# S122 SoftDevice

**SoftDevice Specification**

v1.0

NORDIC®
SEMICONDUCTOR

# Contents

NORDIC
SEMICONDUCTOR

NORDIC
SEMICONDUCTOR

# Revision history

| Date | Version | Description |
|------|---------|-------------|
| July 2020 | 1.0 | First release |

NORDIC
SEMICONDUCTOR

# 1   S122 SoftDevice

The S122 SoftDevice is a *Bluetooth*® Low Energy central protocol stack solution. It supports up to eight central connections with an additional Observer role running concurrently. The S122 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 5.2 compliant single-mode Bluetooth Low Energy protocol stack<br>   • Up to eight central connections and one Observer running concurrently<br>   • Configurable number of connections and connection properties<br>   • Configurable attribute table size<br>   • Custom UUID support<br>   • Link layer supporting LE 1M PHY and LE 2M PHY<br>   • LL Privacy<br>   • ATT and SM protocols<br>   • LE Secure Connections pairing model<br>   • GATT and GAP APIs<br>   • GATT Client and Server<br>   • Configurable ATT MTU<br>• Complementary nRF5 SDK including Bluetooth profiles and example applications<br>• Master boot record for over-the-air device firmware update<br>   • SoftDevice, application, and bootloader can be updated separately<br>• Thread-safe supervisor-call based API<br>• Asynchronous, event-driven behavior<br>• No RTOS dependency<br>   • Any RTOS can be used<br>• No link-time dependencies<br>   • Standard Arm® Cortex®- M4 project configuration for application development<br>• Support for concurrent and non-concurrent multiprotocol operation<br>   • Concurrent with the Bluetooth stack using Radio Timeslot API<br>   • Alternate protocol stack in application space<br>• Support for control of external power amplifiers and low noise amplifiers<br>• Quality of service feature for channel monitoring | • HID USB dongle<br>• Gateways<br>• Smart home<br>• Asset tracking |

NORDIC
SEMICONDUCTOR

# 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)*<br>• Errata: Contains information on anomalies related to the *IC*<br>• 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 5.2, Volumes 1, 3, 4, and 6, describe Bluetooth terminology which is used throughout the SoftDevice Specification |

*Table 1: Additional documentation*

# 3 Product overview

The S122 SoftDevice is a precompiled and linked binary image implementing a Bluetooth 5.2 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.
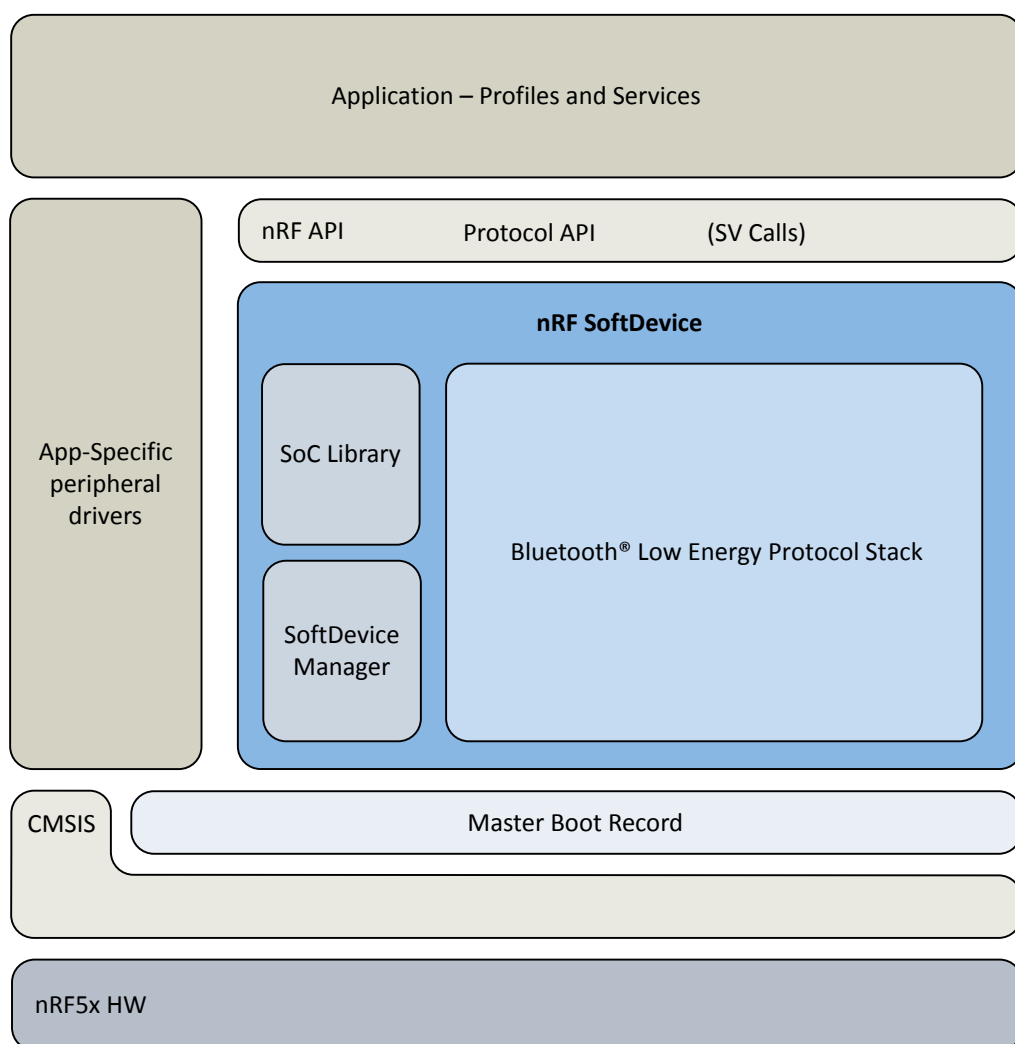


*Figure 1: SoC application with the SoftDevice*

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 5.2 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

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 82). 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 68.

## 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 against SoftDevice memory protection rules

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.

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 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 67 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 67.

## 5.2 Clock source

The SoftDevice can use one of three available LFCLK sources: the internal RC oscillator, external crystal oscillator, or synthesized low-frequency clock. The synthesized low-frequency clock is not tested or intended for use with the Bluetooth Low Energy protocol stack.

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`).

# 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

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.

> **Note:** The S122 SoftDevice does not enable the protection of the SoftDevice RAM and peripherals, as the *Memory Watch Unit (MWU)* peripheral is not available through the S122 SoftDevice *API*. Writing to the SoftDevice RAM and peripherals will lead to undefined behavior.

The following figure presents an overview of the regions.

NORDIC
SEMICONDUCTOR

**Program Memory**

**RAM**
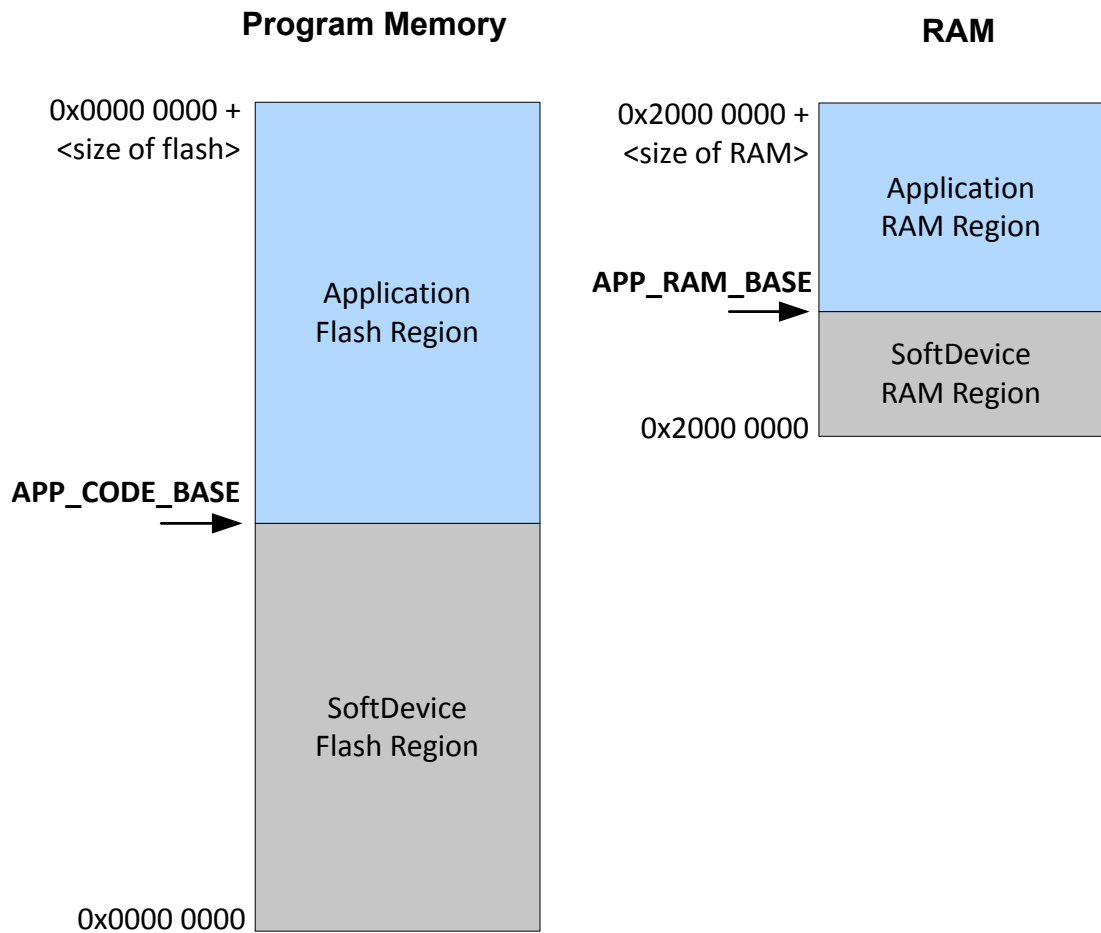


*Figure 2: Memory region designation*

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 19: Memory resource map on page 54:

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

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.

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.

NORDIC
SEMICONDUCTOR

**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.<br><br>**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:<br><br>• Access to RESETREAS register<br>• Set power modes<br>• Configure power fail comparator<br>• Control RAM block power<br>• Use general purpose retention register<br>• Configure DC/DC converter state:<br> • DISABLED<br> • 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.[1] |
| 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. |

NORDIC
SEMICONDUCTOR

| Feature | Description |
|---|---|
| Flash memory API | Application access to flash write, erase, and protect. Can be safely used during all protocol stack states.[1] See Flash memory API on page 22. |
| Temperature | Application access to the temperature sensor |
| USB power API | *API* for enabling and disabling USB power events and for reading the USBREGSTATUS register |

*Table 2: SoC features*

---

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

NORDIC
SEMICONDUCTOR

# 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. When the SoftDevice is enabled, it shall only be accessed through the SoftDevice API. Through this *API*, the application has limited access. The S122 SoftDevice will not prevent the application from accessing the peripheral directly. Doing so will lead to undefined behavior. |
| Blocked | The hardware peripheral is used by the SoftDevice and must never be accessed by the application. Doing so will lead to undefined behavior. 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 |

NORDIC
SEMICONDUCTOR

| Instance | Access<br><br>SoftDevice enabled | Access<br><br>SoftDevice disabled |
|---|---|---|
| RADIO | Blocked[2] | Open |
| TIMER0 | Blocked[2] | Open |
| RTC0 | Blocked | Open |
| TEMP | Restricted | Open |
| RNG | Restricted | Open |
| ECB | Restricted | Open |
| CCM | Blocked[3] | Open |
| AAR | Blocked[3] | Open |
| EGU1/SWI1/Radio Notification | Restricted[4] | Open |
| EGU5/SWI5 | Blocked | Open |
| ACL | Restricted | Open |
| NVMC | Restricted | Open |
| FICR | Blocked | Blocked |
| UICR | Restricted | Open |
| NVIC | Restricted[5] | 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.

---

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

[3] 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.

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

[5] 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.

NORDIC
SEMICONDUCTOR

| 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).

NORDIC
SEMICONDUCTOR

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

*Table 6: SVC number allocation*

## 7.5 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.

NORDIC
SEMICONDUCTOR

# 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 64 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 |
|---|---|
| All possible Bluetooth Low Energy roles running concurrently (connections as a Central and Observer) | Low to medium probability of flash operation success<br><br>Probability of success increases with:<br><br>• Configurations with shorter event lengths<br>• Lower data traffic<br>• Decrease in scan window<br>• Increase in scan interval |
| 8 high bandwidth connections as a Central<br><br>All active connections fulfill the following criteria:<br><br>• Supervision time-out > 6 x connection interval<br>• Connection interval ≥ 150 ms<br>• All connections have the same connection interval | High probability of flash operation success |

NORDIC
SEMICONDUCTOR

| Bluetooth Low Energy activity | Flash write/erase |
|---|---|
| 8 low bandwidth connections as a Central<br><br>All active connections fulfill the following criteria:<br><br>• Supervision time-out > 6 x connection interval<br>• Connection interval ≥ 110 ms<br>• All connections have the same connection interval | 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

The SoftDevice's multiprotocol support facilitates radio activity coordination with other collocated wireless protocols.

Developers can 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 64 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 26: 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 S122 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 68.

| 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 68), *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.

## 9.2.8 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.

### 9.2.8.1 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.
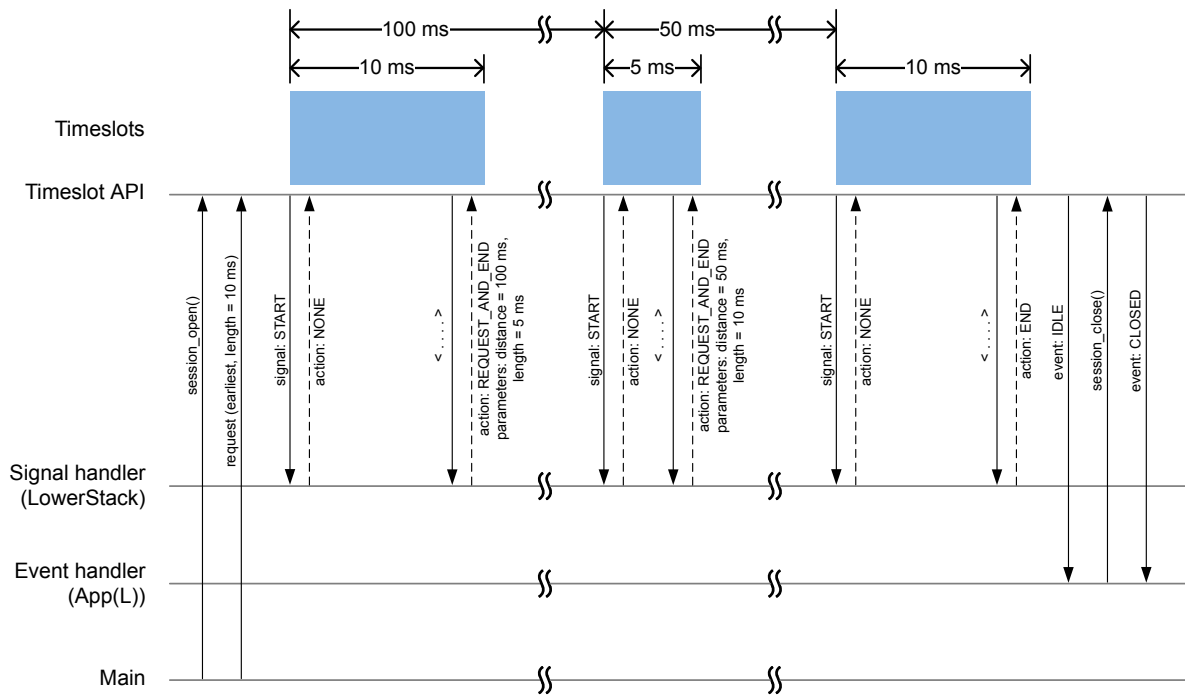
NORDIC®
SEMICONDUCTOR

*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 68 for the available interrupt levels.

## 9.2.8.2 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.
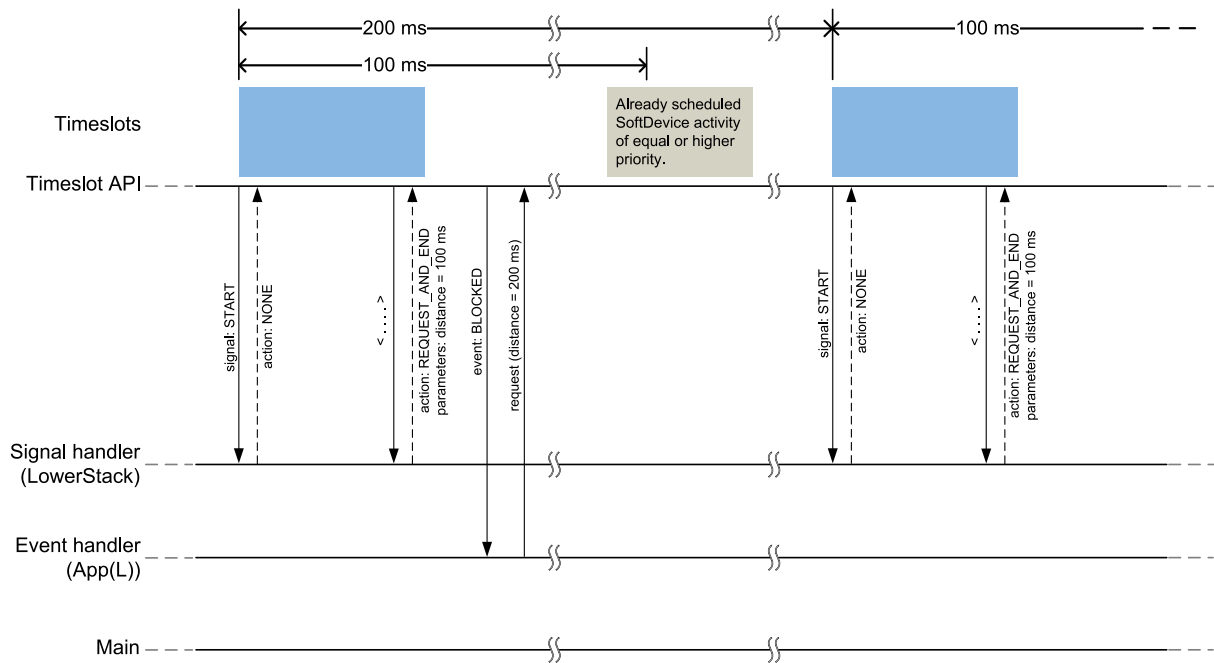
*Figure 4: Blocked timeslot scenario*

## 9.2.8.3 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.
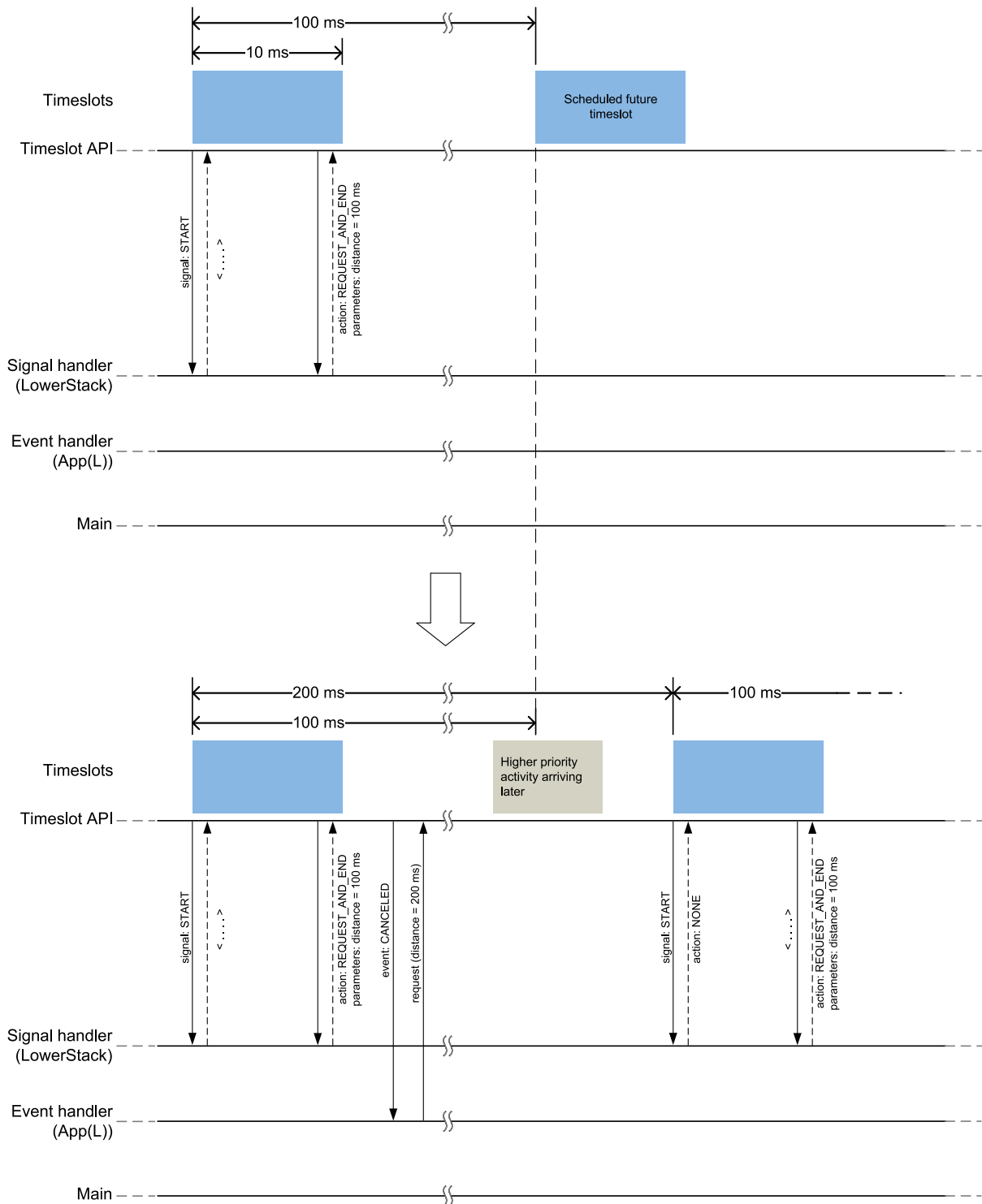
*Figure 5: Canceled timeslot scenario*

## 9.2.8.4 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.
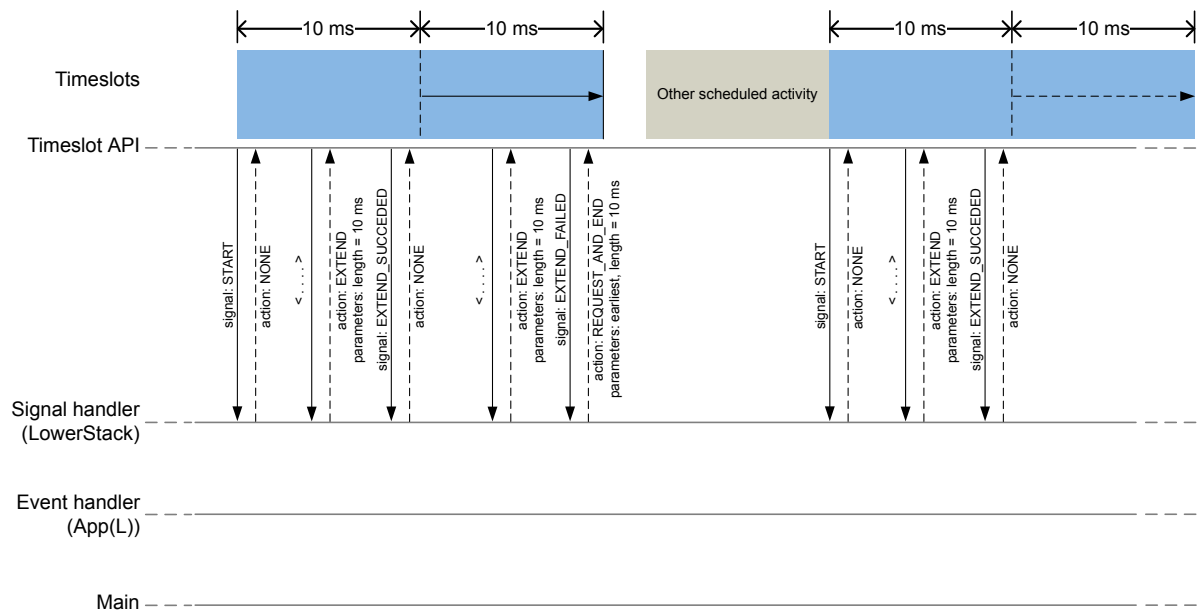


*Figure 6: Radio Timeslot extension example*

# 10 Bluetooth Low Energy protocol stack

The Bluetooth 5.2 compliant Host and Controller implemented by the SoftDevice are fully qualified with multirole support (Central, Initator, and Observer).

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*.
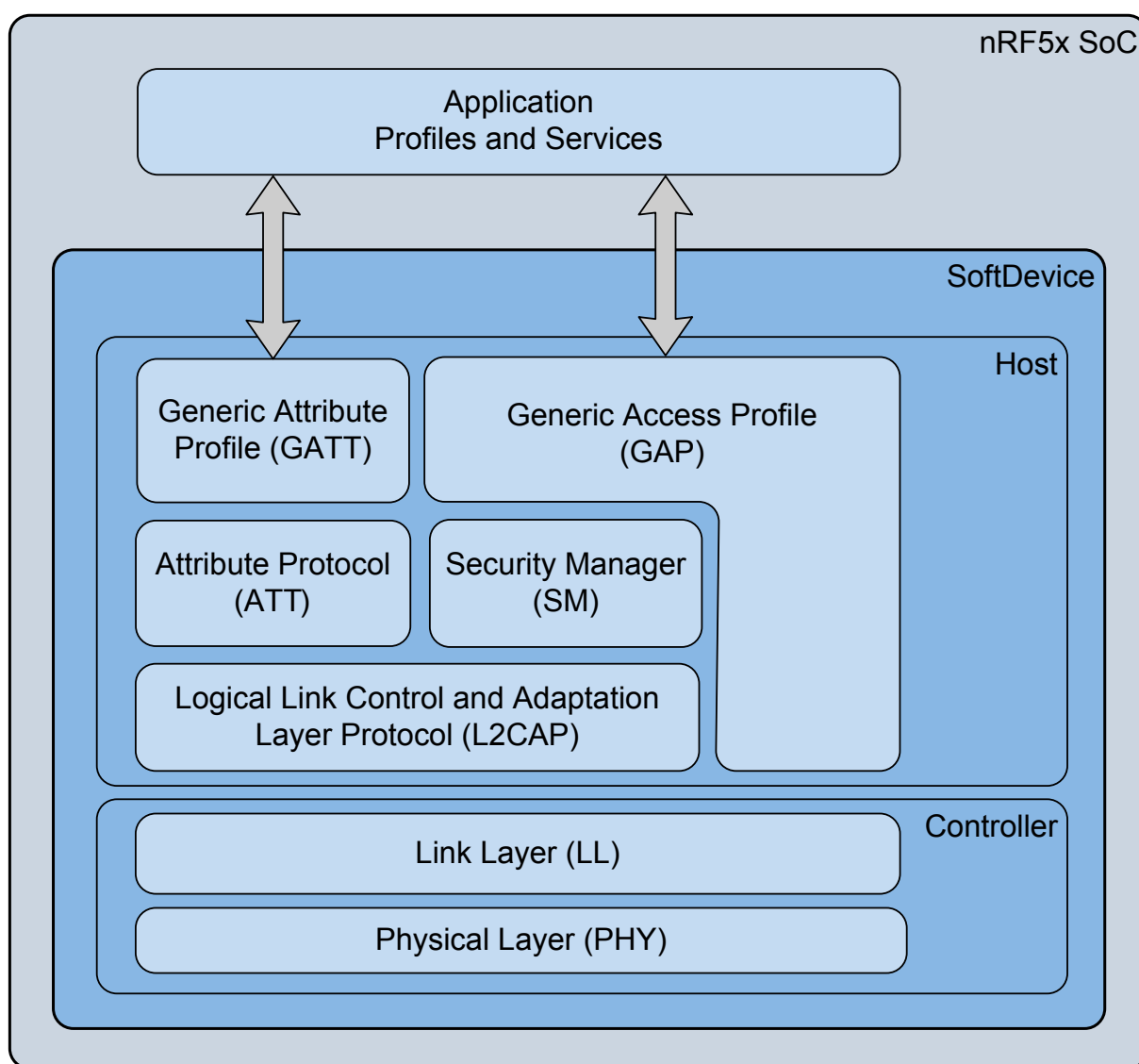


*Figure 7: SoftDevice stack architecture*

# 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*<br>Battery<br>Device Information |
| Heart Rate | Heart Rate<br>Device Information |
| Proximity | Link Loss<br>Immediate Alert<br>TX Power |
| Blood Pressure | Blood Pressure<br>Device Information |
| Health Thermometer | Health Thermometer<br>Device Information |
| Glucose | Glucose<br>Device Information |
| Phone Alert Status | Phone Alert Status |
| Alert Notification | Alert Notification |
| Time | Current Time<br>Next DST Change<br>Reference Time Update |
| Find Me | Immediate Alert |
| Cycling Speed and Cadence | Cycling Speed and Cadence<br>Device Information |
| Running Speed and Cadence | Running Speed and Cadence<br>Device Information |
| Location and Navigation | Location and Navigation |
| Cycling Power | Cycling Power |
| Scan Parameters | Scan Parameters |

NORDIC
SEMICONDUCTOR

| Adopted profile | Adopted services |
|---|---|
| Weight Scale | Weight Scale<br><br>Body Composition<br><br>User Data<br><br>Device Information |
| Continuous Glucose Monitoring | Continuous Glucose Monitoring<br><br>Bond Management<br><br>Device Information |
| Environmental Sensing | Environmental Sensing |
| Pulse Oximeter | Pulse Oximeter<br><br>Device Information<br><br>Bond Management<br><br>Battery<br><br>Current Time |
| Automation IO | Automation IO |
| | Indoor Positioning |
| Fitness Machine Profile | Fitness Machine<br><br>Device Information<br><br>User Data |
| Reconnection Configuration Profile | Reconnection Configuration Service |
| Binary Sensor Profile | Binary Sensor Service |
| Emergency Profile | Emergency Configuration Service |
| Insulin Delivery Profile | Insulin Delivery Service<br><br>Device Information Service<br><br>Current Time Service<br><br>Battery Service<br><br>Immediate Alert Service<br><br>Bond Management Service |
| | HTTP Proxy 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.

NORDIC
SEMICONDUCTOR

## 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 |

*Table 13: API features in the Bluetooth Low Energy stack*

| GAP features | Description |
|---|---|
| Multirole | Initiator and Observer can run concurrently with connections. |
| Multiple bond support | Keys and peer information stored in application space.<br><br>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*

NORDIC
SEMICONDUCTOR

| GATT features | Description |
|---|---|
| *GATT* Server | Support for one *ATT* server for all connections<br><br>Includes configurable Service Changed support |
| Support for authorization | Enables control points<br><br>Enables the application to provide fresh data<br><br>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<br><br>Discover all Primary Services<br><br>Discover Primary Service by Service UUID<br><br>Find included Services<br><br>Discover All Characteristics of a Service<br><br>Discover Characteristics by UUID<br><br>Discover All Characteristic Descriptors<br><br>Read Characteristic Value<br><br>Read using Characteristic UUID<br><br>Read Long Characteristic Values<br><br>Read Multiple Characteristic Values (Client only)<br><br>Write Without Response<br><br>Write Characteristic Value<br><br>Notifications<br><br>Indications<br><br>Read Characteristic Descriptors<br><br>Read Long Characteristic Descriptors<br><br>Write Characteristic Descriptors<br><br>Write Long Characteristic Values<br><br>Write Long Characteristic Descriptors<br><br>Reliable Writes |

*Table 15: GATT features in the Bluetooth Low Energy stack*

NORDIC
SEMICONDUCTOR

| 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: <br><br> 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*

| LL features | Description |
|---|---|
| Master role <br><br> Scanner/initiator role | The SoftDevice supports multiple concurrent central connections and an additional scanner or initiator role. The initiator role is not available when the number of available simultaneous connections has been reached, but the scanner role is still supported for new device discovery. |
| Channel map configuration | Setup of channel map for all central connections from the application. |
| Master-initiated connection parameter update | Central role may initiate connection parameter update. |
| LE 1M PHY <br><br> 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. Central role is able to initiate a PHY update procedure and respond to a peer-initiated PHY update procedure. |
| Encryption | |
| LE Ping | |
| Privacy | The *LL* can generate and resolve resolvable private addresses in the scanner, and initiator roles. |

*Table 18: LL features in the Bluetooth Low Energy stack*

NORDIC
SEMICONDUCTOR

| Proprietary features | Description |
|---|---|
| TX Power control | Access for the application to change transmit power settings for a specific role or connection handle. |
| MBR for *Device Firmware Update (DFU)* | Enables over-the-air firmware replacement, including full SoftDevice update capability. |
| *Quality of Service (QoS)* channel survey | Measures the energy level of Bluetooth Low Energy channels. The application can then set an adapted channel map to avoid busy channels. |
| *QoS* RSSI | Channel-specific signal strength measurements during scanning and central connections. |
| *QoS* Event Report | Generates a report on every connection event. |
| Channel map for Observer role | Access for the application to set a channel map for the Observer role. This can be used to avoid busy or uninteresting channels. |

*Table 19: Proprietary features in the Bluetooth Low Energy stack*

## 10.3 Limitations on procedure concurrency

When the SoftDevice has established multiple connections as a Central, the concurrency of protocol procedures will have some limitations.

The Host instantiates both *GATT* and *GAP* instances for each connection, while the *SM* Initiator has a configurable number of instantiations. The *LL* also has concurrent procedure limitations that are handled inside the SoftDevice without requiring management from the application.

| Protocol procedures | Limitation with multiple connections |
|---|---|
| *GATT* | None. All procedures can be executed in parallel. |
| *GAP* | None. All procedures can be executed in parallel. Note that some *GAP* procedures require *LL* control procedures (connection parameter update and encryption). In this case, the *GAP* module will queue the *LL* procedures and execute them in sequence. |
| *SM* | None. The number of concurrent procedures for central connections are fully configurable. |
| *LL* | The *LL* Disconnect procedure has no limitations and can be executed on any or all links simultaneously. The *LL* connection parameter update on a master link can only be executed on one master link at a time. |

*Table 20: Limitations on procedure concurrency*

NORDIC
SEMICONDUCTOR

# 10.4 Bluetooth Low Energy role configuration

The S122 SoftDevice stack supports concurrent operation in multiple Bluetooth Low Energy roles. The roles available can be configured when the S122 SoftDevice stack is enabled at runtime.

The SoftDevice provides a mechanism for enabling the number of central  roles the application can run concurrently, and for enabling *QoS* channel survey. The SoftDevice can be configured with multiple connections as a Central. The SoftDevice supports running one Scanner or Observer concurrently with the Bluetooth Low Energy connections.

An Initiator 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 will also allocate memory for the *QoS* channel survey if the application has enabled it.

The SoftDevice supports per connection bandwidth configuration by giving the application control over 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 63 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 65 for details about recommended configurations.

NORDIC
SEMICONDUCTOR

# 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[6].

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.
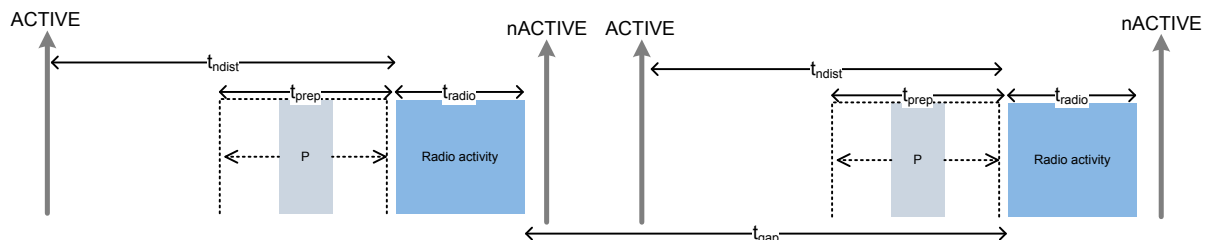


*Figure 8: Two radio events with ACTIVE and nACTIVE signals*

---

[6]  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.
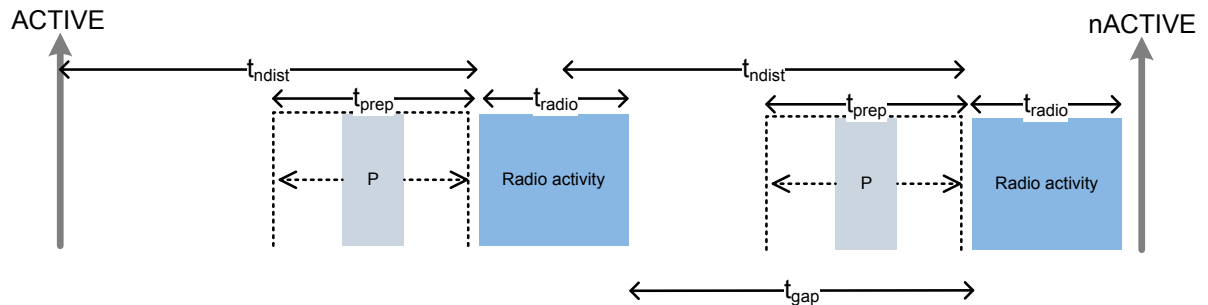


*Figure 9: Two radio events without ACTIVE and nACTIVE signals between the events*

| 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.<br><br>**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 | |

NORDIC
SEMICONDUCTOR

| 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): 200, 425, 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 connected roles on page 44.

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 23: 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 $t_{prep}$.

NORDIC
SEMICONDUCTOR

## 11.2 Radio Notification on connection events as a Central

This section clarifies the functionality of the Radio Notification feature when the SoftDevice operates as a Bluetooth Low Energy Central. The behavior of the notification signals is shown under various combinations of active central links and scanning events.

See Table 21: Radio Notification notation and terminology on page 43 for the notations used in the text and the figures of this section. For a comprehensive understanding of role scheduling, see Scheduling on page 57.

For a central link, multiple packets may be exchanged within a single Radio (connection) Event. This is shown in the following figure.
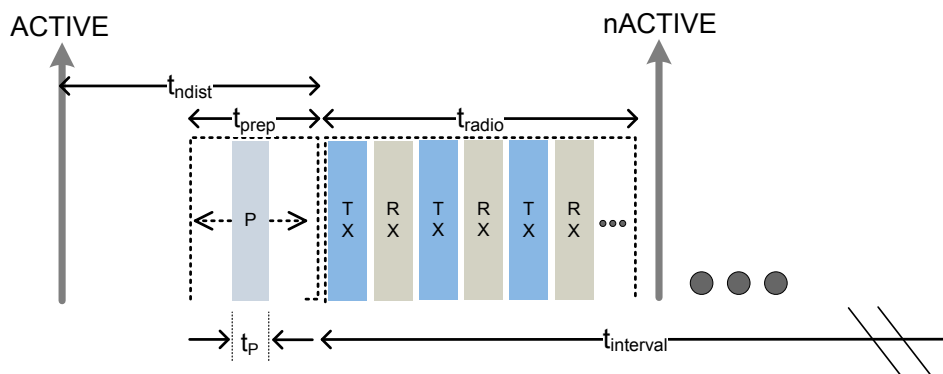


*Figure 10: Central link with multiple packet exchange per connection event*

To ensure that the ACTIVE notification signal will be available to the application at the configured time when a single central link is established (Figure 11: Radio Notification signal in relation to a single active link on page 45), the following condition must hold:

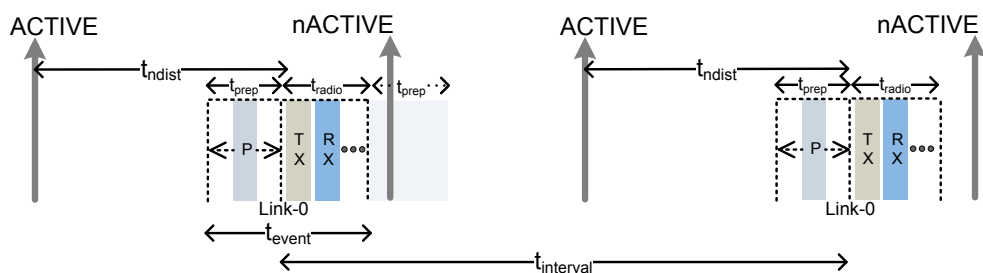$$t_{ndist} + t_{event} - t_{prep} < t_{interval}$$



*Figure 11: Radio Notification signal in relation to a single active link*

A SoftDevice operating as a Central may establish multiple central links and schedule them back-to-back in each connection interval. An example of a Central with three links is shown in Figure 12: Radio Notification signal with three active Central links on page 46. To ensure that the ACTIVE notification signal will be available to the application at the configured time when three links are established as a Central, the following condition must hold:

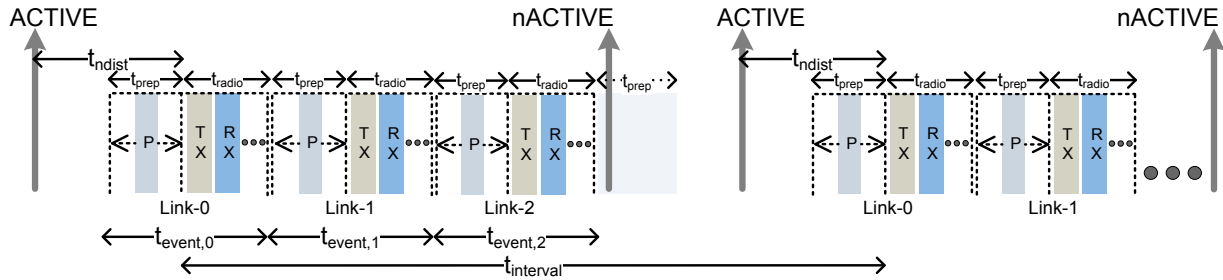$$t_{ndist} + t_{event,0} + t_{event,1} + t_{event,2} - t_{prep} < t_{interval}$$

*Figure 12: Radio Notification signal with three active Central links*

In case one or several central links are dropped, an idle time interval will exist between active central links. If the interval is sufficiently long, the application may unexpectedly receive the Radio Notification signal. In particular, the notification signal will be available to the application in the idle time interval, if this interval is longer than $t_{ndist}$. This can be expressed as:

$$\Sigma_{\texttt{i=m, ...n}}\ t_{event,i} + t_{prep} > t_{ndist}$$

where Link-m, ..., Link-n are consecutive inactive central links.

For example, in the scenario shown in Figure 13: Radio Notification signal with two active Central links on page 46, Link-1 is not active and a gap of $t_{event,1}$ time units (e.g. ms) exists between Link-0 and Link-2. Consequently, the ACTIVE notification signal will be available to the application, if the following condition holds:

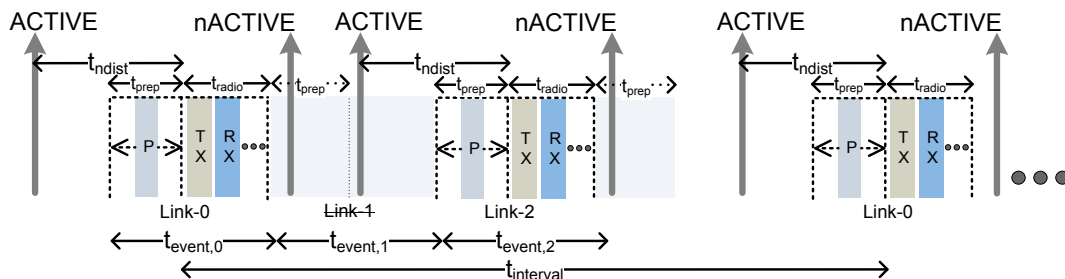$$t_{event,1} + t_{prep} > t_{ndist}$$



*Figure 13: Radio Notification signal with two active Central links*

A SoftDevice may additionally run a Scanner in parallel to the central links. This is shown in Figure 14: Radio Notification signals for three active central connections while scanning on page 46, where three central links and a Scanner have been established. To guarantee in this case that the ACTIVE notification signal will be available to the application at the configured time, the following condition must hold:

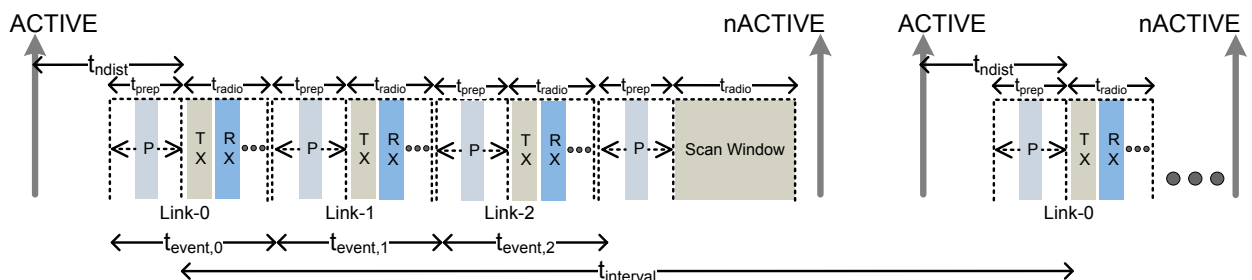$$t_{ndist} + t_{event,0} + t_{event,1} + t_{event,2} + Scan\ Window < t_{interval}$$



*Figure 14: Radio Notification signals for three active central connections while scanning*

NORDIC
SEMICONDUCTOR

# 11.3 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 central links, Radio Notification does not impose limits on how long the Radio event ($t_{radio}$) may be extended. This implies that all idle time in between connection events can be used for event extension. Because of this, the ACTIVE signal and nACTIVE signals between connection events cannot be guaranteed when Connection Event Length Extension is enabled. The following figure shows an example of how the idle time between connection events can be utilized when Connection Event Length Extension and Radio Notification signals are enabled.
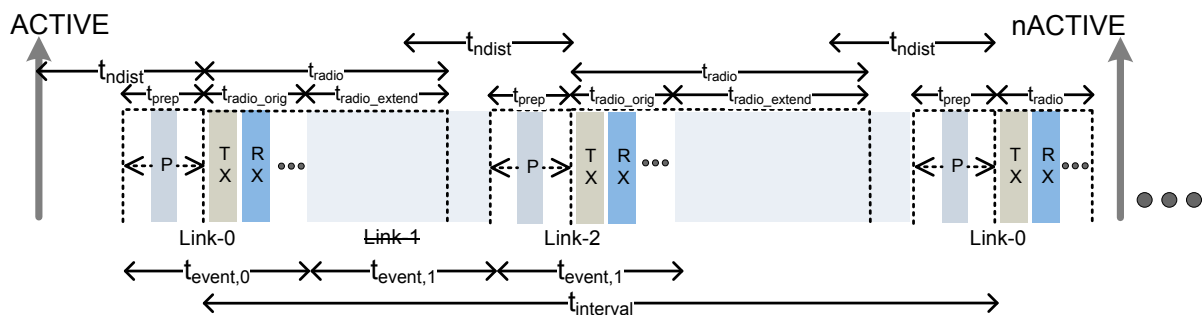


*Figure 15: Connection Event Length Extension with central links*

# 11.4 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.
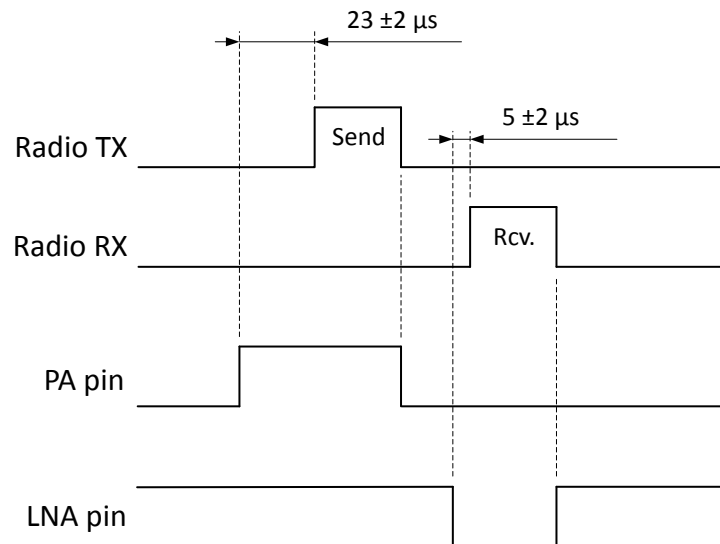
*Figure 16: PA/LNA and radio activity timing*

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 67.

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 on page 13) for storing the MBR parameters. The address of this flash page is referred to as MBRPARAMADDR (see Figure 17: 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 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 17: 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 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.
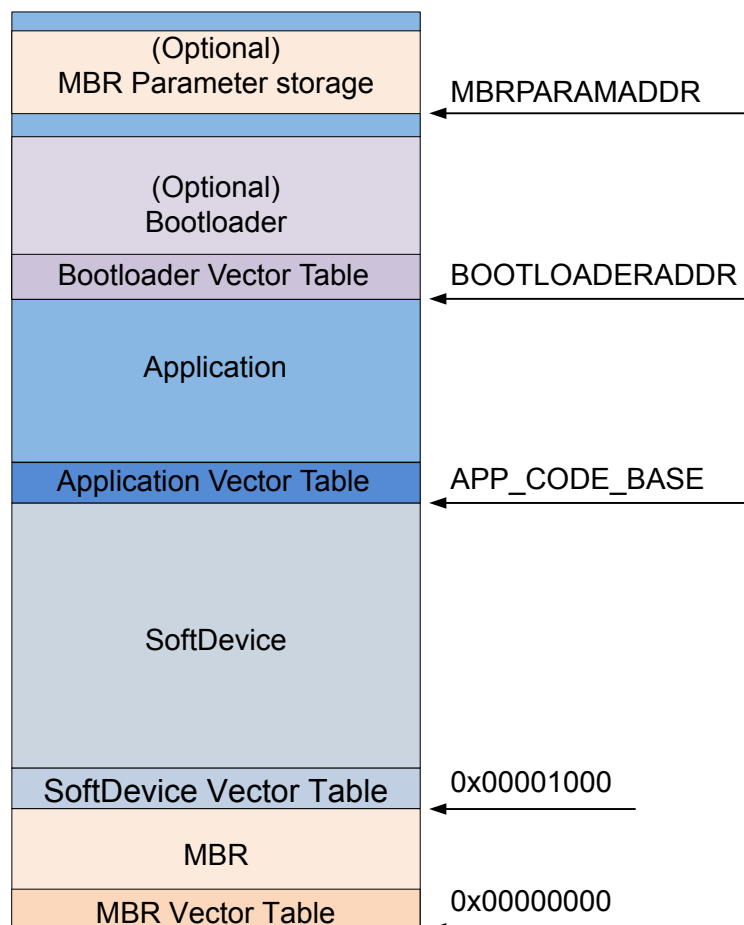


*Figure 17: MBR, SoftDevice, and bootloader architecture*

# 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.

NORDIC
SEMICONDUCTOR

# 13 SoftDevice information structure

The SoftDevice binary file contains an information structure.

The structure is illustrated in Figure 18: 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.
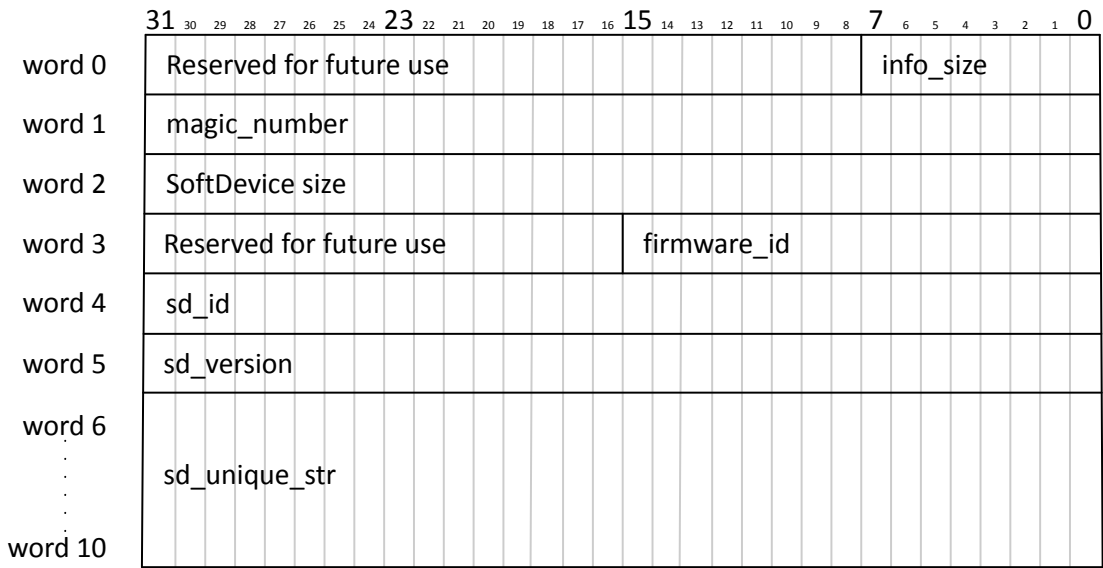
| | 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|
| word 0 | Reserved for future use | info_size |
| word 1 | magic_number |
| word 2 | SoftDevice size |
| word 3 | Reserved for future use | firmware_id |
| word 4 | sd_id |
| word 5 | sd_version |
| word 6 ⋮ word 10 | sd_unique_str |

*Figure 18: 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.

NORDIC
SEMICONDUCTOR

# 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)
- Security (see Security configuration on page 56)
- 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 19: 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 24: S122 Memory resource requirements for RAM on page 54.
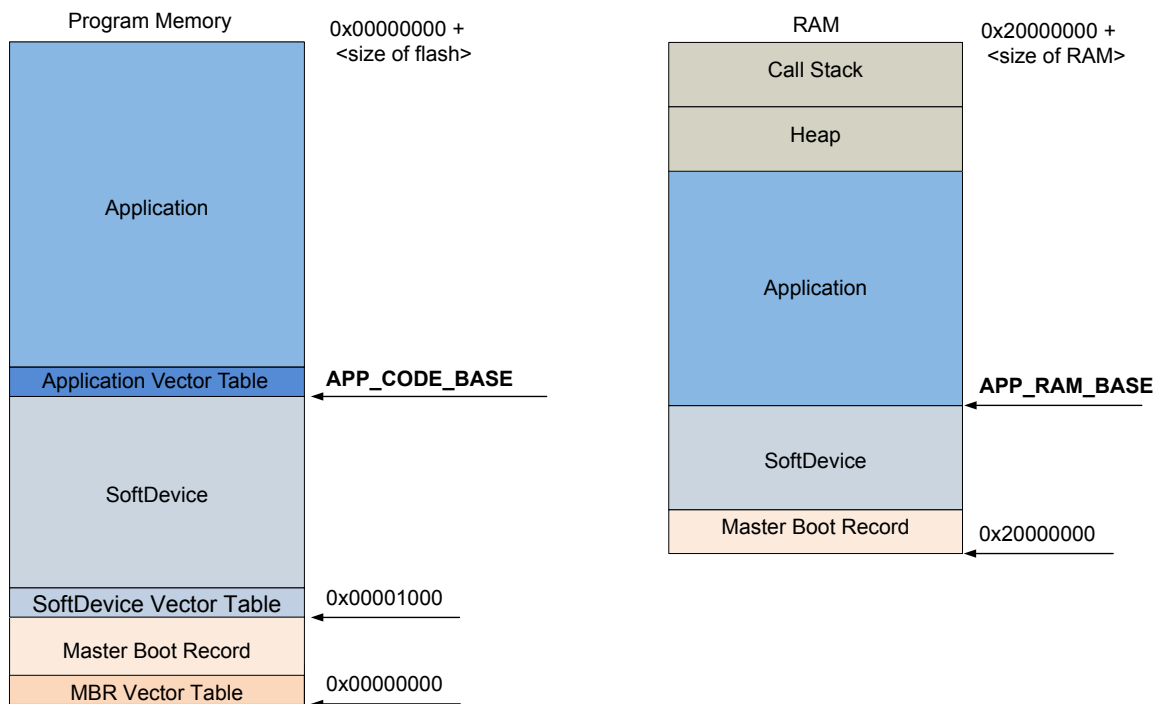
*Figure 19: Memory resource map*

## 14.1.1 Memory resource requirements

This section describes the memory resource requirements for an enabled and disabled S122 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 19: 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[7], to the `SD_FLASH_SIZE` defined in `nrf_sdm.h`.

### RAM

| RAM | S122 Enabled | S122 Disabled |
|---|---|---|
| SoftDevice RAM consumption | Minimum required RAM [8] + Configurable Resources | 8 bytes |
| APP_RAM_BASE address (minimum required value) | 0x20000000 + SoftDevice RAM consumption | 0x20000008 |

*Table 24: S122 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*.

---

[7]  1 kB = 1024 bytes

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

NORDIC
SEMICONDUCTOR

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 S122 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 25: S122 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 67. 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 | S122 Enabled | S122 Disabled |
|---|---|---|
| Maximum usage with FPU disabled | 1536 bytes (0x600) | 0 bytes |

*Table 25: S122 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

NORDIC
SEMICONDUCTOR

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 Security configuration

The SoftDevice allows the number of *Security Manager Protocol (SMP)* instances available for all connections operating in central role to be specified by the application.

At least one *SMP* instance is needed in order to carry out *SMP* operations for central role connections, and a *SMP* instance can be shared amongst multiple central role connections. A larger number of *SMP* instances will allow multiple connections to have ongoing concurrent *SMP* operations, but this will result in increased RAM usage by the SoftDevice. The number of *SMP* instances is specified through `sd_ble_cfg_set`.

## 14.5 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 S122 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 Central 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 Observer or a Scanner. In addition to these Bluetooth Low Energy roles, Flash memory API, *QoS* channel survey, and Radio Timeslot API can also run simultaneously.

Central link timing-events are added relative to already running central link timing-events. Flash access timing-event and Radio Timeslot timing-event are scheduled independently and may occur at the same time and collide. *QoS* channel survey timing-events are scheduled the same way as the scanner, which means relative to existing central link timing-events, but the *QoS* channel survey timing-events always have the lowest priority. If central links, scanner, and the *QoS* channel survey are scheduled with a common base interval and the sum of their timing-event lengths is less than this interval, they never block each other. If a channel survey timing-event is blocked by a higher priority event, the timing-event is skipped and a new one is requested one interval later.

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 | • Central connections that are about to time out |
| Second priority | • Central connection setup (waiting for ack from peer)<br>• Initiator<br>• Scanner 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<br>• Flash access after it has been blocked consecutively for a few times<br>• Radio Timeslot with high priority |
| Fourth priority | • Flash access<br>• Radio Timeslot with normal priority |
| Last priority | • *QoS* channel survey |

*Table 26: Scheduling priorities*

## 15.2 Initiator timing

This section introduces the different situations that happen with the Initiator when establishing a connection.

When establishing a connection with no other connections active, the Initiator will establish the connection in the minimum time and allocate the first central link connection event transmitWindowDelay after the connect request was sent, as shown in the following figure.
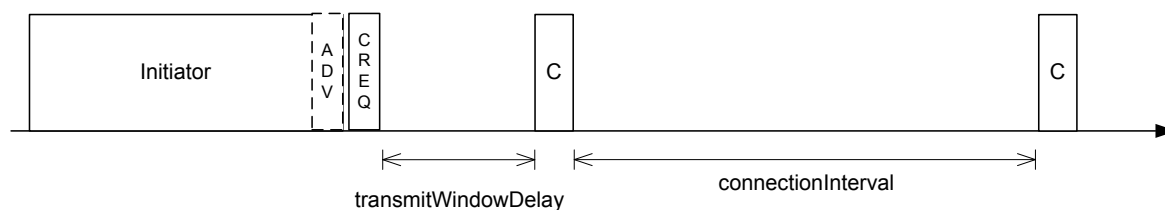


*Figure 20: Initiator - first connection*

When establishing a new connection with other connections already made as a Central, the Initiator will start synchronously to the connected link timing-events and schedule the new central connection's first timing-event in any free time between existing central timing-events or after the existing central timing-events. Central link timing-events will be scheduled close to each other (without any free time between them). The minimum time between the start of two central role timing-events is the event length of the central role to which the first timing-event belongs. This minimum time is referred to as $t_{event}$. The following figure illustrates the case of establishing a new central connection with one central connection already running.
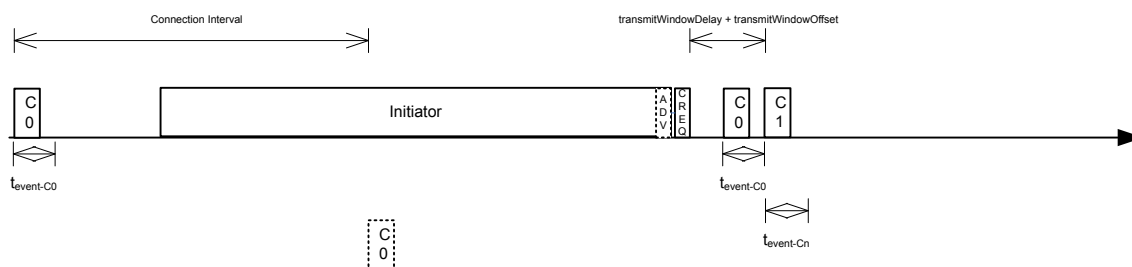
NORDIC
SEMICONDUCTOR

*Figure 21: Initiator - one central connection running*

When a central link disconnects, the timings of other central link timing-events remain unchanged. The following figure illustrates the case when central link C1 is disconnected, which results in free time between C0 and C2.
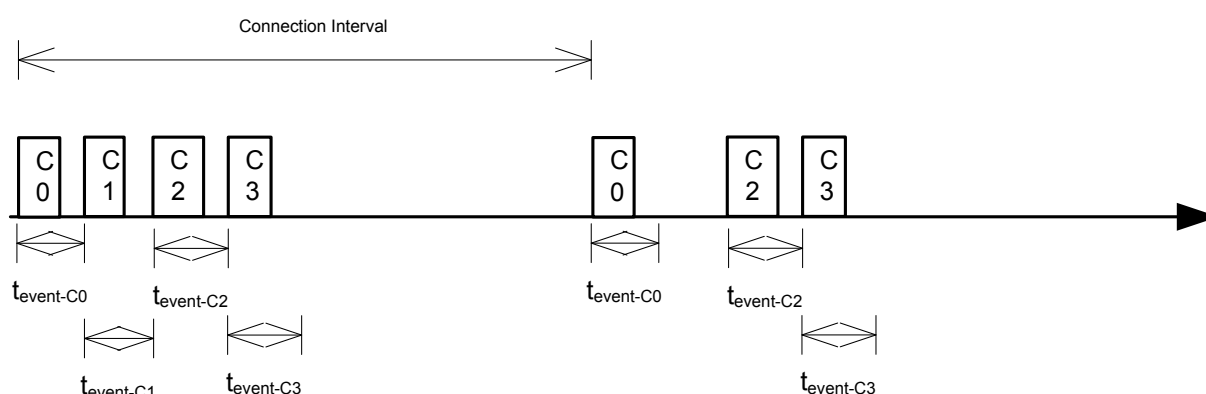


*Figure 22: Initiator - free time due to disconnection*

When establishing a new connection in cases where free time is available between already running central link timing-events, best-fit algorithm is used to find which free time space should be used. Figure 23: Initiator - one or more connections as a Central on page 59 illustrates the case when all existing central connections have the same connection interval and the initiator timing-event starts around the same time as the 1st central connection (C0) timing-event in the schedule. There is available time between C1–C2 and between C2–C3. A timing-event for new a connection, Cn, is scheduled in the available time between C2–C3 because that is the best fit for Cn.
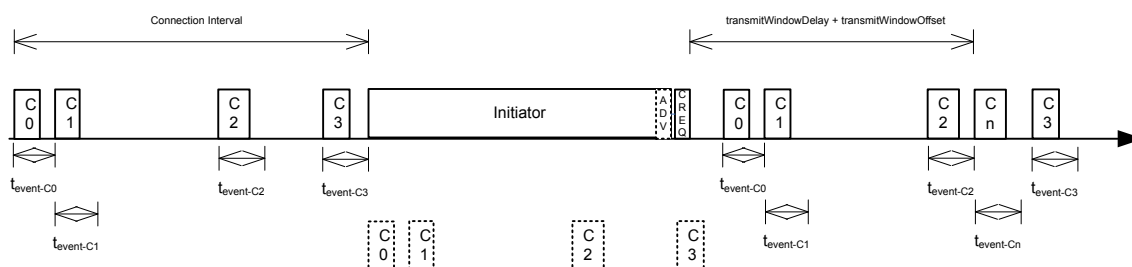


*Figure 23: Initiator - one or more connections as a Central*

Figure 24: Initiator - free time not enough on page 60 illustrates the case when any free time between existing central link timing-events is not long enough to fit the new connection. The new central link timing-event is placed after all running central link timing-events in this case.
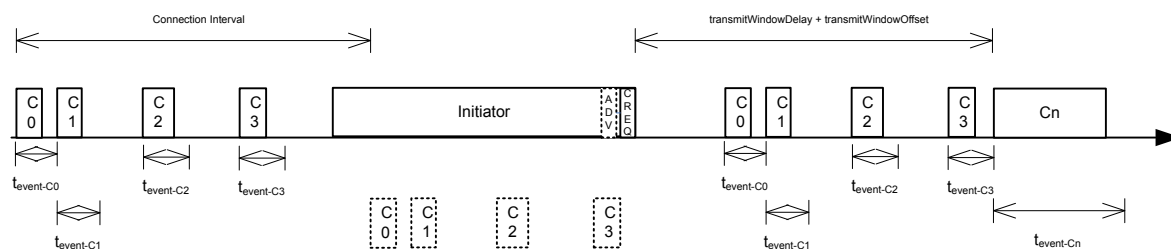
*Figure 24: Initiator - free time not enough*

The Initiator is scheduled relative to existing central connections or the *QoS* channel survey. This is illustrated in Figure 25: Initiator - scheduling and connection setup on page 60. The application requests the Initiator to start at (1). A reference point, $t_r$, is determined from the existing links. From $t_r$, the earliest possible scan window is found by subtracting the scan interval ($T_{SI}$) until the nearest point in the future is found. In the example, the Initiator blocks the second connection (C1) during its second scan window. In that window, it receives a connectable advertisement (2) and responds with a connection indication (3). The Initiator has now become a Central (C3) and its link is placed between C1 and C2 by the best-fit algorithm.
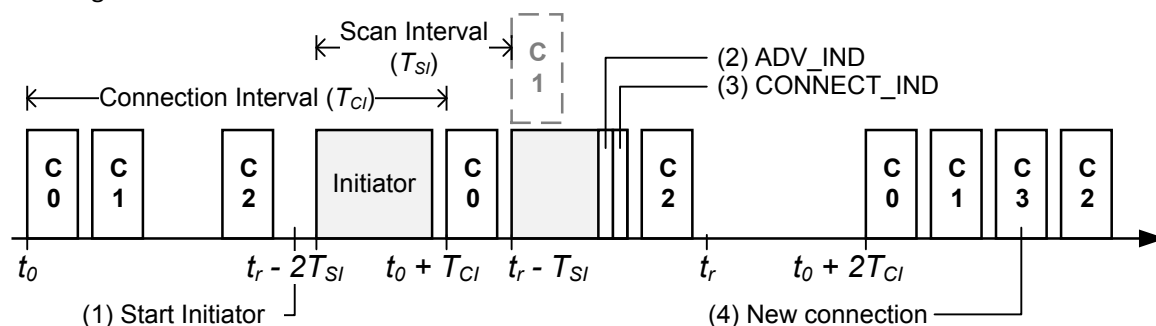


*Figure 25: Initiator - scheduling and connection setup*

# 15.3 Connection timing as a Central

Central link timing-events are added relative to already running central link timing-events.

Central link timing-events are offset from each other by $t_{event}$ depending on the configuration of the connection. For details about $t_{event}$, see Initiator timing on page 58.

Figure 26: Multilink scheduling - one or more connections as a Central, factored intervals on page 60 shows a scenario where two central links are established. C0 timing-events correspond to the first Central connection, and C1 timing-events correspond to the second Central connection. C1 timing-events are initially offset from C0 timing-events by $t_{event-C0}$. In this example, C1 has exactly double the connection interval of C0 (the connection intervals have a common factor which is "connectionInterval 0"), so the timing-events remain forever offset by $t_{event-C0}$.
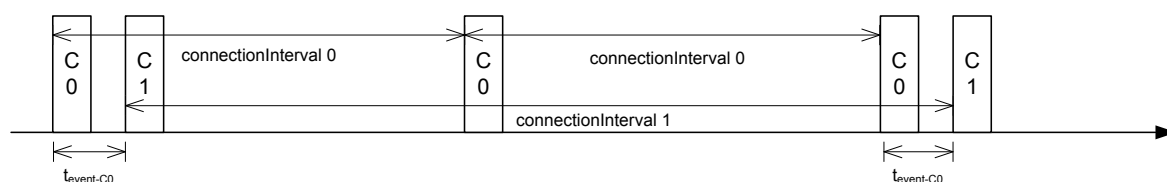


*Figure 26: Multilink scheduling - one or more connections as a Central, factored intervals*

In Figure 27: Multilink scheduling - one or more connections as a Central, unfactored intervals on page 61 the connection intervals do not have a common factor. This connection parameter configuration is possible, though this will result in dropped packets when events overlap. In this scenario, the second timing-event shown for C1 is dropped because it collides with the C0 timing-event.
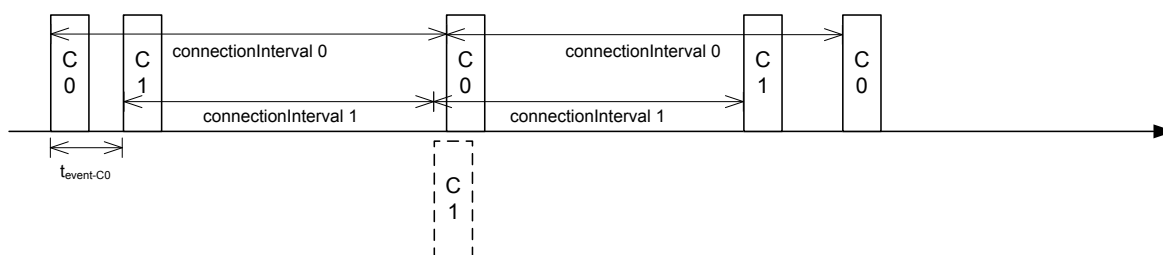


*Figure 27: Multilink scheduling - one or more connections as a Central, unfactored intervals*

Figure 28: Multilink scheduling with eight connections as a Central and minimum interval on page 61 shows eight concurrent links as a Central with an event length of 2.5 ms and a connection interval of 20 ms. In this case, all eight Centrals will have a connection event within the 20 ms interval, and the connection events will be 2.5 ms apart.
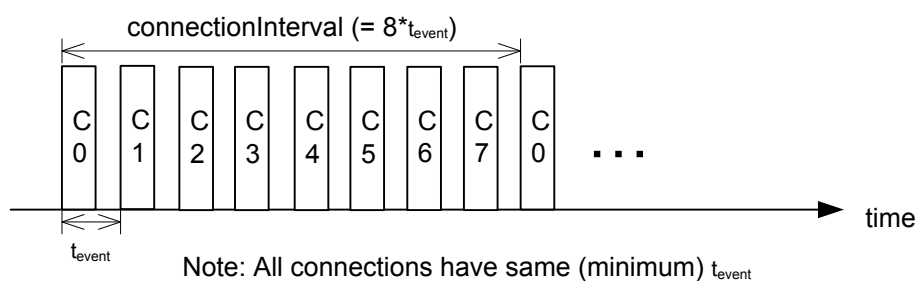


*Figure 28: Multilink scheduling with eight connections as a Central and minimum interval*

Figure 29: Multilink scheduling of connections as a Central and interval > min on page 61 shows a scenario similar to the one illustrated above except the connection interval is longer than 20 ms, and Central 1 and 4 has been disconnected or does not have a timing-event in this time period. It shows the idle time during a connection interval and that the timings of central link timing-events are not affected if other central links disconnect.
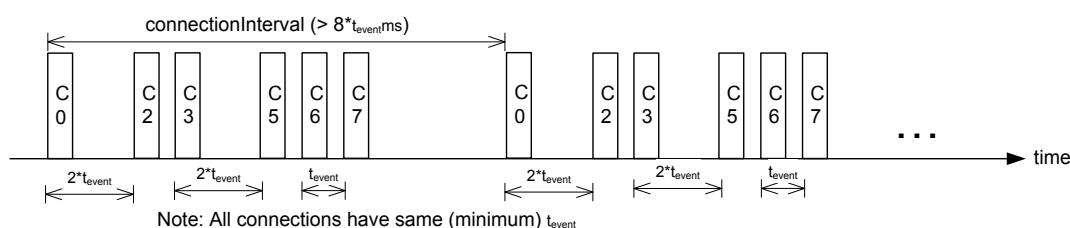


*Figure 29: Multilink scheduling of connections as a Central and interval > min*

## 15.3.1 Advanced Central connection timing

In some advanced use-cases, the Central may require links to be closer together than what is supported on the initiating PHY. This can be achieved by choosing a shorter connection event length and initiating the PHY update procedure to the faster PHY after establishing a connection.

When connection event length checking is disabled, the Central may create connections that have scheduling conflicts on the initiating PHY. This is illustrated in Figure 30: Too short connection event length causes scheduling conflicts on page 62. The central connections C0 and C1 alternate on blocking each other's timing-events. This degrades the average link performance.
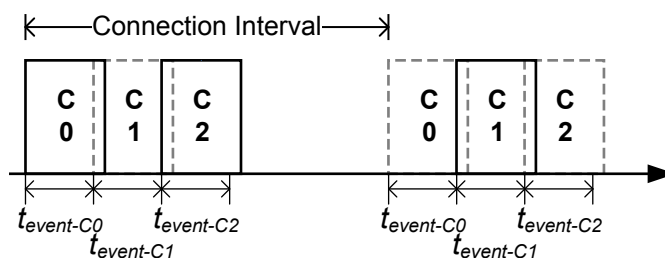
NORDIC
SEMICONDUCTOR

*Figure 30: Too short connection event length causes scheduling conflicts*

After C0 has been updated to a faster PHY, it no longer blocks the timing-events of C1. However, C1 and C2 are still in conflict, and their average link performance is degraded. See Figure 31: No scheduling conflict for first link after PHY update on page 62.
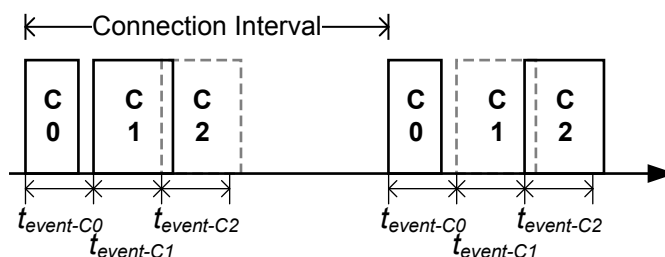


*Figure 31: No scheduling conflict for first link after PHY update*

When all links have been updated to the faster PHY, the Central has achieved the closest packing of links that is possible for a given connection interval. In Figure 32: Optimal packing of four links and a scanner on page 62, all four central links are on the faster PHY, and a scanner is consuming the remainder of the available time.
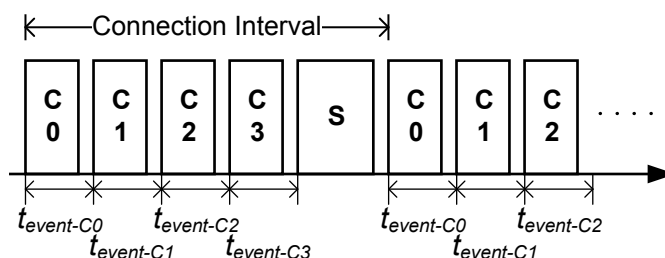


*Figure 32: Optimal packing of four links and a scanner*

# 15.4 Scanner timing

Scanning is a periodic activity where the SoftDevice listens for packets from advertisers. When the SoftDevice starts scanning, it will listen for packets on the primary advertising channels. This SoftDevice does not support scanning on a secondary advertising channel.

## 15.4.1 Primary channel scanner timing

The following figure shows that when scanning for advertisers with no active connections, the scan interval and window can be any value within the *Bluetooth Core Specification*.
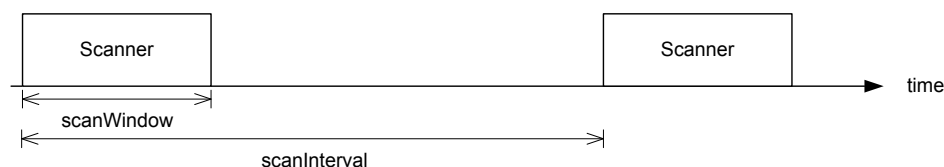
*Figure 33: Scanner timing - no active connections*

A primary channel scanner timing-event is always placed after the central link timing-events. Figure 34: Scanner timing - one or more connections as a Central on page 63 shows that when there are one or more active connections, the scanner or observer role timing-event will be placed after the link timing-events. With scanInterval equal to the connectionInterval and a *scanWindow* $\leq$ $(connectionInterval - \sum t_{event})$, scanning will proceed without overlapping with central link timing-events.
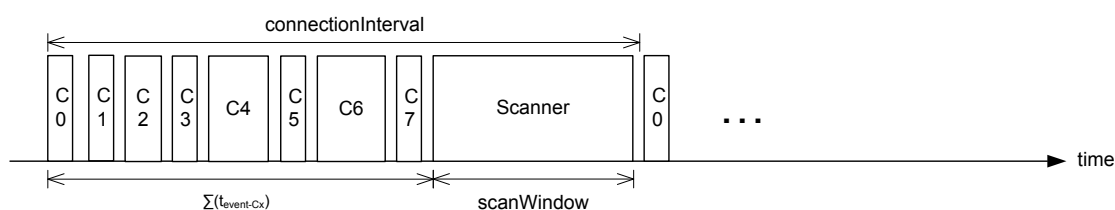


*Figure 34: Scanner timing - one or more connections as a Central*

The following figure shows a scenario where free time is available between link timing-events, but still the scanner timing-event is placed after all connections.
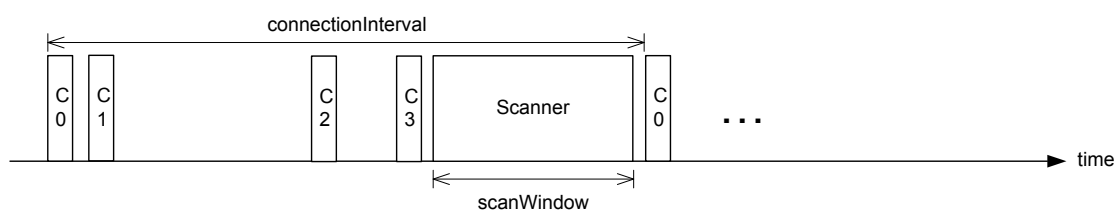


*Figure 35: Scanner timing - always after connections*

The following figure shows a Scanner with a long scanWindow which will cause some connection timing-events to be dropped.
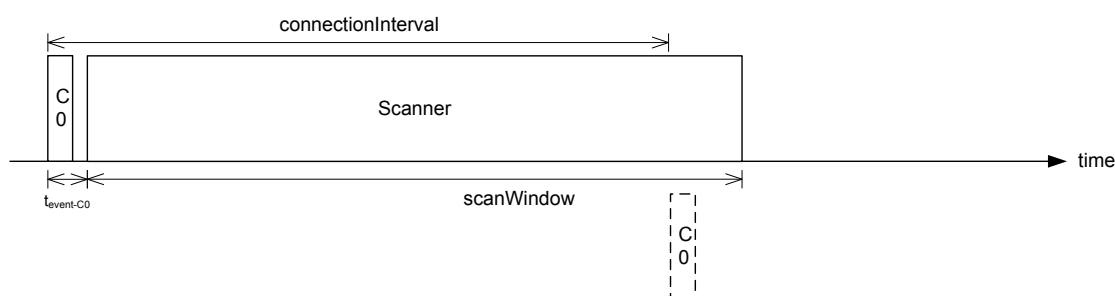


*Figure 36: Scanner timing - one connection, long window*

# 15.5 Connection timing with Connection Event Length Extension

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

NORDIC
SEMICONDUCTOR

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 65.

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 37: Multilink scheduling and connection event length extension on page 64. Here C1 can utilize the free time left by a previously disconnected link C2, C3 has idle time as the last central link, and C0 is benefitting from having a connection interval set to half of that of C1 and C3.
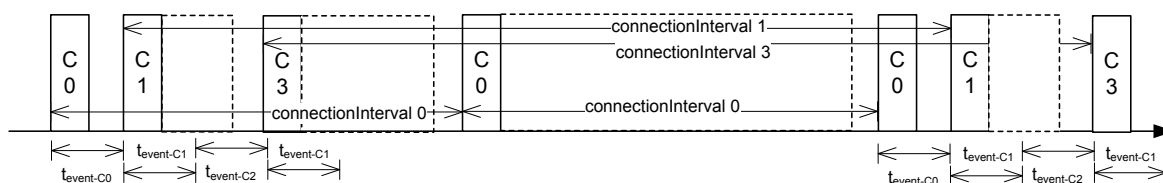


*Figure 37: Multilink scheduling and connection event length extension*

# 15.6 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 26: 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.

# 15.7 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 26: 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

NORDIC
SEMICONDUCTOR

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.

# 15.8 Suggested intervals and windows

This section provides recommendations for choosing intervals and windows for connection and scanning on LE 1M PHY unless specified otherwise.

The time required to fit one timing-event of all active central links is equal to the sum of $t_{event}$ of all active central links. Therefore, 20 link timing-events can complete in $\sum t_{event-Cx}$, which is 50 ms for connections with a 2.5 ms event length.

This does not leave sufficient time in the schedule for scanning or initiating new connections (when the number of connections already established is less than the maximum). Scanner, observer, and initiator events can therefore cause connection packets to be dropped.

It is recommended that all connections have intervals that have a common factor. This common factor should be greater than or equal to $\sum t_{event-Cx}$. For example, for eight connections with an event length of 2.5 ms, the lowest recommended connection interval is 20 ms. This means all connections would then have a connection interval of 20 ms or a multiple of 20 ms, such as 40 ms, 60 ms, and so on.

If short connection intervals are not essential to the application and there is a need to have a Scanner running at the same time as connections, then it is possible to avoid dropping packets on any connection as a Central by having a connection interval larger than $\sum t_{event-Cx}$ + *scanWindow*. The Initiator is scheduled asynchronously to any other role (and any other timing-activity), hence the initiator timing-event might collide with other timing-events even if the above recommendation is followed.

For example, setting the connection interval to 43.75 ms will allow three connection events with event length of 3.75 ms and a scan window of 31.0 ms, which is sufficient to ensure advertising packets from a 20 ms (nominal) advertiser hitting and being responded to within the window.

To summarize, a recommended configuration for operation without dropped packets for cases of only central roles running is that all central role intervals (i.e. connection interval, scanner/observer/initiator intervals) should have a common factor. This common factor should be greater than or equal to $\sum t_{event-Cx}$ + *scanWindow*.

The probability of collision can be reduced (though not eliminated) if the central role link parameters are set as suggested in this section, and the following rules are applied for all roles:

- Interval of all roles have a common factor which is greater than or equal to $\sum t_{event-Cx}$ + *ScanWindow*

If only Bluetooth Low Energy role events are running and the above conditions are met, the worst-case collision scenario will be Initiator and one or more connections as Central colliding in time. The number of colliding connections as Central depends on the maximum timing-event length of other asynchronous timing-activities. For example, there will be two Central connection collisions if all connections have the same bandwidth and both the initiator scan window is approximately equal to the $t_{event}$ of the Central connections. The following figure shows this case of collision.

NORDIC
SEMICONDUCTOR
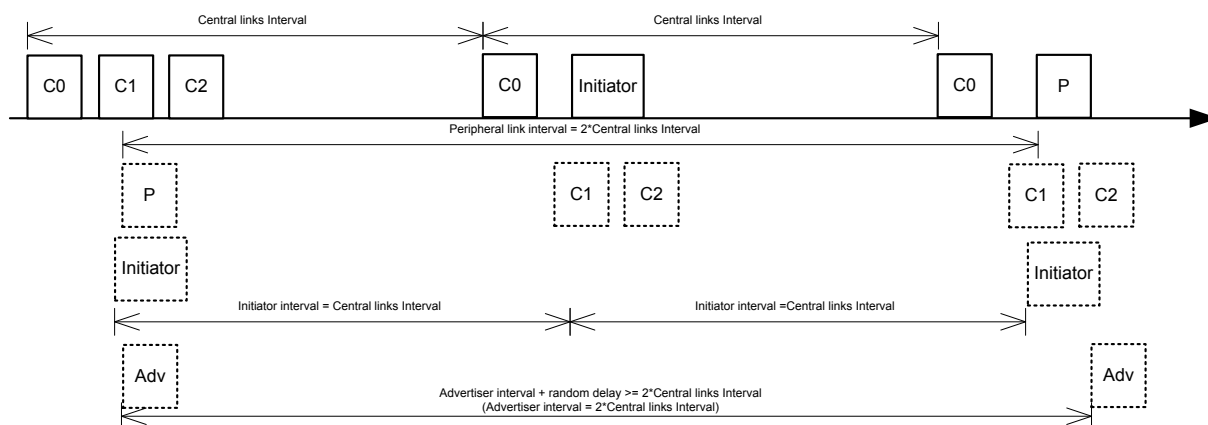
*Figure 38: Worst-case collision of Bluetooth Low Energy roles*

**Note:** The S122 SoftDevice does not support the Peripheral or Broadcaster roles.

These collisions will result in collision resolution through the priority mechanism (refer to Table 26: Scheduling priorities on page 58).

Collision resolution may cause bad performance if suboptimal intervals are chosen. For example, a scanner that is configured with a scan interval of 2000 ms and a scan window of 1000 ms will collide with a connection with a connection interval of 50 ms. In this case, the connection that schedules events often compared to the scanner will raise its priority and may cause the scanner to receive less radio time than expected.

**Note:** These are worst-case collision numbers, and an average case will most likely be better.

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 will also add up to the worst-case collision scenario.

Dropped packets are possible due to collision between different roles as explained above. Application should tolerate dropped packets by setting the supervision time-out for connections long enough to avoid loss of connection when packets are dropped.

# 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 19: 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 70.

NORDIC
SEMICONDUCTOR

| 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 27: 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 reserved for future use on the S122 SoftDevice.
- 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.
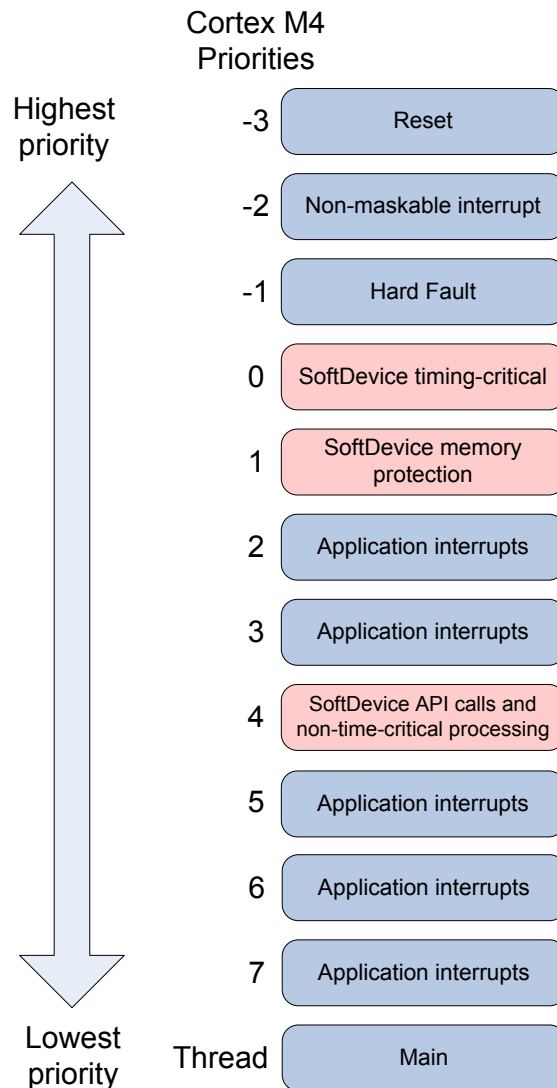
*Figure 39: Exception model*

**Note:** Priority level 1 is reserved for future use on the S122 SoftDevice.

As seen from Figure 39: Exception model on page 69, 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 can 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.
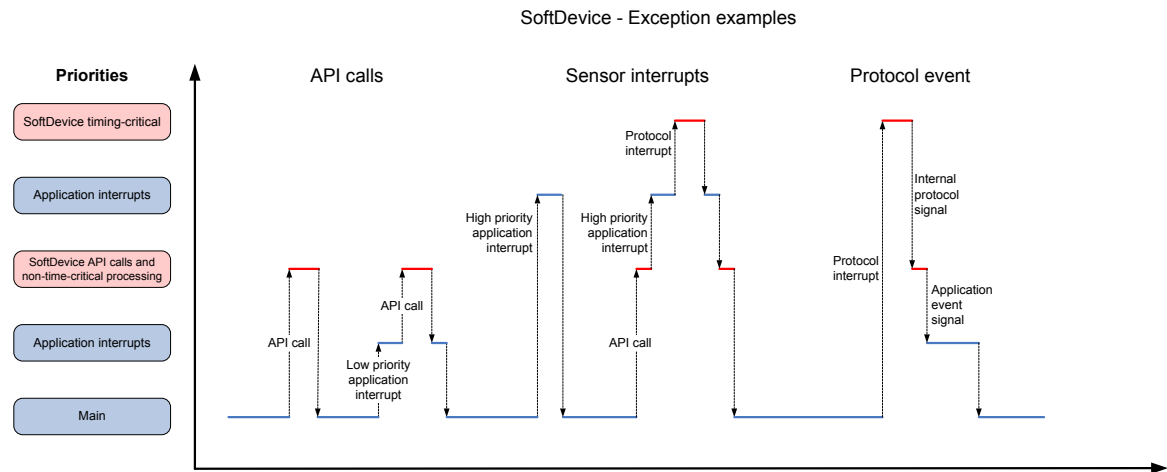
*Figure 40: SoftDevice exception examples*

# 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.
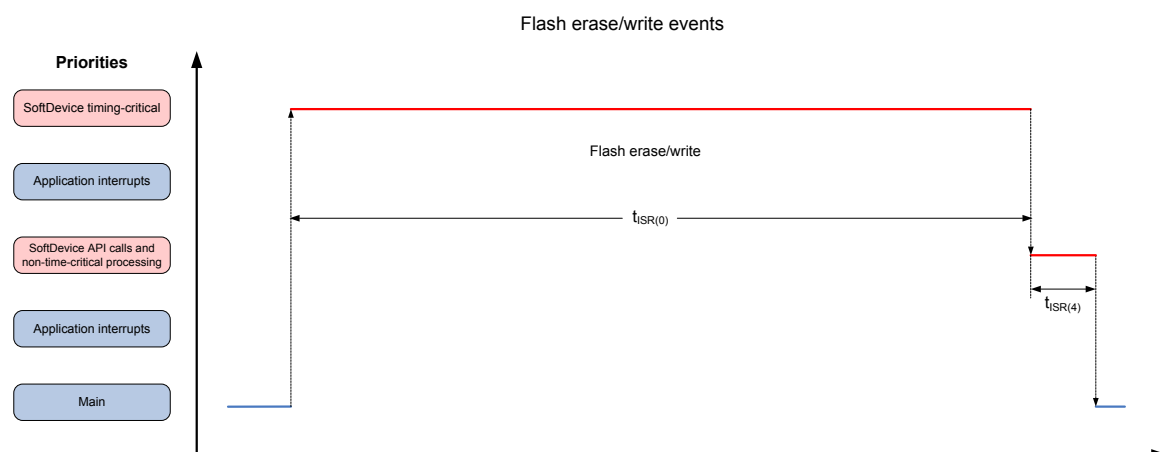


*Figure 41: Flash API activity (some priority levels left out for clarity)*

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.

NORDIC
SEMICONDUCTOR

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 28: 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.

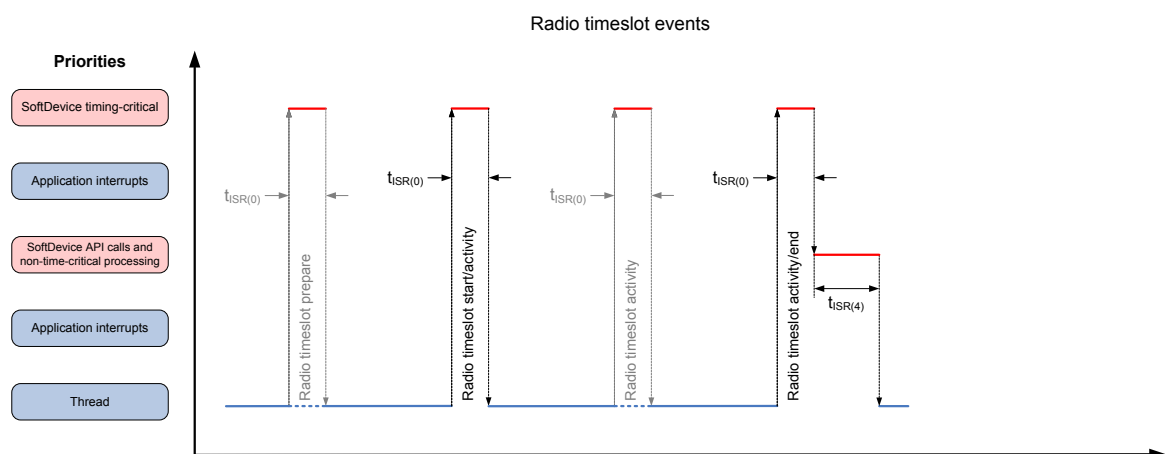See Radio Timeslot API on page 26 for more information on the Radio Timeslot API.



*Figure 42: 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:

NORDIC
SEMICONDUCTOR

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 29: 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 scanner and initiator processor usage

This section describes the processor availability and interrupt processing time for the SoftDevice when the scanner or initiator role is running.



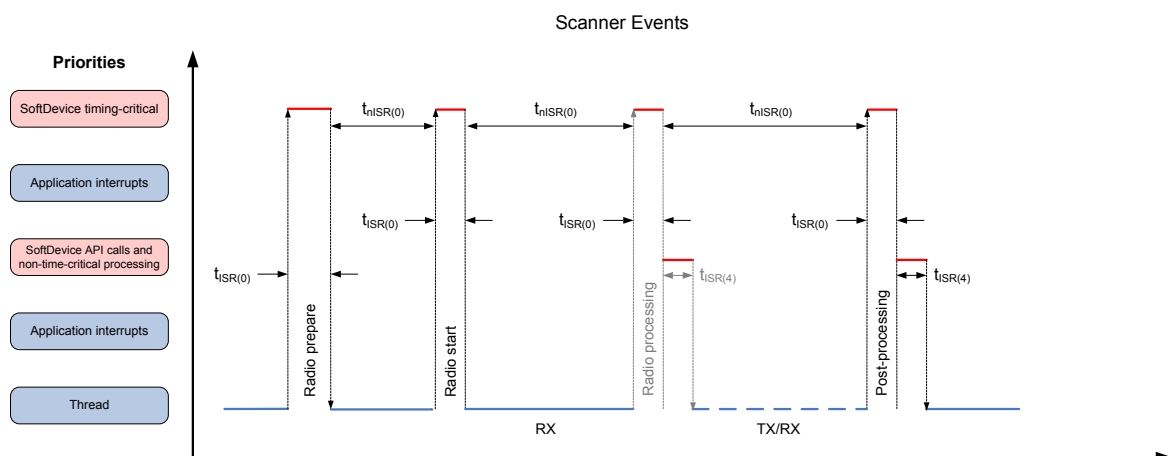*Figure 43: Scanning or initiating (some priority levels left out for clarity)*

When scanning or initiating, the pattern of SoftDevice processing activity at interrupt priority level 0 is as follows:

1. An interrupt (Radio prepare) sets up and prepares the software and hardware for this scanner or initiator event.
2. A short interrupt occurs when the Radio starts listening for advertisement packets.

3. During scanning, there will be zero or more instances of radio processing, depending upon whether the active role is a Scanner or an Initiator, whether scanning is passive or active, whether advertising packets are received or not, and upon the type of the received advertising packets. Such radio processing may be followed by the SoftDevice processing at interrupt priority level 4.

4. When the event ends (either by timeout, or if the Initiator receives a connectable advertisement packet it accepts), the SoftDevice does some Post processing, which may be followed by processing at interrupt priority level 4.

SoftDevice processing activity in the different priority levels when scanning or initiating is outlined in Table 30: Processor usage for scanning or initiating on page 73. The typical case is seen when scanning or initiating without using a whitelist and without sending scan or connect requests. The max case can be seen when scanning or initiating with a full whitelist, using private addresses, sending scan or connect requests while having a maximum number of connections, and utilizing the Radio Timeslot API, Flash memory API, using PA/LNA, and *QoS* channel survey module at the same time.

| Parameter | Description | Min | Typical | Max |
|---|---|---|---|---|
| $t_{ISR(0),RadioPrepare}$ | Processing when preparing the radio for scanning or initiating | | 67 µs | 68 µs |
| $t_{ISR(0),RadioStart}$ | Processing when starting the scan or initiation | | 23 µs | 25 µs |
| $t_{ISR(0),RadioProcessing}$ | Processing after sending/receiving packet | | 95 µs | 116 µs |
| $t_{ISR(0),PostProcessing}$ | Processing at the end of a scanner or initiator event | | 120 µs | 199 µs |
| $t_{nISR(0)}$ | Distance between interrupts during scanning | 80 µs | >1.5 ms | |
| $t_{ISR(4)}$ | Priority level 4 interrupt at the end of a scanner or initiator event | | 27 µs | |

*Table 30: Processor usage for scanning or initiating*

From the table we can calculate a typical processing time for one scan event receiving one advertisement packet to be:

$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + t_{ISR(4)} =$ `332 µs`

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

## 16.3.3.2 Bluetooth Low Energy central connection processor usage

This section describes the processor availability and interrupt processing time for the SoftDevice in a central connection event.
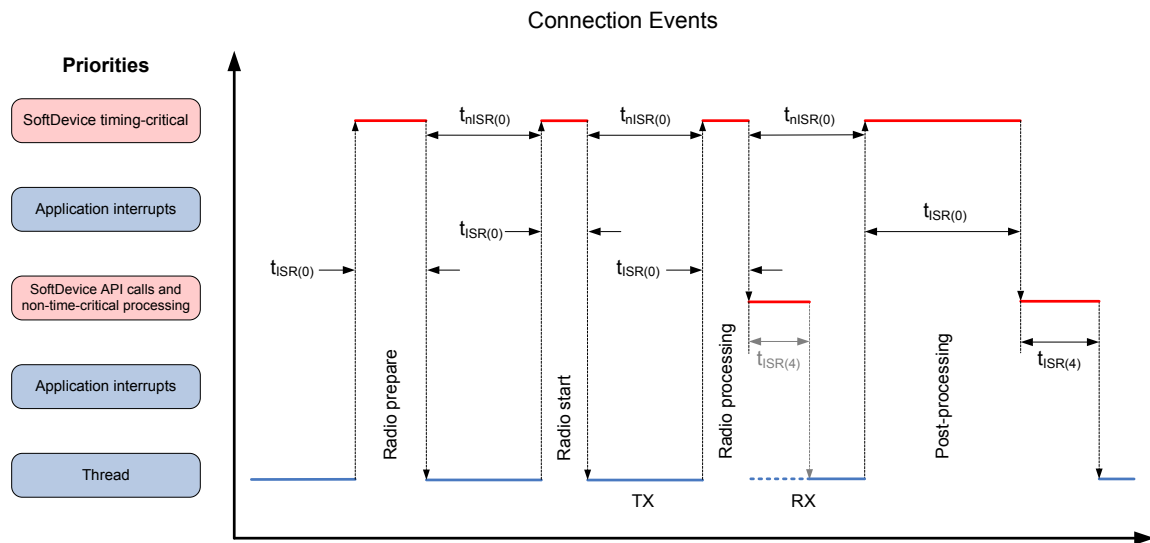
*Figure 44: Central connection events (some priority levels left out for clarity)*

In a central 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.
2. A short interrupt occurs when the Radio starts transmitting the first packet in the connection event.
3. When the transmission is complete, there is a radio processing interrupt that switches the Radio to reception.
4. When the reception is complete, there is a radio processing interrupt that processes the received packet and either switches the Radio back to transmission (and possibly a new reception after that), or the event ends with post processing.
5. After the priority level 0 processing, the SoftDevice processes any received data packets, executes any *GATT*, *ATT*, or *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 in the different priority levels during central connection events is outlined in Table 31: Processor usage latency when connected on page 75. The typical case is seen when receiving *GATT* write commands writing 20 bytes. The max case can be seen when sending and receiving maximum length packets and initiating encryption, while having a maximum number of connections and utilizing the Radio Timeslot API, Flash memory API, using PA/LNA, and *QoS* channel survey module at the same time.

| Parameter | Description | Min | Typical | Max |
|---|---|---|---|---|
| $t_{ISR(0),RadioPrepare}$ | Processing when preparing the radio for a connection event | | 83 µs | 84 µs |
| $t_{ISR(0),RadioStart}$ | Processing when starting the connection event | | 18 µs | 18 µs |
| $t_{ISR(0),RadioProcessing}$ | Processing after sending or receiving a packet | | 98 µs | 102 µs |
| $t_{ISR(0),PostProcessing}$ | Processing at the end of a connection event | | 97 µs | 207 µs |
| $t_{nISR(0)}$ | Distance between connection event interrupts | 183 µs | > 200 µs | |
| $t_{ISR(4)}$ | Priority level 4 interrupt after a packet is sent or received | | 40 µs | |

*Table 31: Processor usage latency when connected*

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

$$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + 2 * t_{ISR(4)} = \text{376 µs}$$

This means typically more than 99% of the processor time is available to the application when one central 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 role 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 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 role
- the sum of the max time of the first interrupt of the Bluetooth Low Energy role and the last interrupt of the Radio Timeslot API

NORDIC
SEMICONDUCTOR

# 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 63 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.

The following table shows maximum data throughput at a connection interval of 7.5 ms for a single central 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 | 277.3 kbps |
| | | | Send Write command | 192.0 kbps | 277.3 kbps |
| | | | Send Write request | 10.6 kbps | 10.6 kbps |
| | | | Simultaneous receive Notification and send Write command | 149.3 kbps (each direction) | 213.3 kbps (each direction) |
| GATT Server | 23 | 7.5 ms | Send Notification | 192.0 kbps | 277.3 kbps |
| | | | Receive Write command | 192.0 kbps | 277.3 kbps |
| | | | Receive Write request | 10.6 kbps | 10.6 kbps |

NORDIC
SEMICONDUCTOR

| 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 | 149.3 kbps (each direction) | 213.3 kbps (each direction) |
| GATT Server | 158 | 7.5 ms | Send Notification | 248.0 kbps | 358.2 kbps |
| | | | Receive Write command | 248.0 kbps | 358.2 kbps |
| | | | Receive Write request | 82.6 kbps | 82.6 kbps |
| | | | Simultaneous send Notification and receive Write command | 192.9 kbps (each direction) | 275.5 kbps (each direction) |
| GATT Client | 23 | 3.75 ms | Receive Notification | 85.3 kbps | 106.6 kbps |
| | | | Send Write command | 85.3 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) | 106.6 kbps (each direction) |
| GATT Server | 23 | 3.75 ms | Send Notification | 85.3 kbps | 106.6 kbps |
| | | | Receive Write command | 85.3 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) | 106.6 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 | 42.6 kbps (each direction) | 64.0 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 | 42.6 kbps (each direction) | 64.0 kbps (each direction) |

*Table 32: Data throughput for a single connection with 23 byte ATT MTU*

NORDIC
SEMICONDUCTOR

A connection interval of 20 ms and an event length of 2.5 ms allows up to eight connections. The maximum throughput per connection for this case, using 23 byte ATT MTU, is shown in Table 33: Data throughput for up to 8 connections on page 78.

For connections with longer event length, a longer connection interval would need to be used for each connection to prevent connection events from overlapping. See Scheduling on page 57 for more information on how connections can be configured.

| Protocol | Event length | Method | Maximum data throughput (LE 1M PHY) |
|---|---|---|---|
| GATT Client | 2.5 ms | Receive Notification | 16.0 kbps |
| | | Send Write command | 16.0 kbps |
| | | Send Write request | 4.0 kbps |
| | | Simultaneous receive Notification and send Write command | 8.0 kbps (each direction) |
| GATT Server | 2.5 ms | Send Notification | 16.0 kbps |
| | | Receive Write command | 16.0 kbps |
| | | Receive Write request | 4.0 kbps |
| | | Simultaneous send Notification and receive Write command | 8.0 kbps (each direction) |

Table 33: Data throughput for up to 8 connections

NORDIC
SEMICONDUCTOR

# 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 Scanning event

This section gives an overview of the power profile of the scanning event implemented in the SoftDevice. Figure 45: Scanning event on page 79 shows the current profile for a scan event. When the SoftDevice receives a packet, it transitions from stage E to stage F. For each advertising packet the SoftDevice receives, the sequence D-F is added to the current profile.
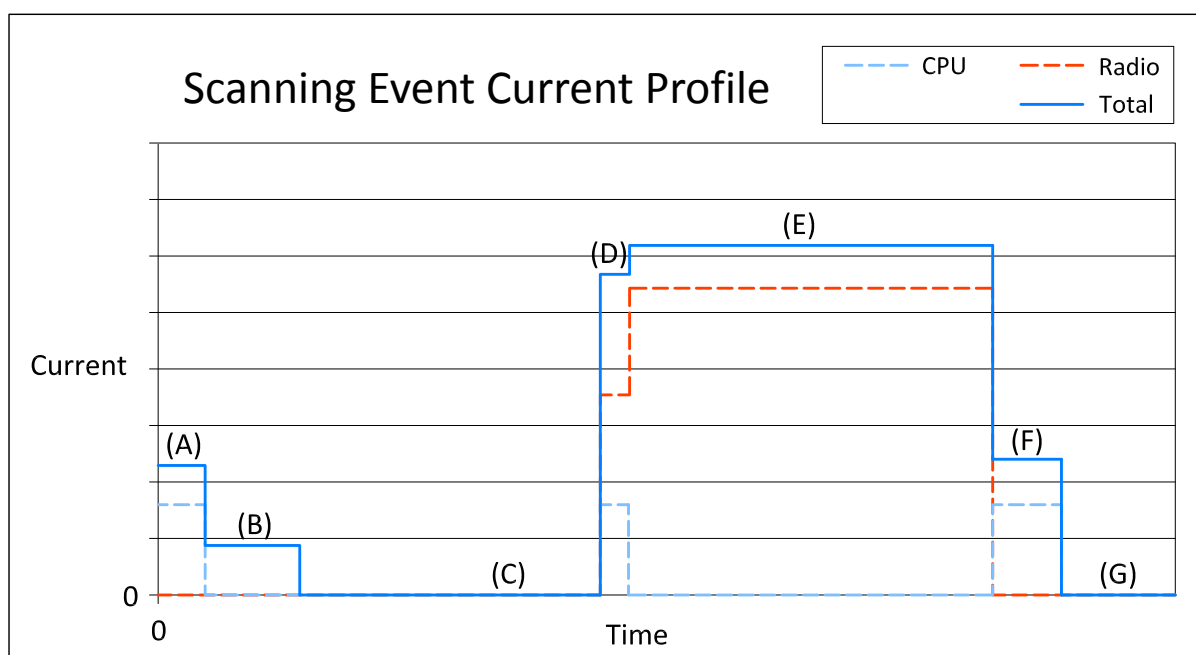


*Figure 45: Scanning event*

| Stage | Description |
|-------|-------------|
| (A) | Pre-processing (CPU) |
| (B) | Standby IDLE + HFXO ramp |
| (C) | Standby |
| (D) | Radio startup |
| (E) | Radio RX |
| (F) | Post-processing (CPU) |
| (G) | Standby |

*Table 34: Scanning event*

## 18.2 Central connection event

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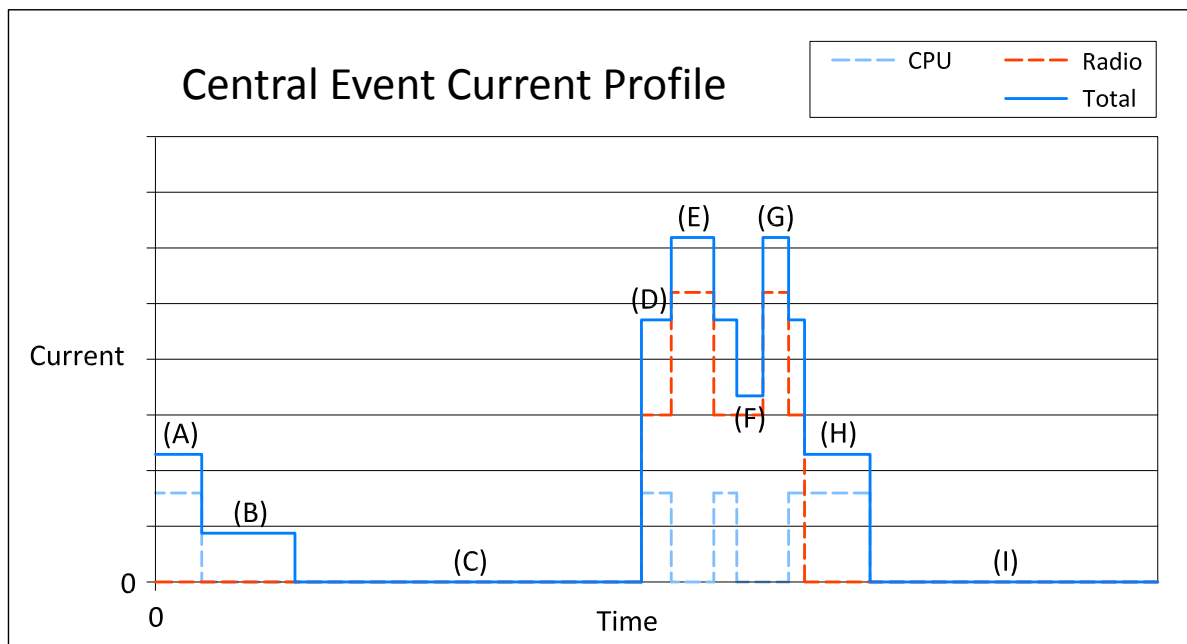


*Figure 46: Central connection event*

| 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 35: Central 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 37: SoftDevice revision examples on page 82.

| 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 36: 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 37: SoftDevice revision examples*

NORDIC
SEMICONDUCTOR

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

*Table 38: 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.

NORDIC
SEMICONDUCTOR

# 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 5.2, 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 5.2, 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 5.2, 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 5.2, 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)**

NORDIC
SEMICONDUCTOR

"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 5.2, 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.

**Quality of Service (QoS)**

A module in the SoftDevice that is dedicated to providing various signal quality measurements. The application can use this data to estimate the quality of received signal.

**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)**

NORDIC
SEMICONDUCTOR

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.

NORDIC
SEMICONDUCTOR

# 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

**QoS**
Quality of Service

**RC**
Resistor-Capacitor

**SDK**
Software Development Kit

**SDM**
SoftDevice Manager

**SM**
Security Manager

**SMP**
Security Manager Protocol

**SoC**
System on Chip

NORDIC
SEMICONDUCTOR

**SVC**
Supervisor Call

**UUID**
Universally Unique Identifier

NORDIC
SEMICONDUCTOR

# 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

© 2020 Nordic Semiconductor ASA. All rights are reserved. Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.

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

NORDIC
SEMICONDUCTOR