multilink scheduling can be used to get improved performance on all links. Refer to Suggested intervals and windows on page 61 for details about recommended configurations.

11 Radio Notification

The Radio Notification is a configurable feature that enables ACTIVE and INACTIVE (nACTIVE) signals from the SoftDevice to the application notifying it when the radio is in use.

11.1 Radio Notification signals

Radio notification signals are used to inform the application about radio activity.

The Radio Notification signals are sent right before or at the end of defined time intervals of radio operation, namely the SoftDevice or application Radio Events[8].

Radio notifications behave differently when Connection Event Length Extension is enabled. Radio Notification with Connection Event Length Extension on page 47 explains the behavior when this feature is enabled. Otherwise, this chapter assumes that the feature is disabled.

To ensure that the Radio Notification signals behave in a consistent way, the Radio Notification shall always be configured when the SoftDevice is in an idle state with no protocol stack or other SoftDevice activity in progress. Therefore, it is recommended to configure the Radio Notification signals directly after the SoftDevice has been enabled.

If it is enabled, the ACTIVE signal is sent before the Radio Event starts. Similarly, if the nACTIVE signal is enabled, it is sent at the end of the Radio Event. These signals can be used by the application developer to synchronize the application logic with the radio activity. For example, the ACTIVE signal can be used to switch off external devices to manage peak current drawn during periods when the radio is ON, or to trigger sensor data collection for transmission during the upcoming Radio Event.

The notification signals are sent using software interrupt as specified in Table 5: Allocation of software interrupt vectors to SoftDevice signals on page 20.

As 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.

Refer to Table 21: Radio Notification notation and terminology on page 43 for the notation that is used in this section.

When there is sufficient time between Radio Events ($t_{gap} > t_{ndist}$), both the ACTIVE and nACTIVE notification signals will be present at each Radio Event. Figure 8: Two radio events with ACTIVE and nACTIVE signals on page 42 illustrates an example of this scenario with two Radio Events. The figure also illustrates the ACTIVE and nACTIVE signals with respect to the Radio Events.

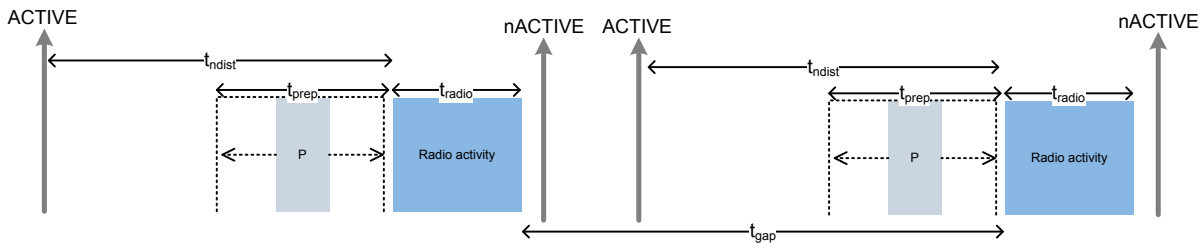

Figure 8: Two radio events with ACTIVE and nACTIVE signals

[8] Application Radio Events are defined as Radio Timeslots, see Multiprotocol support on page 24.

When there is not sufficient time between the Radio Events ($t_{gap} < t_{ndist}$), the ACTIVE and nACTIVE notification signals will be skipped. There will still be an ACTIVE signal before the first event and an nACTIVE signal after the last event. This is shown in Figure 9: Two radio events without ACTIVE and nACTIVE signals between the events on page 43 that illustrates two radio events where t_{gap} is too small and the notification signals will not be available between the events.

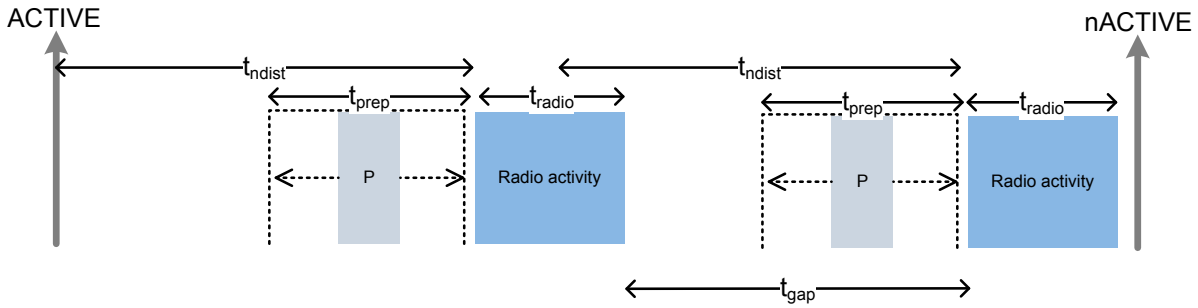

Figure 9: Two radio events without ACTIVE and nACTIVE signals between the events

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	SoftDevice CPU processing in interrupt priority level 0 between the ACTIVE signal and the start of the Radio Event	The CPU processing may occur anytime, up to t_{prep} before the start of the Radio Event.
RX	Reception of packet	
TX	Transmission of packet	
t_{radio}	The total time of a Radio Activity in a connection event	
t_{gap}	The time between the end of one Radio Event and the start of the following one	
t_{ndist}	The notification distance - the time between the ACTIVE signal and the first RX/TX in a Radio Event	This time is configurable by the application developer.
t_{prep}	The time before first RX/TX available to the protocol stack to prepare and configure the radio	The application will be interrupted by a SoftDevice interrupt handler at priority level 0 t_{prep} time units before the start of the Radio Event. **Note:** All packet data to send in an event should be sent to the stack t_{prep} before the Radio Event starts.
t_P	Time used for preprocessing before the Radio Event	

Label	Description	Notes
$t_{interval}$	Time period of periodic protocol Radio Events (e.g. Bluetooth Low Energy connection interval)	
t_{event}	Total Length of a Radio Event, including processing overhead	The length of a Radio Event for connected roles can be configured per connection by the application. This includes all the overhead associated with the Radio Event.

Table 21: Radio Notification notation and terminology

The application can configure t_{ndist} and set the following values (µs): 800, 1740, 2680, 3620, 4560, 5500.

Value	Range (µs)
t_{prep}	167 to 1542
t_P	≤ 165

Table 22: Bluetooth Low Energy Radio Notification timing ranges

The timing range for t_{radio} depends on the radio activity, as shown in Table 23: Bluetooth Low Energy Radio Activity (t_{radio}) timing ranges for advertising on LE 1M PHY on page 44 and Table 24: Bluetooth Low Energy Radio Activity (t_{radio}) timing ranges for connected roles on page 44.

Radio activity	Range
Undirected and scannable advertising - 0 to 31-byte payload, 3 channels	2750 to 5500 µs
Non-connectable advertising - 0 to 31-byte payload, 3 channels	2150 to 2950 µs
High-duty cycle directed advertising - 3 channels	1.28 s

Table 23: Bluetooth Low Energy Radio Activity (t_{radio}) timing ranges for advertising on LE 1M PHY

For connected roles, the time when the radio is active depends on the PHY. A higher bitrate reduces the radio activity time, while a lower bitrate increases the radio activity time.

PHY	Range (µs)
LE 1M PHY	310 to t_{event} - 900
LE 2M PHY	230 to t_{event} - 900

Table 24: Bluetooth Low Energy Radio Activity (t_{radio}) timing ranges for connected roles

Based on Table 21: Radio Notification notation and terminology on page 43, the amount of CPU time available to the application between the ACTIVE signal and the start of the Radio Event is:

$t_{ndist} - t_P$

The following expression shows the length of the time interval between the ACTIVE signal and the stack prepare interrupt:

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

If the data packets are to be sent in the following Radio Event, they must be transferred to the stack using the protocol *API* within this time interval.

> **Note:** t_{prep} may be greater than t_{ndist}. If time is required to handle packets or manage peripherals before interrupts are generated by the stack, t_{ndist} must be set larger than the maximum value of t_{prep}.

11.2 Radio Notification on connection events as a Peripheral

This section clarifies the functionality of the Radio Notification feature when the SoftDevice operates as a Bluetooth Low Energy Peripheral.

Radio Notification events are as shown in the following figure.

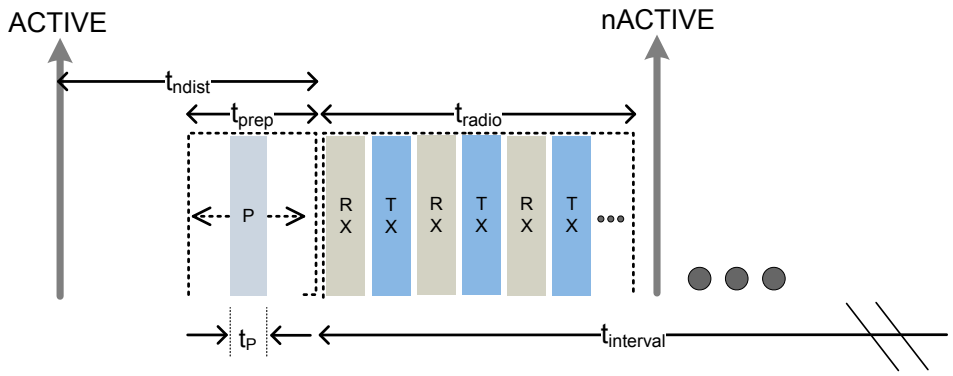

Figure 10: Peripheral link with multiple packet exchange per connection event

To guarantee that the ACTIVE notification signal is available to the application at the configured time when a single peripheral link is established, the following condition must hold:

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

For exceptions, see Table 25: Maximum peripheral packet transfer per Bluetooth Low Energy Radio Event on page 46.

The SoftDevice will limit the length of a Radio Event (t_{radio}), thereby reducing the maximum number of packets exchanged, to accommodate the selected t_{ndist}. Figure 11: Consecutive peripheral Radio Events with Radio Notification signals on page 45 shows consecutive Radio Events with Radio Notification signal and illustrates the limitation in t_{radio} which may be required to guarantee t_{ndist} is preserved.

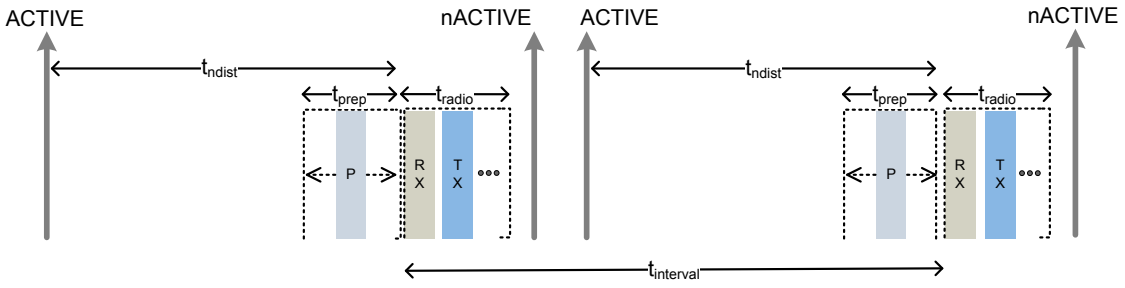

Figure 11: Consecutive peripheral Radio Events with Radio Notification signals

Table 25: Maximum peripheral packet transfer per Bluetooth Low Energy Radio Event on page 46 shows the limitation on the maximum number of 27-byte packets which can be transferred per Radio Event for given combinations of t_{ndist} and $t_{interval}$.

The data in this table assumes symmetric connections using LE 1M PHY, 27-byte packets, and full-duplex with Bluetooth Low Energy connection event length configured to be 7.5 ms and Connection Event Length Extension disabled.

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	4	6

Table 25: Maximum peripheral packet transfer per Bluetooth Low Energy Radio Event

11.3 Radio Notification with concurrent peripheral events

The Peripheral link events are arbitrarily scheduled with respect to each other. Therefore, if one link event ends too close to the start of a peripheral event, the notification signal before the peripheral connection event might not be available to the application.

Figure 12: Radio Event distance too short to trigger the notification signal on page 46 shows an example where the gap before Link-3 is too short to trigger the nACTIVE and ACTIVE notification signals.

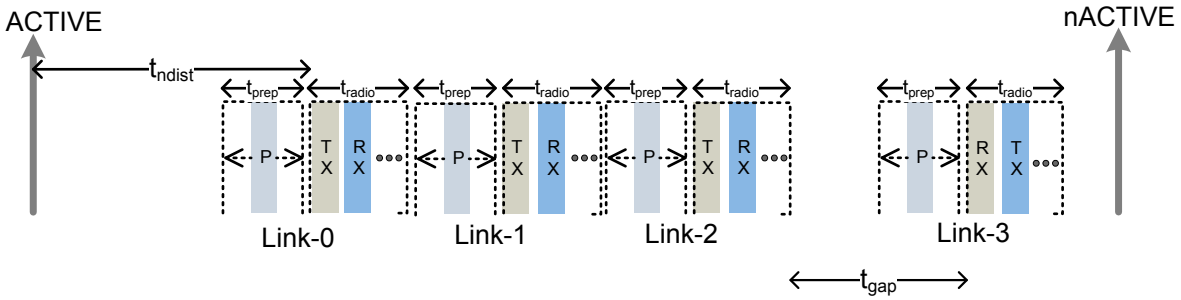

Figure 12: Radio Event distance too short to trigger the notification signal

As shown in Figure 13: Radio Event distance is long enough to trigger notification signal on page 47, the notification signal will arrive if the following condition is met:

$t_{gap} > t_{ndist}$

In the figure, the gap before Link-3 is sufficient to trigger the nACTIVE and ACTIVE notification signals.

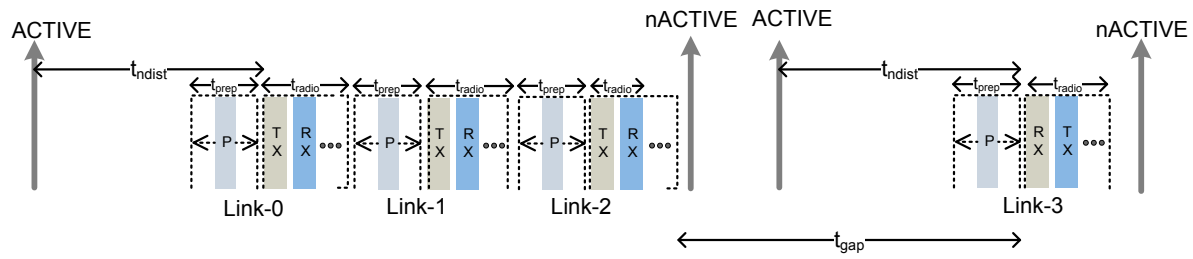

Figure 13: Radio Event distance is long enough to trigger notification signal

11.4 Radio Notification with Connection Event Length Extension

This section clarifies the functionality of the Radio Notification signal when Connection Event Length Extension is enabled in the SoftDevice.

When Connection Event Length Extension is enabled, connection events may be extended beyond their initial t_{radio} to accommodate the exchange of a higher number of packet pairs. This allows more idle time to be used by the radio and will consequently affect the radio notifications.

In peripheral links, the SoftDevice will impose a limit on how long the Radio Event (t_{radio}) may be extended, thereby restricting the maximum number of packets exchanged to accommodate the selected t_{ndist}. The following figure shows an example where the Radio Notification t_{ndist} is limiting the extension of the first Radio Event.

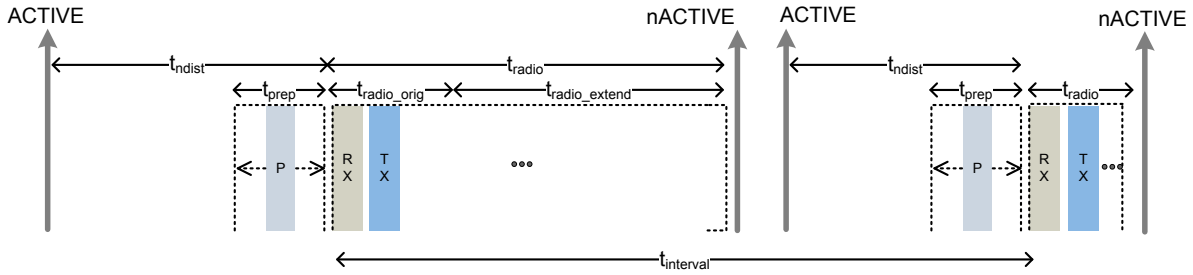

Figure 14: Peripheral connection event length extension limited by Radio Notification

11.5 Power amplifier and low noise amplifier control configuration

The SoftDevice can be configured by the application to toggle GPIO pins before and after radio transmission and before and after radio reception to control a *Power Amplifier (PA)* and/or *Low-Noise Amplifier (LNA)*.

The *PA*/*LNA* control functionality is provided by the SoftDevice protocol stack implementation and must be enabled by the application before it can be used.

> **Note:** In order to be used along with proprietary radio protocols that make use of the Timeslot API, the *PA*/*LNA* control functionality needs to be implemented as part of the proprietary radio protocol stack.

The *PA* and the *LNA* are controlled by one GPIO pin each. The *PA* pin is activated during radio transmission, and the *LNA* pin is activated during radio reception. The pins can be configured to be active low or active

high. The following figure shows an example of *PA*/*LNA* timings where the *PA* pin is configured active high and the *LNA* pin is configured active low.

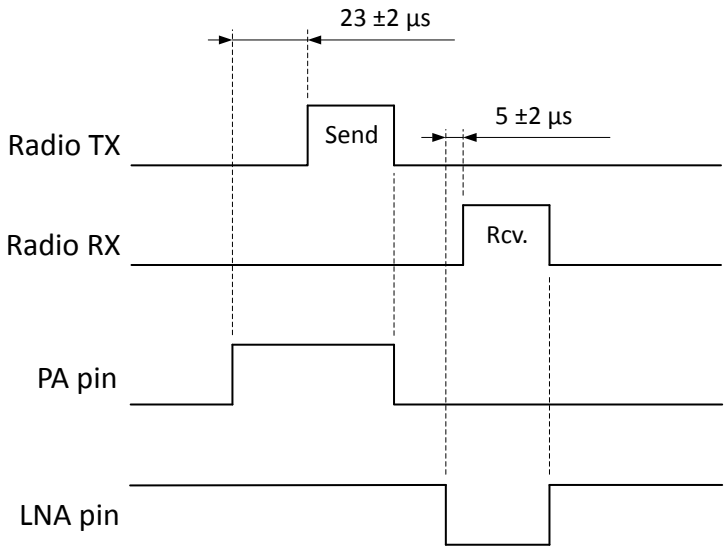

Figure 15: PA/LNA and radio activity timing

The SoftDevice uses a GPIOTE connected to a timer through a *PPI* channel to set the PA and LNA pins to active before the EVENTS_READY signal of the RADIO. The PA pin is set active 23±2 µs before EVENTS_READY for TX, and the LNA pin is set active 5±2 µs before EVENTS_READY for RX. The pins are restored to inactive state using a *PPI* connected to the EVENTS_DISABLED event on the RADIO. See the relevant product specification (Table 1: Additional documentation on page 7) for more details on the nRF52 RADIO notification signals.

12 Master boot record and bootloader

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

The nRF52 software architecture includes an MBR (see Figure 1: SoC application with the SoftDevice on page 8). The MBR is necessary 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.

Note: The SoftDevice is built to run on different versions of the nRF52 *SoC*. Depending on the available flash memory on the *SoC* and the memory requirements of the SoftDevice and application, the SoftDevice, application or bootloader may be updated. For *SoC* flash memory size, see the relevant product specification (Table 1: Additional documentation on page 7) .

12.1 Master boot record

The main functionality of the MBR is to provide an interface to allow in-system updates of the application, the SoftDevice, and bootloader firmware.

The MBR module occupies a defined region in the *SoC* program memory where the System Vector table resides.

All exceptions (reset, hard fault, interrupts, *SVC*s) are first processed by the MBR and then forwarded to the appropriate handlers (for example the bootloader or the SoftDevice exception handlers). For more information on the interrupt forwarding scheme, see Interrupt model and processor availability on page 63.

During a firmware update process, the MBR is never erased. The MBR ensures that the bootloader can recover from any unexpected resets during an ongoing update process.

When issuing the SD_MBR_COMMAND_COPY_BL or SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET commands, the MBR requires a page in the application flash region (see Memory isolation and runtime protection on page 13) for storing the MBR parameters. The address of this flash page is referred to as MBRPARAMADDR (see Figure 16: MBR, SoftDevice, and bootloader architecture on page 50). The MBRPARAMADDR address can be provided either at the `MBR_PARAM_ADDR` flash memory location, which is defined in `nrf_mbr.h`, or in the UICR.NRFFW[1] register. Using the flash memory location is the safest because it can be write protected. This is also the location that will be checked first by the MBR. UICR.NRFFW[1] is checked only if `MBR_PARAM_ADDR` has the default value, which is 0xFFFFFFFF.

When an MBRPARAMADDR address is provided, the page it refers to must not be used by the application. The page will be cleared by the MBR and used to store parameters before chip reset.

The MBR commands that require flash access will return `NRF_ERROR_NO_MEM` if the MBRPARAMADDR address is not provided. If the MBR commands that require flash access are not used, the application does not need to reserve the flash page, and it can leave the `MBR_PARAM_ADDR` flash memory location and the UICR.NRFFW[1] register as 0xFFFFFFFF, which is the default value.

12.2 Bootloader

A bootloader may be used to handle in-system update procedures.

The bootloader has full access to the SoftDevice *API* and can be implemented like any application that uses the SoftDevice. In particular, the bootloader can make use of the SoftDevice *API* for Bluetooth Low Energy communication.

The bootloader is supported in the SoftDevice architecture by using a configurable base address for the bootloader in the application flash region. This address is referred to as BOOTLOADERADDR (see Figure 16: MBR, SoftDevice, and bootloader architecture on page 50). The BOOTLOADERADDR address can be provided either at the `MBR_BOOTLOADER_ADDR` flash memory location, which is defined in `nrf_mbr.h`, or in the UICR.NRFFW[0] register. Using the flash memory location is the safest because it can be write protected protected. This is also the location that will be checked first by the MBR. UICR.NRFFW[0] is checked only if `MBR_BOOTLOADER_ADDR` has the default value, which is 0xFFFFFFFF.

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 and verifying the integrity of the firmware, including the application, SoftDevice, and the bootloader itself. If an unexpected reset occurs during a firmware update, the bootloader is responsible for detecting it and resuming the update procedure.

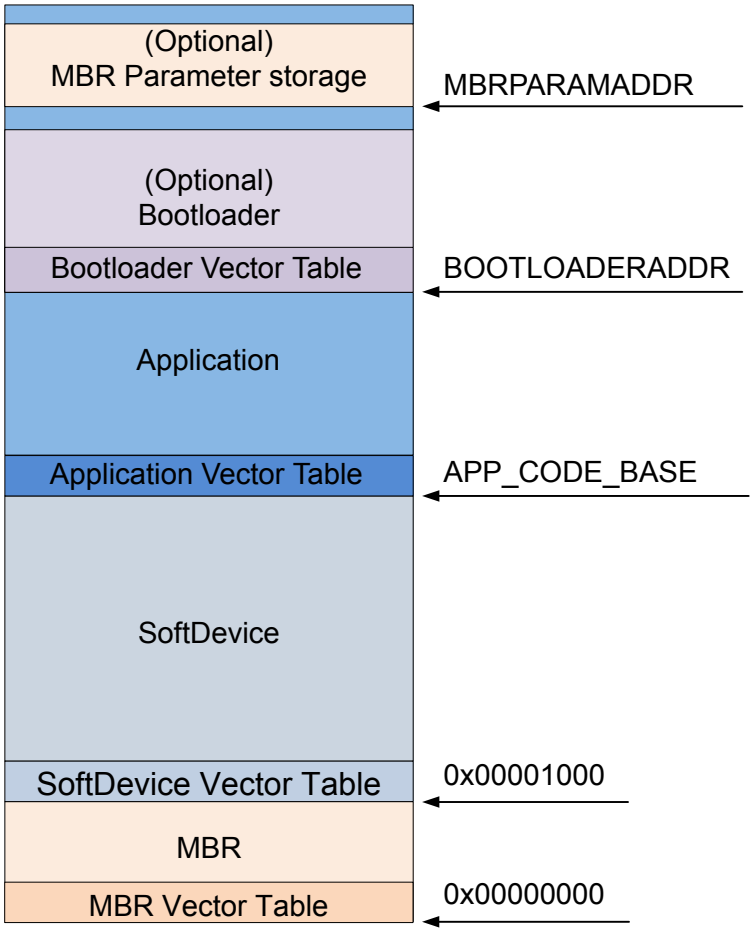

Figure 16: MBR, SoftDevice, and bootloader architecture

12.3 Master boot record and SoftDevice reset procedure

Upon system reset, the execution branches to the MBR Reset Handler as specified in the System Vector Table.

This section describes the MBR and SoftDevice reset behavior.

- If an in-system bootloader update procedure is in progress:
  - The in-system update procedure continues its execution.
  - System resets.
- Else if SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET has been called previously:
  - Forward interrupts to the address specified in the `sd_mbr_command_vector_table_base_set_t` parameter of the SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET command.
  - Run from Reset Handler (defined in the vector table which is passed as command parameter).
- Else if a bootloader is present:
  - Forward interrupts to the bootloader.
  - Run Bootloader Reset Handler (defined in bootloader Vector Table at BOOTLOADERADDR).
- Else if a SoftDevice is present:
  - Forward interrupts to the SoftDevice.
  - Execute the SoftDevice Reset Handler (defined in SoftDevice Vector Table at 0x00001000).
  - In this case, `APP_CODE_BASE` is hardcoded inside the SoftDevice.
  - The SoftDevice invokes the Application Reset Handler (as specified in the Application Vector Table at `APP_CODE_BASE`).
- Else system startup error:
  - Sleep forever.

12.4 Master boot record and SoftDevice initialization procedure

The SoftDevice can be enabled by the bootloader.

The bootloader can enable the SoftDevice by using the following procedure:

1. Issuing a command for MBR to forward interrupts to the SoftDevice using `sd_mbr_command()` with SD_MBR_COMMAND_INIT_SD.
2. Issuing a command for the SoftDevice to forward interrupts to the bootloader using `sd_softdevice_vector_table_base_set(uint32_t address)` with BOOTLOADERADDR as parameter.
3. Enabling the SoftDevice using `sd_softdevice_enable()`.

The bootloader can transfer the execution from itself to the application by using the following procedure:

1. Issuing a command for MBR to forward interrupts to the SoftDevice using `sd_mbr_command()` with SD_MBR_COMMAND_INIT_SD, if interrupts are not forwarded to the SoftDevice.
2. Issuing `sd_softdevice_disable()`, to ensure that the SoftDevice is disabled.
3. Issuing a command for the SoftDevice to forward interrupts to the application using `sd_softdevice_vector_table_base_set(uint32_t address)` with `APP_CODE_BASE` as a parameter.
4. Branching to the application Reset Handler as specified in the Application Vector Table.

13 SoftDevice information structure

The SoftDevice binary file contains an information structure.

The structure is illustrated in Figure 17: SoftDevice information structure on page 52. The location of the structure and the contents of various structure fields can be obtained at run time by the application using macros defined in the `nrf_sdm.h` header file. The information structure can also be accessed by parsing the binary SoftDevice file.

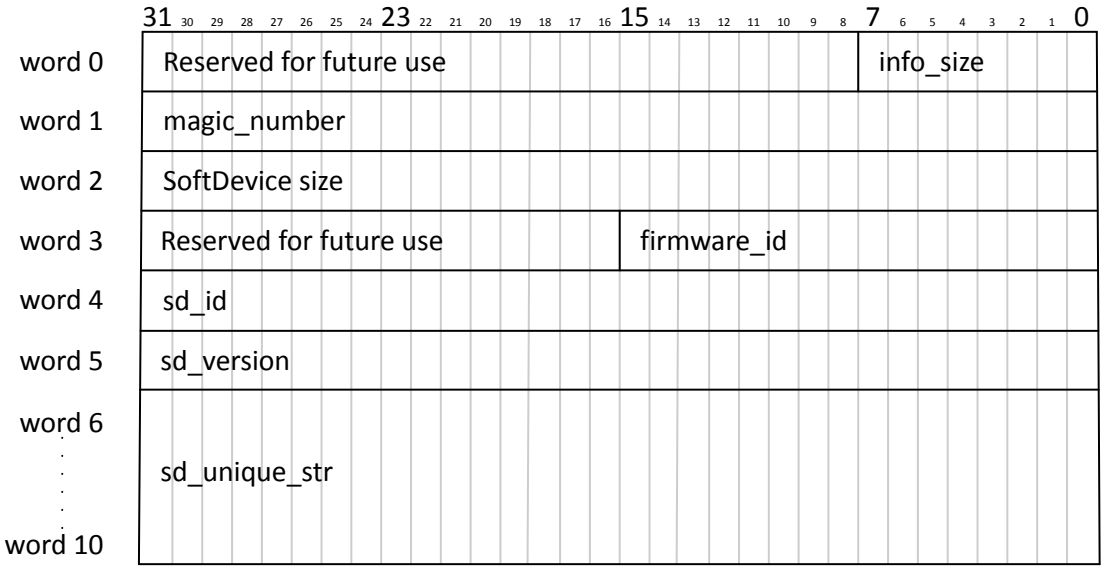

Figure 17: SoftDevice information structure

The SoftDevice release is identified by the Firmware ID, located in `firmware_id`, and the code revision, located in `sd_unique_str`. A unique Firmware ID is assigned to each production and beta release. Alpha and prealpha releases usually have a firmware ID set to 0xFFFE. The code revision in `sd_unique_str` is the git hash from which the SoftDevice is built.

14 SoftDevice memory usage

The SoftDevice shares the available flash memory and RAM on the nRF52 *SoC* with the application. The application must therefore be aware of the memory resources needed by the SoftDevice and leave the parts of the memory used by the SoftDevice undisturbed for correct SoftDevice operation.

The SoftDevice requires a fixed amount of flash memory and RAM, which are detailed in Memory resource requirements on page 54. In addition, depending on the runtime configuration, the SoftDevice will require:

- Additional RAM for Bluetooth Low Energy roles and bandwidth (see Role configuration on page 56)
- Attributes (see Attribute table size on page 55)
- UUID storage (see Vendor specific UUID counts on page 56)

14.1 Memory resource map and usage

The memory map for program memory and RAM when the SoftDevice is enabled is described in this section.

Figure 18: Memory resource map on page 54 illustrates the memory usage of the SoftDevice alongside a user application. The flash memory for the SoftDevice is always reserved, and the application program code should be placed above the SoftDevice at `APP_CODE_BASE`. The SoftDevice uses the first eight bytes of RAM when not enabled. Once enabled, the RAM usage of the SoftDevice increases. With the exception of the call stack, the RAM usage for the SoftDevice is always isolated from the application usage. Therefore, the application is required to not access the RAM region below `APP_RAM_BASE`. The value of `APP_RAM_BASE` is obtained by calling `sd_softdevice_enable`, which will always return the required minimum start address of the application RAM region for the given configuration. An access below the required minimum application RAM start address will result in undefined behavior. The RAM requirements of an enabled SoftDevice are detailed in Table 26: S113 Memory resource requirements for RAM on page 54.

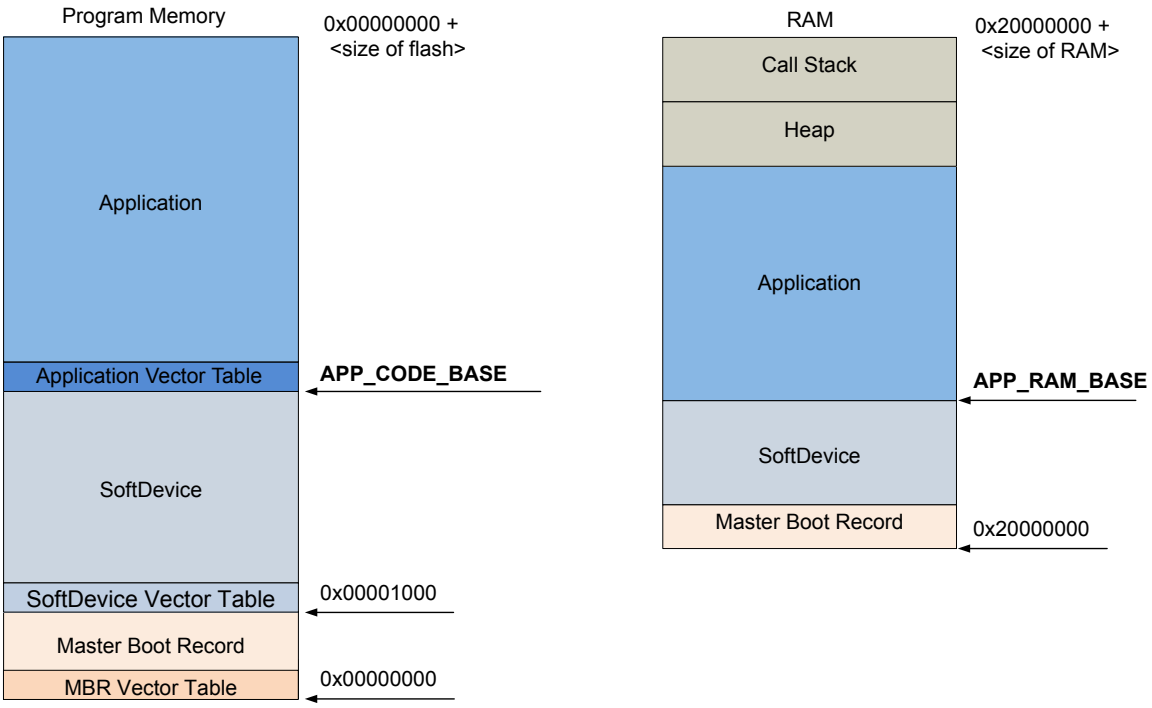

Figure 18: Memory resource map

14.1.1 Memory resource requirements

This section describes the memory resource requirements for an enabled and disabled S113 SoftDevice.

Flash

The combined flash usage of the SoftDevice and the MBR can be found in the SoftDevice properties section of the release notes. This value corresponds to `APP_CODE_BASE` in Figure 18: Memory resource map on page 54. The combined flash usage of the SoftDevice and the MBR can also be calculated by adding the MBR flash usage, which is 4 kB[9], to the `SD_FLASH_SIZE` defined in `nrf_sdm.h`.

RAM

RAM	S113 Enabled	S113 Disabled
SoftDevice RAM consumption	Minimum required RAM [10] + Configurable Resources	8 bytes
APP_RAM_BASE address (minimum required value)	0x20000000 + SoftDevice RAM consumption	0x20000008

Table 26: S113 Memory resource requirements for RAM

Call stack

By default, the nRF52 *SoC* will have a shared call stack with both application stack frames and SoftDevice stack frames, managed by the *MSP*.

[9] 1 kB = 1024 bytes

[10] For the minimum RAM required by the SoftDevice, see the SoftDevice properties section of the release notes.

The application configures the call stack, and the *MSP* gets initialized on reset to the address specified by the application vector table entry 0. In its reset vector the application may configure the CPU to use the *Process Stack Pointer (PSP)* in thread mode. This configuration is optional but may be required by an OS, for example, to isolate application threads and OS context memory. The application programmer must be aware that the SoftDevice will use the *MSP* as it is always executed in exception mode.

Note: It is customary, but not required, to let the stack run downwards from the upper limit of the RAM Region.

With each major release of an S113 SoftDevice, its maximum (worst case) call stack requirement may be updated. The SoftDevice uses the call stack when SoftDevice interrupt handlers execute. These are asynchronous to the application, so the application programmer must reserve call stack for the application in addition to the call stack requirement by the SoftDevice.

The application must reserve sufficient space to satisfy both the application and the SoftDevice stack memory requirements. The nRF52 *SoC* has no designated hardware for detecting stack overflow. The application can use the *MWU* peripheral to implement a mechanism for stack overflow detection if the *MWU* is available on the *SoC*. For available peripherals, see the relevant product specification in Table 1: Additional documentation on page 7.

The SoftDevice does not use the ARM Cortex-M4 *Floating-Point Unit (FPU)* and does not configure any floating-point registers. Table 27: S113 Memory resource requirements for call stack on page 55 depicts the maximum call stack size that may be consumed by the SoftDevice when not using the *FPU*.

Note: The *FPU* is not available on some nRF52 *SoC*s. See the relevant product specification in Table 1: Additional documentation on page 7.

The SoftDevice uses multiple interrupt levels, as described in detail in Interrupt model and processor availability on page 63. If *FPU* is used by the application, the processor will need to reserve memory in the stack frame for stacking the *FPU* registers for each interrupt level used by the SoftDevice. This must be accounted for when configuring the total call stack size. For more information on how the use of multiple interrupt levels impacts the stack size when using the *FPU*, see Application Note 298 from ARM regarding the ARM Cortex-M4 processor with *FPU*.

Call stack	S113 Enabled	S113 Disabled
Maximum usage with FPU disabled	1536 bytes (0x600)	0 bytes

Table 27: S113 Memory resource requirements for call stack

Heap

There is no heap required by nRF52 SoftDevices. The application is free to allocate and use a heap without disrupting the SoftDevice functionality.

14.2 Attribute table size

The size of the attribute table can be configured through the SoftDevice *API* when enabling the Bluetooth Low Energy stack.

The default and minimum values of the attribute table size, `ATTR_TAB_SIZE`, can be found in `ble_gatts.h`. Applications that require an attribute table smaller or bigger than the default size can choose to either reduce or increase the attribute table size. The amount of RAM reserved by the

SoftDevice and the minimum required start address for the application RAM, `APP_RAM_BASE`, will then change accordingly.

The attribute table size is set through `sd_ble_cfg_set`.

14.3 Role configuration

The SoftDevice allows the number of connections, the configuration of each connection, and its role to be specified by the application.

Role configuration, the number of connections, and connection configuration, will determine the amount of RAM resources used by the SoftDevice. The minimum required start address for the application RAM, `APP_RAM_BASE`, will change accordingly. See Bluetooth Low Energy role configuration on page 41 for more details on role configuration.

14.4 Vendor specific UUID counts

The SoftDevice allows the use of vendor specific UUIDs, which are stored by the SoftDevice in the RAM that is allocated once the SoftDevice is enabled.

The number of vendor specific UUIDs that can be stored by the SoftDevice is set through `sd_ble_cfg_set`.

15 Scheduling

The S113 stack has multiple activities, called timing-activities, which require exclusive access to certain hardware resources. These timing-activities are time-multiplexed to give them the required exclusive access for a period of time. This is called a timing-event. Such timing-activities are Bluetooth Low Energy role events like events for Peripheral roles, Flash memory API usage, and Radio Timeslot API timeslots.

If timing-events collide, their scheduling is determined by a priority system. If timing-activity A needs a timing-event at a time that overlaps with timing-activity B, and timing-activity A has higher priority, timing-activity A will get the timing-event. Activity B will be blocked and its timing-event will be rescheduled for a later time. If both timing-activity A and timing-activity B have the same priority, the timing-activity which was requested first will get the timing-event.

The timing-activities run to completion and cannot be preempted by other timing-activities, even if the timing-activity trying to preempt has a higher priority. This is the case, for example, when timing-activity A and timing-activity B request a timing-event at overlapping times with the same priority. Timing-activity A gets the timing-event because it requested it earlier than timing-activity B. If timing-activity B increased its priority and requested again, it would only get the timing-event if timing-activity A had not already started and there was enough time to change the timing-event schedule.

Note: The figures in this chapter do not illustrate all packets that are sent over the air. See *Bluetooth Core Specification* for the complete sequence of packets.

15.1 SoftDevice timing-activities and priorities

The SoftDevice supports multiple connections simultaneously in addition to an Advertiser or a Broadcaster. In addition to these Bluetooth Low Energy roles, Flash memory API and Radio Timeslot API can also run simultaneously.

Advertiser and broadcaster timing-events are scheduled as early as possible. Peripheral link timing-events follow the timings dictated by the connected peer. As peripheral and advertising events are scheduled without knowing about each other, they may occur at the same time and collide. Flash access timing-events and Radio Timeslot timing-events are also scheduled independently and so may occur at the same time and collide.

The different timing-activities have different priorities at different times, dependent upon their state. As an example, if a connection is about to reach supervision time-out, it will block all other timing-activities and get the timing-event it requests. In this case, all other timing-activities will be blocked if they overlap with the connection timing-event, and they will have to be rescheduled. The following table summarizes the priorities.

Priority (Decreasing order)	Role state
First priority	• Peripheral connection setup (waiting for ack from peer) • Peripheral connections that are about to time out
Second priority	• Connectable advertiser/Broadcaster which has been blocked consecutively for a few times
Third priority	• All Bluetooth Low Energy roles in states other than above run with this priority • Flash access after it has been blocked consecutively for a few times • Radio Timeslot with high priority
Fourth priority	• Flash access • Radio Timeslot with normal priority

Table 28: Scheduling priorities

15.2 Advertiser timing

Advertiser is started as early as possible, after a random delay in the range of 3 - 13 ms, asynchronously to any other role timing-events. If no roles are running, advertiser timing-events are able to start and run without any collision.

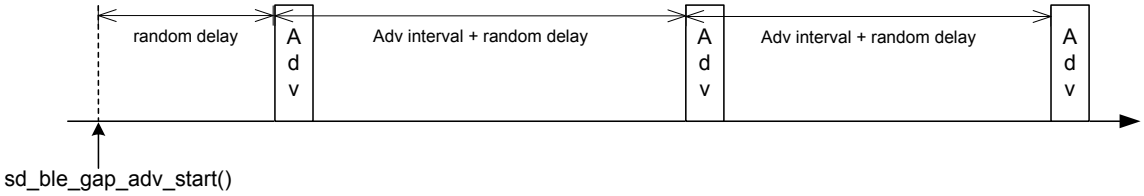

Figure 19: Advertiser

When other role timing-events are running in addition, the advertiser role timing-event may collide with those. The following figure shows a scenario of Advertiser colliding with Peripheral (P).

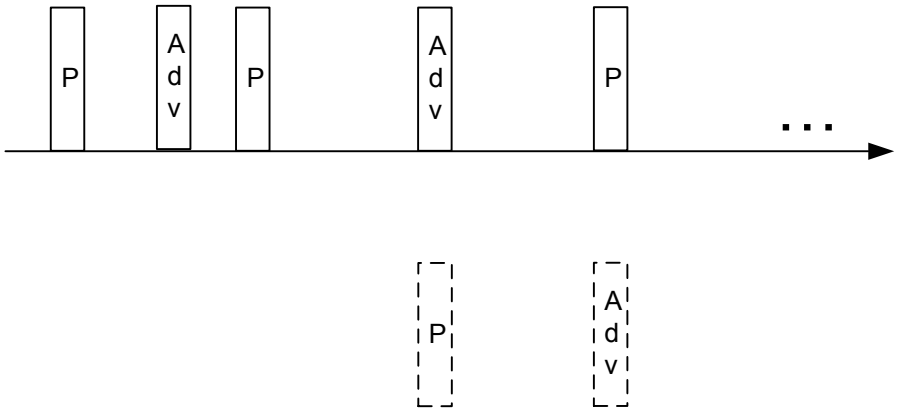

Figure 20: Advertiser collision

A directed high duty cycle advertiser is different compared to other advertiser types because it is not periodic. The scheduling of the single timing-event required by a directed advertiser is done in the same way as other advertiser type timing-events. A directed high duty cycle advertiser timing-event is also started as early as possible, and its priority (refer to Table 28: Scheduling priorities on page 58) is raised if it is blocked by other role timing-events multiple times.

15.3 Peripheral connection setup and connection timing

Peripheral link timing-events are added as per the timing dictated by peer Central.

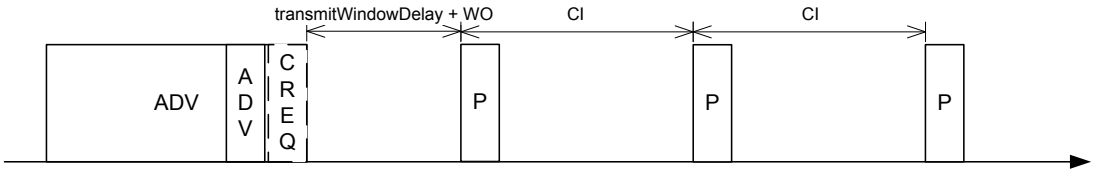

Figure 21: Peripheral connection setup and connection

Peripheral link timing-events may collide with any other running role timing-events because the timing of the connection is dictated by the peer.

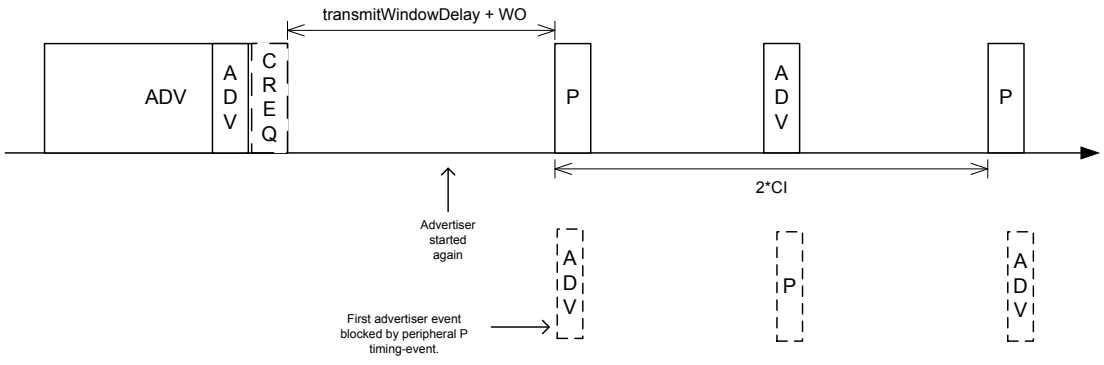

Figure 22: Peripheral connection setup and connection with collision

Value	Description	Value (µs)
$t_{SlaveNominalWindow}$	Listening window on slave to receive first packet in a connection event	`2 * (16 + 16 + 250 + 250)` Assuming 250 ppm sleep clock accuracy on both slave and master with 1-second connection interval, 16 is the sleep clock instantaneous timing on both master and slave.
$t_{SlaveEventNominal}$	Nominal event length for slave link	$t_{SlaveNominalWindow} + t_{event}$ Refer to Table 21: Radio Notification notation and terminology on page 43 and Table 22: Bluetooth Low Energy Radio Notification timing ranges on page 44.
$t_{SlaveEventMax}$	Maximum event length for slave link	$t_{SlaveEventNominal} + 7$ `ms` Where 7 ms is added for the maximum listening window for 500 ppm sleep clock accuracy on both master and slave with 4-second connection interval. The listening window is dynamic and is therefore added so that t_{radio} remains constant.
$t_{AdvEventMax}$	Maximum event length for advertiser (all types except directed high duty cycle advertiser) role	$t_{prep\ (max)} + t_{event\ (max\ for\ adv\ role\ except\ directed\ high\ duty\ cycle\ adv)}$ Refer to Table 21: Radio Notification notation and terminology on page 43 and Table 22: Bluetooth Low Energy Radio Notification timing ranges on page 44.

Table 29: Peripheral role timing ranges

15.4 Connection timing with Connection Event Length Extension

Peripheral links can extend the event if there is radio time available.

The connection event is the time within a timing-event reserved for sending or receiving packets. The SoftDevice can be enabled to dynamically extend the connection event length to fit the maximum number of packets inside the connection event before the timing-event must be ended. The time extended will be in one packet pair at a time until the maximum extend time is reached. The connection event cannot be longer than the connection interval; the connection event will then end and the next connection event will begin. A connection event cannot be extended if it will collide with another timing-event. The extend request will ignore the priorities of the timing-events.

To get the maximum bandwidth on a single link, it is recommended to enable Connection Event Length Extension and increase the connection interval. This will allow the SoftDevice to send more packets within the event and limit the overhead of processing between connection events. For more information, see Suggested intervals and windows on page 61.

Multilink scheduling with connection event length extension can increase the bandwidth for multiple links by utilizing idle time between connection events. An example of this is shown in Figure 23: Multilink scheduling and connection event length extension on page 61.

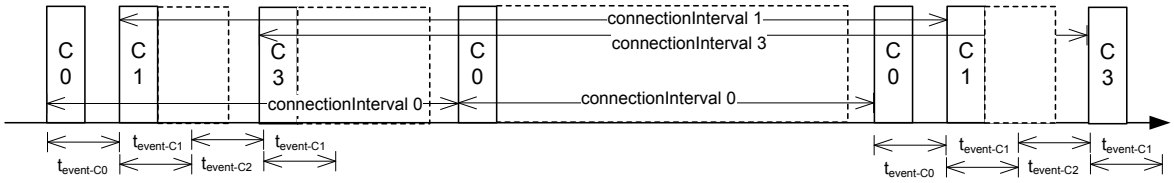

Figure 23: Multilink scheduling and connection event length extension

15.5 Flash API timing

Flash timing-activity is a one-time activity with no periodicity, as opposed to Bluetooth Low Energy role timing-activities. Hence, the flash timing-event is scheduled in any available time left between other timing-events.

To run efficiently with other timing-activities, the Flash API will run in a low priority. Other timing-activities running in higher priority can collide with flash timing-events. Refer to Table 28: Scheduling priorities on page 58 for details on priority of timing-activities, which is used in case of collision. Flash timing-activity will use higher priority if it has been blocked many times by other timing-activities. Flash timing-activity may not get a timing-event at all if other timing-events occupy most of the time and use priority higher than flash timing-activity. To avoid a long wait time while using Flash API, flash timing-activity will fail in case it cannot get a timing-event before a timeout.

15.6 Timeslot API timing

Radio Timeslot API timing-activity is scheduled independently of any other timing activity, hence it can collide with any other timing-activity in the SoftDevice.

Refer to Table 28: Scheduling priorities on page 58 for details on priority of timing-activities, which is used in case of collision. If the requested timing-event collides with already scheduled timing-events with equal or higher priority, the request will be denied (blocked). If a later arriving timing-activity of higher priority causes a collision, the request will be canceled. However, a timing-event that has already started cannot be interrupted or canceled.

If the timeslot is requested as *earliest possible*, Timeslot timing-event is scheduled in any available free time. Hence there is less probability of collision with *earliest possible* request. Timeslot API timing-activity has two configurable priorities. To run efficiently with other timing-activities, the Timeslot API should run in lowest possible priority. It can be configured to use higher priority if it has been blocked many times by other timing-activities and is in a critical state.

15.7 Suggested intervals and windows

The scheduling of Peripheral links is done by the peer devices. The Peripheral does not influence this scheduling, and the links may at some point collide with each other due to clock drifting. Therefore, when scheduling multiple peripheral links, the connection intervals and connection event lengths should be chosen in a way that leaves enough free time to handle collisions.

When collisions occur, they will be resolved using a priority mechanism. The priority mechanism will prioritize the connections in a fair manner, but still try to avoid any connections timing out.

When running multiple Peripherals, a recommended configuration for having fewer colliding Peripherals is to set a short event length and enable the Connection Event Length Extension in the SoftDevice (see Connection timing with Connection Event Length Extension on page 60).

When long *LL* Data Channel PDUs are in use, it is recommended to increase the event length of a connection. For example, *LL* Data Channel PDUs are by default 27 bytes in size. With an event length of 3.75 ms, it is possible to send three full-sized packet pairs on LE 1M PHY in one connection event. Therefore, when increasing the *LL* Data Channel PDU size to 251 bytes, the event length should be increased to 15 ms. To calculate how much time should be added (in ms), use the following formula: `((`*size*` - 27) * 8 * 2 *` *pairs*`) / 1000`.

Timing-activities other than Bluetooth Low Energy role events, such as Flash access and Radio Timeslot API, also use the same time space as all other timing-activities. Hence, they are more likely to collide.

16 Interrupt model and processor availability

This chapter documents the SoftDevice interrupt model, how interrupts are forwarded to the application, and describes how long the processor is used by the SoftDevice in different priority levels.

16.1 Exception model

As the SoftDevice, including the MBR, needs to handle some interrupts, all interrupts are routed through the MBR and SoftDevice. The ones that should be handled by the application are forwarded and the rest are handled within the SoftDevice itself. This section describes the interrupt forwarding mechanism.

For more information on the MBR, see Master boot record and bootloader on page 49.

16.1.1 Interrupt forwarding to the application

The forwarding of interrupts to the application depends on the state of the SoftDevice.

At the lowest level, the MBR receives all interrupts and forwards them to the SoftDevice regardless of whether the SoftDevice is enabled or not. The use of a bootloader introduces some exceptions to this. See Master boot record and bootloader on page 49.

Some peripherals and their respective interrupt numbers are reserved for use by the SoftDevice (see Hardware peripherals on page 18). Any interrupt handler defined by the application for these interrupts will not be called as long as the SoftDevice is enabled. When the SoftDevice is disabled, these interrupts will be forwarded to the application.

The *SVC* interrupt is always intercepted by the SoftDevice regardless of whether it is enabled or disabled. The SoftDevice inspects the *SVC* number, and if it is equal or greater than 0x10, the interrupt is processed by the SoftDevice. *SVC* numbers below 0x10 are forwarded to the application's *SVC* interrupt handler. This allows the application to make use of a range of *SVC* numbers for its own purpose, for example, for an RTOS.

Interrupts not used by the SoftDevice are always forwarded to the application.

For the SoftDevice to locate the application interrupt vectors, the application must define its interrupt vector table at the bottom of the Application Flash Region illustrated in Figure 18: Memory resource map on page 54. When the base address of the application code is directly after the top address of the SoftDevice, the code can be developed as a standard ARM Cortex -M4 application project with the compiler creating the interrupt vector table.

16.1.2 Interrupt latency due to System on Chip framework

Latency, additional to ARM Cortex -M4 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. This is the latency added by the interrupt forwarding latency alone. The maximum application interrupt latency is dependent on SoftDevice activity, as described in section Processor usage patterns and availability on page 66.

Interrupt	SoftDevice enabled	SoftDevice disabled
Open peripheral interrupt	< 4 µs	< 2 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	N/A	< 2 µs
Application SVC interrupt	< 2 µs	< 2 µs

Table 30: Additional latency due to SoftDevice and MBR forwarding interrupts

16.2 Interrupt priority levels

This section gives an overview of interrupt levels used by the SoftDevice and the interrupt levels that are available for the application.

To implement the SoftDevice *API* as *SVC*s (see Application programming interface on page 10) and ensure that embedded protocol real-time requirements are met independently of the application processing, the SoftDevice implements an interrupt model where application interrupts and SoftDevice interrupts are interwoven. This model will result in application interrupts being postponed or preempted, leading to longer perceived application interrupt latency and interrupt execution times.

The application must take care to select the correct interrupt priorities for application events according to the guidelines that follow. The NVIC *API* to the *SoC* Library supports safe configuration of interrupt priorities from the application.

The nRF52 *SoC* has eight configurable interrupt priorities ranging from 0 to 7 (with 0 being highest priority). On reset, all interrupts are configured with the highest priority (0).

The SoftDevice reserves and uses the following priority levels, which must remain unused by the application programmer:

- Level 0 is used for the SoftDevice's timing critical processing.
- Level 1 is used for handling the memory isolation and run time protection, see Memory isolation and runtime protection on page 13.
- Level 4 is used by higher-level deferrable tasks and the *API* functions executed as SVC interrupts.

The application can use the remaining interrupt priority levels, in addition to the main, or thread, context.

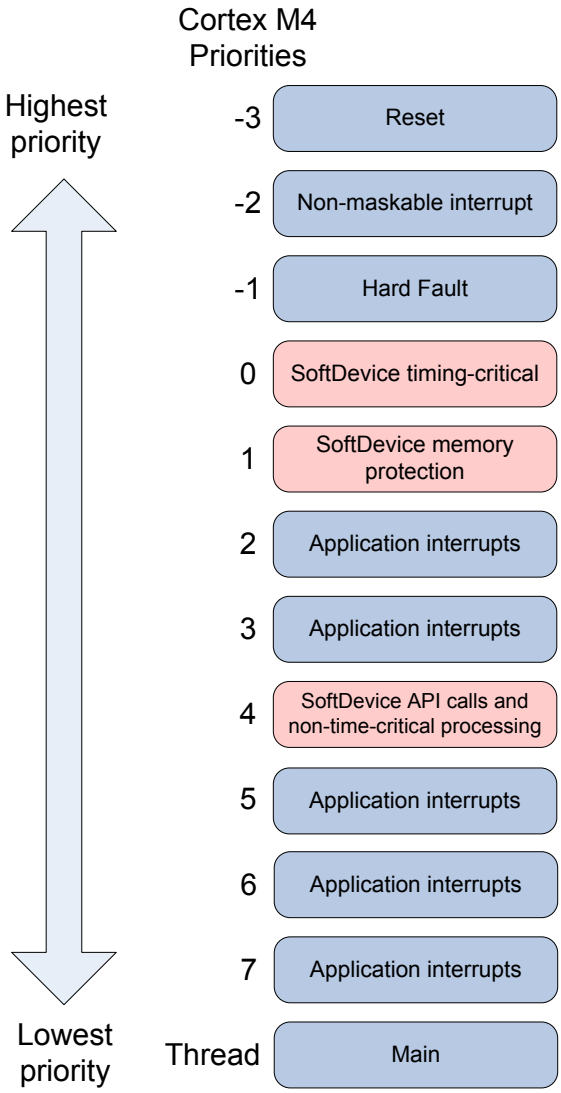

Figure 24: Exception model

As seen from Figure 24: Exception model on page 65, the application has available priority level 2 and 3, located between the higher and lower priority levels reserved by the SoftDevice. This enables a low-latency application interrupt to support fast sensor interfaces. An application interrupt at priority level 2 or 3 will only experience latency from SoftDevice interrupts at priority levels 0 and 1, while application interrupts at priority levels 5, 6, or 7 can experience latency from all SoftDevice priority levels.

Note: The priorities of the interrupts reserved by the SoftDevice cannot be changed. This includes the *SVC* interrupt. Handlers running at a priority level higher than 4 (lower numerical priority value) have neither access to SoftDevice functions nor to application specific *SVC*s or RTOS functions running at lower priority levels (higher numerical priority values).

The following figure shows an example of how interrupts with different priorities may run and preempt each other. Some priority levels are left out for clarity.

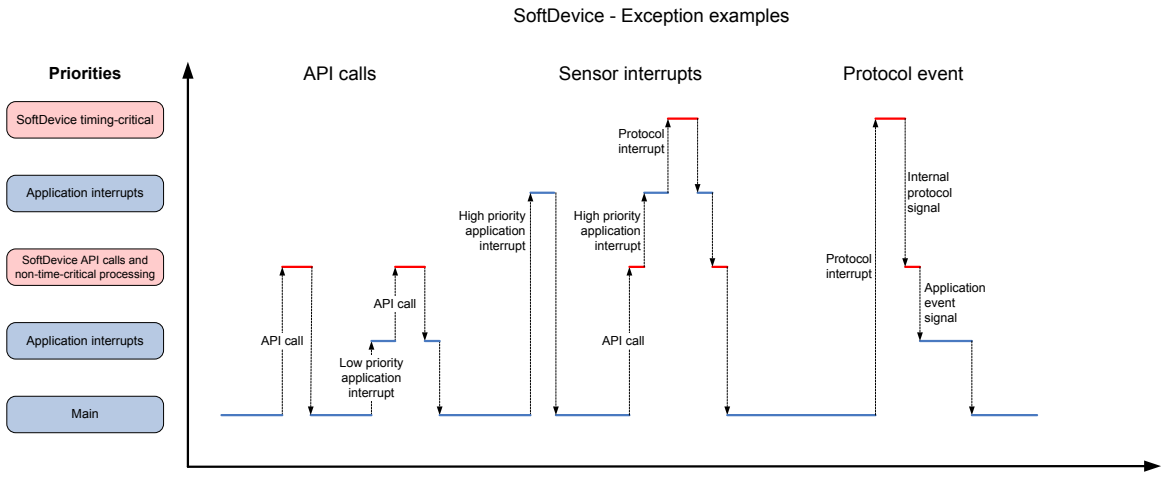

Figure 25: SoftDevice exception examples

16.3 Processor usage patterns and availability

This section gives an overview of the processor usage patterns for features of the SoftDevice and the processor availability to the application in stated scenarios.

The SoftDevice's processor use will also affect the maximum interrupt latency for application interrupts of lower priority (higher numerical value for the interrupt priority). The maximum interrupt processing time for the different priority levels in this chapter can be used to calculate the worst-case interrupt latency the application will have to handle when the SoftDevice is used in various scenarios.

In the following scenarios, $t_{ISR(x)}$ denotes interrupt processing time at priority level x, and $t_{nISR(x)}$ denotes time between interrupts at priority level x.

16.3.1 Flash API processor usage patterns

This section describes the processor availability and interrupt processing time for the SoftDevice when the Flash API is being used.

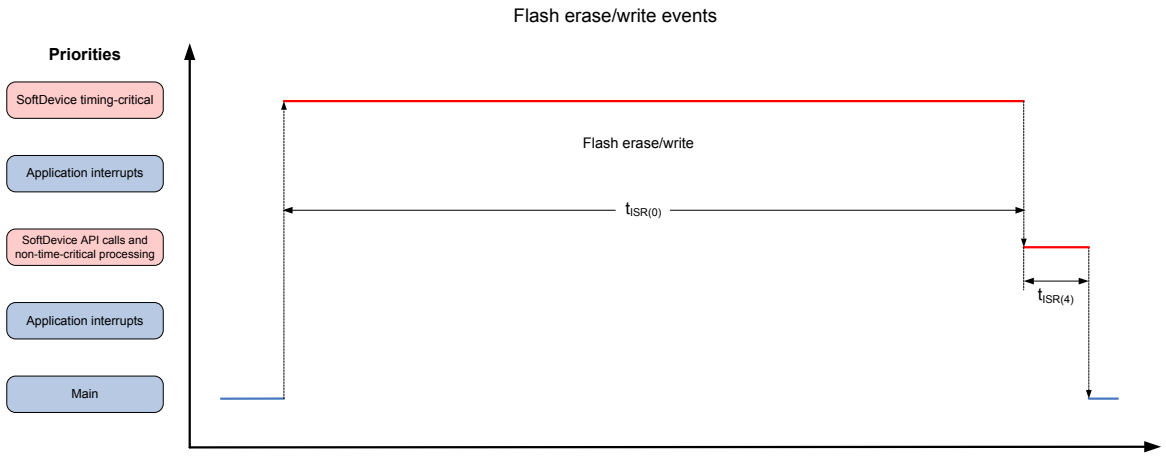

Figure 26: Flash API activity (some priority levels left out for clarity)

When using the Flash API, the pattern of SoftDevice CPU activity at interrupt priority level 0 is as follows:

1. An interrupt at priority level 0 sets up and performs the flash activity. The CPU is halted for most of the time in this interrupt.

2. After the first interrupt is complete, another interrupt at priority level 4 cleans up after the flash operation.

SoftDevice processing activity in the different priority levels during flash erase and write is outlined in the table below.

Parameter	Description	Min	Typical	Max
$t_{ISR(0),FlashErase}$	Interrupt processing when erasing a flash page. The CPU is halted most of the length of this interrupt.			90 ms
$t_{ISR(0),FlashWrite}$	Interrupt processing when writing one or more words to flash. The CPU is halted most of the length of this interrupt. The Max time provided is for writing one word. When writing more than one word, please see the Product Specification in Table 1: Additional documentation on page 7 to get the time to write one word and add it to the Max time provided in this table.			500 µs
$t_{ISR(4)}$	Priority level 4 interrupt at the end of flash write or erase.		10 µs	

Table 31: Processor usage for the Flash API

16.3.2 Radio Timeslot API processor usage patterns

This section describes the processor availability and interrupt processing time for the SoftDevice when the Radio Timeslot API is being used.

See Radio Timeslot API on page 26 for more information on the Radio Timeslot API.

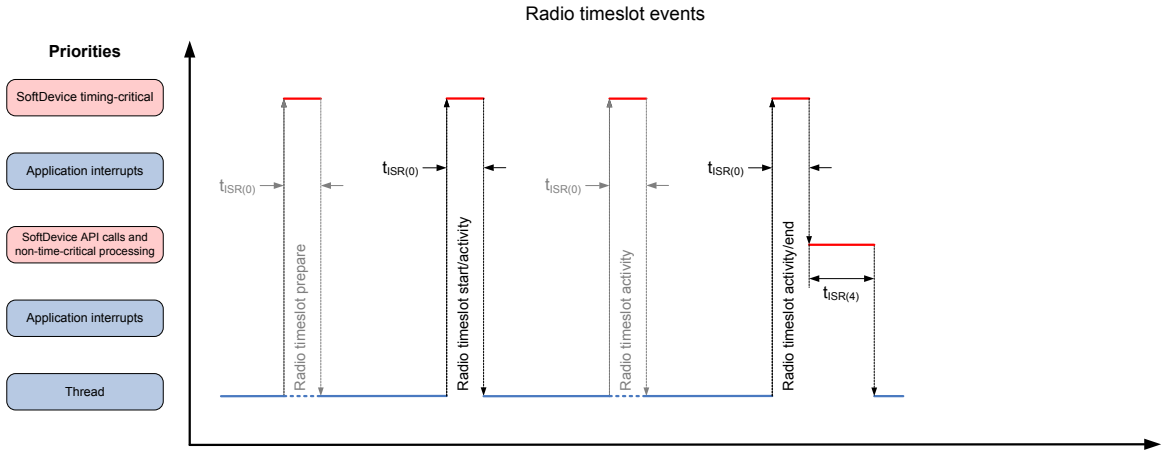

Figure 27: Radio Timeslot API activity (some priority levels left out for clarity)

When using the Radio Timeslot API, the pattern of SoftDevice CPU activity at interrupt priority level 0 is as follows:

1. If the timeslot was requested with NRF_RADIO_HFCLK_CFG_XTAL_GUARANTEED, there is first an interrupt that handles the startup of the high-frequency crystal.
2. The interrupt is followed by one or more Radio Timeslot activities. How many and how long these are is application dependent.
3. When the last of the Radio Timeslot activities is complete, another interrupt at priority level 4 cleans up after the Radio Timeslot operation.

SoftDevice processing activity at different priority levels during use of Radio Timeslot API is outlined in the table below.

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioTimeslotPrepare}$	Interrupt processing when starting up the high-frequency crystal			9 µs
$t_{ISR(0),RadioTimeslotActivity}$	The application's processing in the timeslot. The length of this is application dependent.			
$t_{ISR(4)}$	Priority level 4 interrupt at the end of the timeslot		7 µs	

Table 32: Processor usage for the Radio Timeslot API

16.3.3 Bluetooth Low Energy processor usage patterns

This section describes the processor availability and interrupt processing time for the SoftDevice when roles of the Bluetooth Low Energy protocol are running.

16.3.3.1 Bluetooth Low Energy Advertiser (Broadcaster) processor usage

This section describes the processor availability and interrupt processing time for the SoftDevice when the advertiser (broadcaster) role is running.

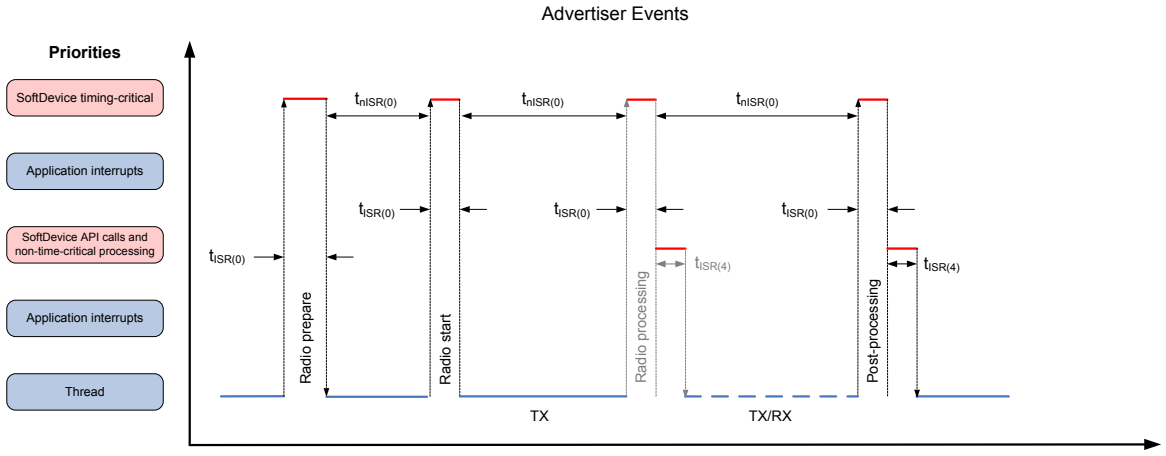

Figure 28: Advertising events (some priority levels left out for clarity)

When advertising, the pattern of SoftDevice processing activity for each advertising interval at interrupt priority level 0 is as follows:

1. An interrupt (Radio prepare) sets up and prepares the software and hardware for this advertising event.
2. A short interrupt occurs when the Radio starts sending the first advertising packet.

3. Depending on the type of advertising, there may be one or more instances of Radio processing (including processing in priority level 4) and further receptions/transmissions.
4. Advertising ends with post processing at interrupt priority level 0 and some interrupt priority level 4 activity.

SoftDevice processing activity in the different priority levels when advertising is outlined in Table 33: Processor usage when advertising on page 69. The typical case is seen when advertising without using a whitelist and without receiving scan or connect requests. The max case can be seen when advertising with a full whitelist, using private addresses, receiving scan and connect requests while having a maximum number of connections, and utilizing the Radio Timeslot API and Flash memory API at the same time.

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing when preparing the radio for advertising		32 µs	50 µs
$t_{ISR(0),RadioStart}$	Processing when starting the advertising		12 µs	20 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending/receiving a packet		54 µs	75 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of an advertising event		77 µs	128 µs
$t_{nISR(0)}$	Distance between interrupts during advertising	40 µs	>170 µs	
$t_{ISR(4)}$	Priority level 4 interrupt at the end of an advertising event		28 µs	

Table 33: Processor usage when advertising

From the table we can calculate a typical processing time for one advertisement event sending three advertisement packets to be:

$$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + 2 * t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + t_{ISR(4)} = 257 \ \mu s$$

That means typically more than 99% of the processor time is available to the application when advertising with a 100 ms interval.

16.3.3.2 Bluetooth Low Energy peripheral connection processor usage

This section describes the processor availability and interrupt processing time for the SoftDevice in a peripheral connection event.

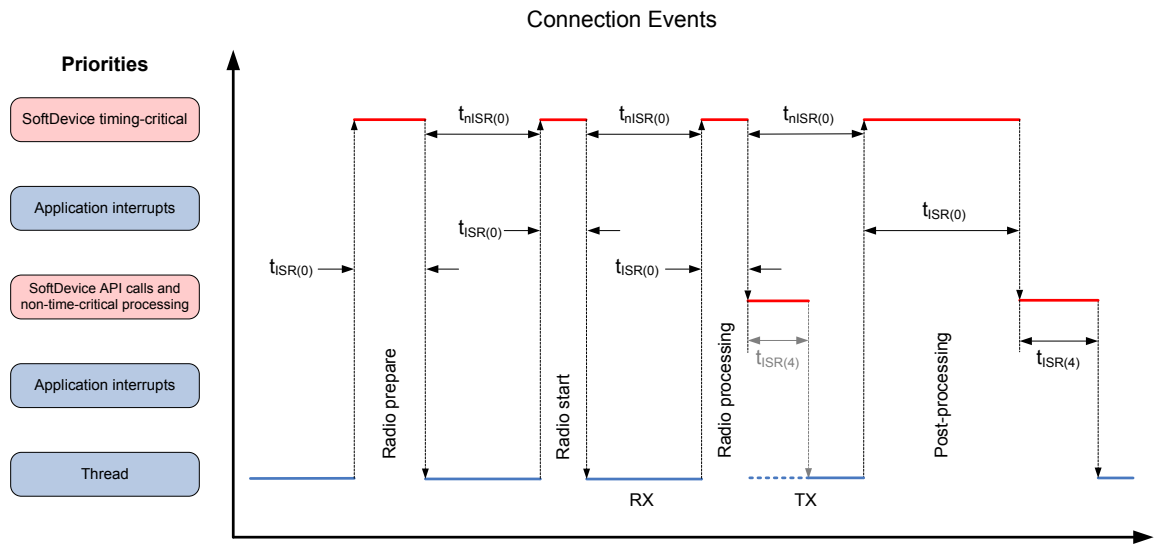

Figure 29: Peripheral connection events (some priority levels left out for clarity)

In a peripheral connection event, the pattern of SoftDevice processing activity at interrupt priority level 0 is typically as follows:

1. An interrupt (Radio prepare) sets up and prepares the software and hardware for the connection event.
2. A short interrupt occurs when the Radio starts listening for the first packet.
3. When the reception is complete, there is a radio processing interrupt that processes the received packet and switches the Radio to transmission.
4. When the transmission is complete, there is either a radio processing interrupt that switches the Radio back to reception (and possibly a new transmission after that), or the event ends with post processing.
5. After the radio and post processing in priority level 0, the SoftDevice processes any received data packets, executes any *GATT*, *ATT*, or *Security Manager Protocol (SMP)* operations, and generates events to the application as required in priority level 4. The interrupt at this priority level is therefore highly variable based on the stack operations executed.

SoftDevice processing activity for different priority levels during peripheral connection events is outlined in Table 34: Processor usage when connected on page 71. The typical case is seen when sending *GATT* write commands writing 20 bytes. The max case can be seen when sending and receiving maximum length packets while having a maximum number of connections and utilizing the Radio Timeslot API and Flash memory API at the same time.

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing when preparing the radio for a connection event		51 µs	65 µs
$t_{ISR(0),RadioStart}$	Processing when starting the connection event		18 µs	24 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending or receiving a packet		60 µs	67 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of a connection event		90 µs	250 µs
$t_{nISR(0)}$	Distance between interrupts during a connection event	183 µs	> 190 µs	
$t_{ISR(4)}$	Priority level 4 interrupt after a packet is sent or received		40 µs	

Table 34: Processor usage when connected

From the table we can calculate a typical processing time for a peripheral connection event where one packet is sent and received to be:

$$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + 2 * t_{ISR(4)} = 299\ \mu s$$

That means typically more than 99% of the processor time is available to the application when one peripheral link is established and one packet is sent in each direction with a 100 ms connection interval.

16.3.4 Interrupt latency when using multiple modules and roles

Concurrent use of the Flash API, Radio Timeslot API, and/or one or more Bluetooth Low Energy roles can affect interrupt latency.

The same interrupt priority levels are used by all Flash API, Radio Timeslot API, and Bluetooth Low Energy roles. When using more than one of these concurrently, their respective events can be scheduled back-to-back (see Scheduling on page 57 for more on scheduling). In those cases, the last interrupt in the activity by one module/role can be directly followed by the first interrupt of the next activity. Therefore, to find the real worst-case interrupt latency in these cases, the application developer must add the latency of the first and last interrupt for all combinations of roles that are used.

For example, if the application uses the Radio Timeslot API while having a Bluetooth Low Energy advertiser running, the worst-case interrupt latency or interruption for an application interrupt is the largest of the following SoftDevice interrupts having higher priority level (lower numerical value) than the application interrupt:

- the worst-case interrupt latency of the Radio Timeslot API
- the worst-case interrupt latency of the Bluetooth Low Energy advertiser role
- the sum of the max time of the first interrupt of the Radio Timeslot API and the last interrupt of the Bluetooth Low Energy advertiser role
- the sum of the max time of the first interrupt of the Bluetooth Low Energy advertiser role and the last interrupt of the Radio Timeslot API

17 Bluetooth Low Energy data throughput

This chapter outlines achievable Bluetooth Low Energy connection throughput for *GATT* procedures used to send and receive data in stated SoftDevice configurations.

The throughput numbers listed in this chapter are based on measurements in an interference-free radio environment. Maximum throughput is only achievable if the application, without delay, reads data packets as they are received and provides new data as packets are transmitted. The connection event length should be set to such a value that the entire connection event can be filled with packets. The SoftDevice may transfer as many packets as can fit within the connection event as specified by the event length for the connection. For example, in simplex communication, where data is transmitted in only one direction, more time will be available for sending packets. Therefore, there may be extra TX-RX packet pairs in connection events. Additionally, more time can be made available for a connection by extending the connection events beyond their reserved time. See Connection timing with Connection Event Length Extension on page 60 for more information.

The maximum data throughput numbers given in this chapter represent the maximum amount of data that can be transferred between two applications in a given time. The maximum throughput depends on the mechanism used to transfer data. When the application utilizes ATT Handle Value Notification or ATT Write Command, the transactions are one direction only. When the application utilizes ATT Write Request, it is assumed that the peer responds with an ATT Write Response in the next connection interval. The throughput will in this case be limited to one packet every second connection interval. The amount of data in each packet is the MTU size subtracted by the ATT header size. Therefore, the throughput can be expressed as follows:

*Throughput_bps = num_packets * (ATT_MTU - 3) * 8 / seconds*

All data throughput values apply to packet transfers over an encrypted connection using maximum payload sizes. Maximum *LL* payload size is 27 bytes unless noted otherwise.

The following table shows maximum data throughput at a connection interval of 7.5 ms for a single peripheral connection.

Protocol	ATT MTU size	Event length	Method	Maximum data throughput (LE 1M PHY)	Maximum data throughput (LE 2M PHY)
GATT Client	23	7.5 ms	Receive Notification	192.0 kbps	256.0 kbps
			Send Write command	192.0 kbps	256.0 kbps
			Send Write request	10.6 kbps	10.6 kbps
			Simultaneous receive Notification and send Write command	128.0 kbps (each direction)	213.3 kbps (each direction)
GATT Server	23	7.5 ms	Send Notification	192.0 kbps	256.0 kbps
			Receive Write command	192.0 kbps	256.0 kbps
			Receive Write request	10.6 kbps	10.6 kbps

Protocol	ATT MTU size	Event length	Method	Maximum data throughput (LE 1M PHY)	Maximum data throughput (LE 2M PHY)
			Simultaneous send Notification and receive Write command	128.0 kbps (each direction)	213.3 kbps (each direction)
GATT Server	158	7.5 ms	Send Notification	248.0 kbps	330.6 kbps
			Receive Write command	248.0 kbps	330.6 kbps
			Receive Write request	82.6 kbps	82.6 kbps
			Simultaneous send Notification and receive Write command	165.3 kbps (each direction)	275.5 kbps (each direction)
GATT Client	23	3.75 ms	Receive Notification	64.0 kbps	106.6 kbps
			Send Write command	64.0 kbps	106.6 kbps
			Send Write request	10.6 kbps	10.6 kbps
			Simultaneous receive Notification and send Write command	64.0 kbps (each direction)	85.3 kbps (each direction)
GATT Server	23	3.75 ms	Send Notification	64.0 kbps	106.6 kbps
			Receive Write command	64.0 kbps	106.6 kbps
			Receive Write request	10.6 kbps	10.6 kbps
			Simultaneous send Notification and receive Write command	64.0 kbps (each direction)	85.3 kbps (each direction)
GATT Client	23	2.5 ms	Receive Notification	42.6 kbps	64.0 kbps
			Send Write command	42.6 kbps	64.0 kbps
			Send Write request	10.6 kbps	10.6 kbps
			Simultaneous receive Notification and send Write command	21.3 kbps (each direction)	42.6 kbps (each direction)
GATT Server	23	2.5 ms	Send Notification	42.6 kbps	64.0 kbps
			Receive Write command	42.6 kbps	64.0 kbps
			Receive Write request	10.6 kbps	10.6 kbps
			Simultaneous send Notification and receive Write command	21.3 kbps (each direction)	42.6 kbps (each direction)

Table 35: Data throughput for a single connection with 23 byte ATT MTU

The following table shows the maximum data throughput for a single peripheral connection. The event length is equal to the connection interval.

Protocol	ATT MTU size	LL payload size[11]	Connection interval	Method	Maximum data throughput (LE 1M PHY)	Maximum data throughput (LE 2M PHY)
GATT Server	247	251	50 ms	Send Notification	702.8 kbps	1327.5 kbps
				Receive Write command	702.8 kbps	1327.5 kbps
				Simultaneous send Notification and receive Write command	390.4 kbps (each direction)	780.8 kbps (each direction)
GATT Server	247	251	400 ms	Send Notification	771.1 kbps	1376.2 kbps
				Receive Write command	760.9 kbps	1376.2 kbps
				Simultaneous send Notification and receive Write command	424.6 (each direction)	800.4 kbps (each direction)
Raw LL data	N/A	251	400 ms	N/A	803 kbps	1447.2 kbps

Table 36: Data throughput for a single connection with 247 byte ATT MTU

[11] Assumming that the peer device accepts the increased *ATT* and *LL* payload sizes.

18 Bluetooth Low Energy power profiles

The power profile diagrams in this chapter give an overview of the stages within a Bluetooth Low Energy Radio Event implemented by the SoftDevice. The profiles illustrate battery current versus time and briefly describe the stages that could be observed.

The profiles are based on typical events with empty packets. The Standby is a state of the SoftDevice where all Peripherals are IDLE.

The time the radio spends to transmit or receive a packet depends on the PHY. Using a higher data rate will decrease the radio time, while using a lower data rate will increase the radio time. Therefore, a higher data rate decreases power consumption, while a lower data rate increases power consumption.

Note: A higher data rate increases throughput but reduces the link budget and therefore the maximum range. A lower data rate decreases throughput but increases the link budget.

18.1 Advertising event

This section gives an overview of the power profile of the advertising event implemented in the SoftDevice. Figure 30: Advertising event on page 75 shows the event current profile of an advertising event consisting of three advertising packets sent on the primary advertising channels.

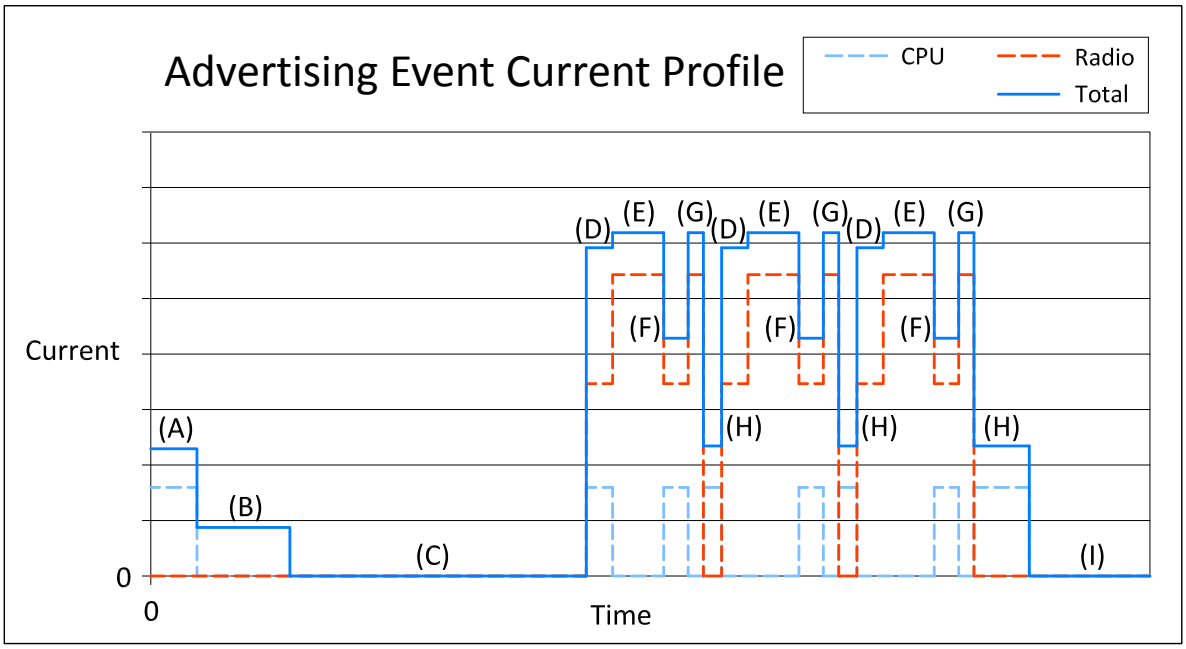

Figure 30: Advertising event

Stage	Description
(A)	Pre-processing (CPU)
(B)	Standby + HFXO ramp
(C)	Standby
(D)	Radio startup
(E)	Radio TX
(F)	Radio switch
(G)	Radio RX
(H)	Post-processing (CPU)
(I)	Standby

Table 37: Advertising event

18.2 Peripheral connection event

This section gives an overview of the power profile of the peripheral connection event implemented in the SoftDevice.

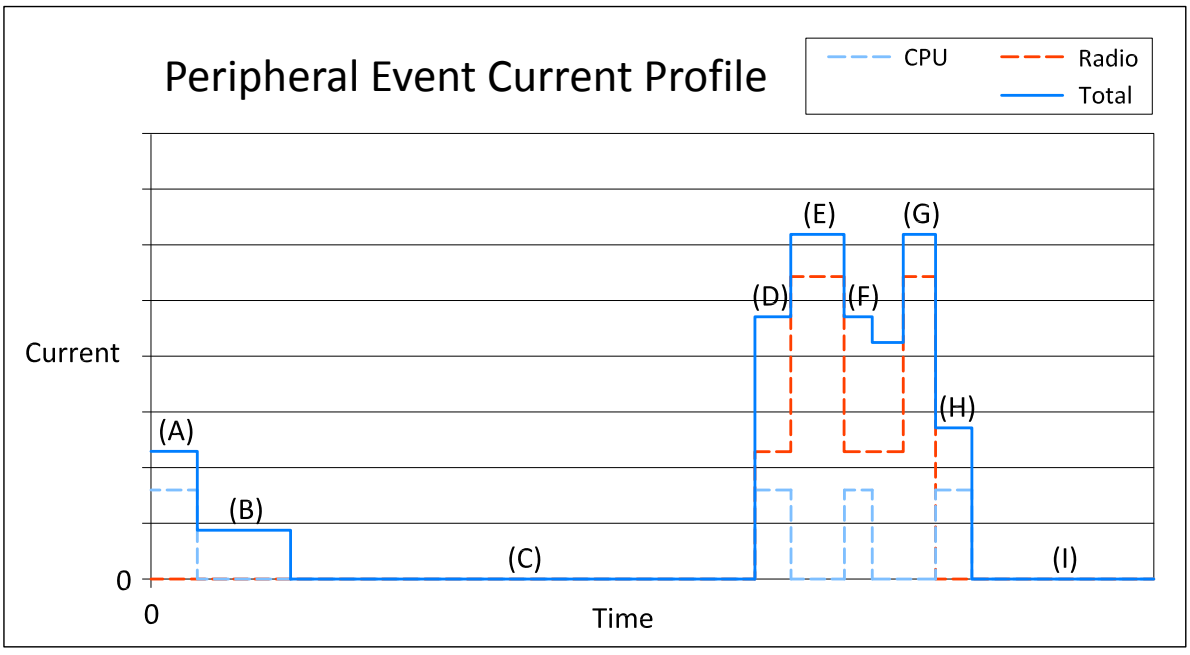

Figure 31: Peripheral connection event

Stage	Description
(A)	Pre-processing (CPU)
(B)	Standby + HFXO ramp
(C)	Standby
(D)	Radio startup
(E)	Radio RX
(F)	Radio switch
(G)	Radio TX
(H)	Post-processing (CPU)
(I)	Standby

Table 38: Peripheral connection event

19 SoftDevice identification and revision scheme

The SoftDevices are identified by the SoftDevice part code, a qualified IC partcode (for example, nRF52832), and a version string.

The identification scheme for SoftDevices consists of the following items:

- For revisions of the SoftDevice which are production qualified, the version string consists of major, minor, and revision numbers only, as described in the table below.
- 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_nrf51_1.2.3-4.alpha, where the major version is 1, minor version is 2, revision number is 3, build number is 4, and test qualification level is alpha. For more examples, see Table 40: SoftDevice revision examples on page 78.

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 minor 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 versions.

Table 39: Revision scheme

Sequence number	Description
s110_nrf51_1.2.3-1.alpha	Revision 1.2.3, first build, qualified at alpha level
s110_nrf51_1.2.3-2.alpha	Revision 1.2.3, second build, qualified at alpha level
s110_nrf51_1.2.3-5.beta	Revision 1.2.3, fifth build, qualified at beta level
s110_nrf51_1.2.3	Revision 1.2.3, qualified at production level

Table 40: SoftDevice revision examples

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
	  • Stable, but may not be feature complete and may contain known issues
	  • Protocol implementations are tested for conformance and interoperability

Production	• Qualified release suitable for production 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 41: Test qualification levels

19.1 Master boot record 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.

Glossary

Application Programming Interface (API)

A language and message format used by an application program to communicate with an operating system, application, or other service.

Attribute Protocol (ATT)

"The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client." *Bluetooth Core Specification, Version 5.1, Vol 3, Part F, Section 1.1*

Cortex Microcontroller Software Interface Standard (CMSIS)

A vendor-independent hardware abstraction layer for the Cortex-M processor series that defines generic tool interfaces.

Device Firmware Update (DFU)

A mechanism for upgrading the firmware of a device.

Floating-Point Unit (FPU)

A part of a CPU specially designed to perform operations on floating point numbers.

Generic Access Profile (GAP)

"The Bluetooth system defines a base profile which all Bluetooth devices implement. This profile is the Generic Access Profile (GAP), which defines the basic requirements of a Bluetooth device." *Bluetooth Core Specification, Version 5.1, Vol 1, Part A, Section 6.2*

Generic Attribute Protocol (GATT)

"Generic Attribute Profile (GATT) is built on top of the Attribute Protocol (ATT) and establishes common operations and a framework for the data transported and stored by the Attribute Protocol." *Bluetooth Core Specification, Version 5.1, Vol 1, Part A, Section 6.4*

Human Interface Device (HID)

Type of a computer device that interacts directly with, and most often takes input from, humans and may deliver output to humans. The term "HID" most commonly refers to the USB-HID specification.

Integrated Circuit (IC)

A semiconductor chip consisting of fabricated transistors, resistors, and capacitors.

Link Layer (LL)

"A control protocol for the link and physical layers that is carried over logical links in addition to user data." *Bluetooth Core Specification, Version 5.1, Vol 1, Part A, Section 1.2*

Low-Noise Amplifier (LNA)

In a radio receiving system, an electronic amplifier that amplifies a very low-power signal without significantly degrading its signal-to-noise ratio.

Logical Link Control and Adaptation Protocol (L2CAP)

"Provides a channel-based abstraction to applications and services. It carries out segmentation and reassembly of application data and multiplexing and de-multiplexing of multiple channels over a shared logical link." *Bluetooth Core Specification, Version 5.1, Vol 1, Part A, Section 1.2*

Main Stack Pointer (MSP)

The default stack pointer. By default, the nRF52 has a shared call stack for the application and the SoftDevice, managed by the MSP.

Man-in-the-Middle (MITM)

A man-in-the-middle attack is a form of eavesdropping where communication between two devices is monitored and modified by an unauthorized party who relays information between the two devices giving the illusion that they are directly connected.

Memory Watch Unit (MWU)

A peripheral that can be used to generate events when a memory region is accessed by the CPU.

Power Amplifier (PA)

A device used to increase the transmit power level of a radio signal.

Programmable Peripheral Interconnect (PPI)

Enables peripherals to interact autonomously with each other using tasks and events independent of the CPU.

Process Stack Pointer (PSP)

A separate stack pointer that can be used for application threads. This is an optional configuration, but it may be required if using an RTOS.

Qualified Design Identification (QDID)

A unique identifier assigned to a design that has completed Bluetooth Qualification.

Software Development Kit (SDK)

A set of tools used for developing applications for a specific device or operating system.

SoftDevice Manager (SDM)

A SoftDevice component that controls the SoftDevice state and configures the behavior of certain core functionality.

Security Manager (SM)

Provides means for bonding devices, encrypting and decrypting data, and enabling device privacy.

Security Manager Protocol (SMP)

A protocol used for pairing and key distribution.

System on Chip (SoC)

A microchip that integrates all the necessary electronic circuits and components of a computer or other electronic systems on a single integrated circuit.

Supervisor Call (SVC)

Generates a software exception in which access to system resources or privileged operations can be provided.

Acronyms and abbreviations

These acronyms and abbreviations are used in this document.

API
Application Programming Interface

ATT
Attribute Protocol

CMSIS
Cortex Microcontroller Software Interface Standard

DFU
Device Firmware Update

FPU
Floating-Point Unit

GAP
Generic Access Profile

GATT
Generic Attribute Protocol

GPIO
General-Purpose Input/Output

GPIOTE
General-Purpose Input/Output Tasks and Events

HFCLK
High-Frequency Clock

HFXO
High-Frequency Crystal Oscillator

HID
Human Interface Device

IC
Integrated Circuit

IRQ
Interrupt Request

LFCLK
Low-Frequency Clock

LL
Link Layer

LNA
Low-Noise Amplifier

L2CAP
Logical Link Control and Adaptation Protocol

MBR
Master Boot Record

MITM
Man-in-the-Middle

MSP
Main Stack Pointer

MTU
Maximum Transmission Unit

MWU
Memory Watch Unit

NVIC
Nested Vectored Interrupt Controller

PA
Power Amplifier

PDU
Packet Data Unit

PPI
Programmable Peripheral Interconnect

PSP
Process Stack Pointer

QDID
Qualified Design Identification

RC
Resistor-Capacitor

SDK
Software Development Kit

SDM
SoftDevice Manager

SM
Security Manager

SMP
Security Manager Protocol

SoC
System on Chip

SVC
Supervisor Call

UUID
: Universally Unique Identifier

Legal notices

By using this documentation you agree to our terms and conditions of use. Nordic Semiconductor may change these terms and conditions at any time without notice.

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.

Nordic Semiconductor ASA does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. If there are any discrepancies, ambiguities or conflicts in Nordic Semiconductor's documentation, the Product Specification prevails.

Nordic Semiconductor ASA reserves the right to make corrections, enhancements, and other changes to this document without notice.

Life support applications

Nordic Semiconductor 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.

RoHS and REACH statement

Complete hazardous substance reports, material composition reports and latest version of Nordic's REACH statement can be found on our website www.nordicsemi.com.

Trademarks

All trademarks, service marks, trade names, product names, and logos appearing in this documentation are the property of their respective owners.

Copyright notice

COMPANY WITH
QUALITY SYSTEM
CERTIFIED BY DNV GL
= ISO 9001 =