SoftDevice Specification

S130 SoftDevice v2.0

2016-04-08

Contents

Chapter 1
S130 SoftDevice

The S130 SoftDevice is a *Bluetooth®* low energy (BLE) Central and Peripheral protocol stack solution. It supports up to eight connections with an additional Observer and a Broadcaster role all running concurrently. The S130 SoftDevice integrates a BLE Controller and Host, and provides a full and flexible API for building *Bluetooth®* Smart nRF51 System on Chip (SoC) solutions.

Key features	Applications

• *Bluetooth®* 4.2 compliant low energy single-mode protocol stack suitable for *Bluetooth®* Smart products
  • Concurrent Central, Observer, Peripheral, and Broadcaster roles with up to eight concurrent connections along with one observer and one broadcaster
  • Configurable number of connections and bandwidth per connection to optimize memory and performance
  • Configurable attribute table size
  • Custom UUID support
  • Link layer
  • L2CAP, ATT, and SM protocols
  • LE Secure Connections pairing model
  • GATT and GAP APIs
  • GATT Client and Server
• Complementary nRF5 SDK including *Bluetooth®* profiles and example applications
• Master Boot Record for over-the-air device firmware update
  • SoftDevice, application, and bootloader can be updated separately
• Memory isolation between the application and the protocol stack for robustness and security
• Thread-safe supervisor-call based API
• Asynchronous, event-driven behavior
• No RTOS dependency
  • Any RTOS can be used
• No link-time dependencies
  • Standard ARM® Cortex®-M0 project configuration for application development
• Support for concurrent and non-concurrent multiprotocol operation
  • Concurrent with the *Bluetooth®* stack using Radio Timeslot API
  • Alternate protocol stack in application space
	• Sports and fitness devices
	  • Sports watches
	  • Bike computers
	• Personal Area Networks
	  • Health and fitness sensor and monitoring devices
	  • Medical devices
	  • Key fobs and wrist watches
	• Home automation
	• AirFuel wireless charging
	• Remote control toys
	• Computer peripherals and I/O devices
	  • Mice
	  • Keyboards
	  • Multi-touch trackpads
	• Interactive entertainment devices
	  • Remote controls
	  • Gaming controllers

Chapter 2
Revision history

Date	Version	Description

April 2016	2.0	Corresponding to SoftDevice S130 version 2.0.0.
		
		Updated:
		
		• SoftDevice Manager on page 10. Added documentation on (previously in Appendix A):
		  • Clock source,
		  • Power management,
		  • Memory isolation and runtime protection
		• Flash memory API on page 21. Documented changes and new numbers.
		• Profile and service support on page 32. Updated the list of profiles and services currently adopted by the *Bluetooth*® Special Interest Group.
		• Radio Notification on page 39.
		• Master Boot Record on page 47.
		• Scheduling on page 54 (previously Chapter 14: Multilink scheduling).
		• Interrupt model and processor availability on page 65. Updated section to align with the new SoftDevice priority level structure.
		• BLE data throughput on page 76. Updated documentation with new numbers.
		
		Added:
		
		• Application Programming Interface (API) on page 9 (previously in Appendix A). Now also including section Error handling on page 9.
		• BLE role configuration on page 38.
		• SoftDevice memory usage on page 51 (Memory resource map and usage on page 51 and Attribute table size on page 53 previously in Chapter 13: System on Chip resource requirements; Call Stack and Heap information previously in Appendix A can be found under Memory resource requirements on page 52).
		• SoftDevice timing-activities and priorities on page 54.
		• Timeslot API timing on page 62.
		
		Several chapters have been restructured, relocated and revised. Appendix A is removed.

June 2015	1.0	Corresponding to SoftDevice S130 version 1.0.0.
July 2014	0.5	Preliminary release.

Chapter 3
Documentation

Additional recommended reading for developing applications using the SoftDevice on the nRF51 SoC includes the product specification, errata, compatibility matrix, and *Bluetooth*® core specification.

A list of the recommended documentation for the SoftDevice is given in Table 1: S130 SoftDevice core documentation on page 7.

Table 1: S130 SoftDevice core documentation

Documentation	Description
nRF51822 Product Specification	Contains a description of the hardware, peripherals, and electrical specifications specific to the nRF51822 IC.
nRF51822 PAN	Contains information on anomalies related to the nRF51822 IC.
nRF51 Series Compatibility Matrix	Contains information on the compatibility between nRF51 Integrated Circuit (IC) revisions, SoftDevices and SoftDevice Specifications, SDKs, development kits, documentation, and Qualified Design Identifications (QDIDs).
Bluetooth Core Specification	The *Bluetooth*® Core Specification version 4.2, Volumes 1, 3, 4, and 6, describes *Bluetooth*® terminology which is used throughout the SoftDevice Specification.

Chapter 4
Product overview

The S130 SoftDevice is a precompiled and linked binary image implementing a *Bluetooth*® 4.2 low energy protocol stack for the nRF51 Series of SoCs.

See the nRF51 Series Compatibility Matrix for SoftDevice/IC compatibility information.

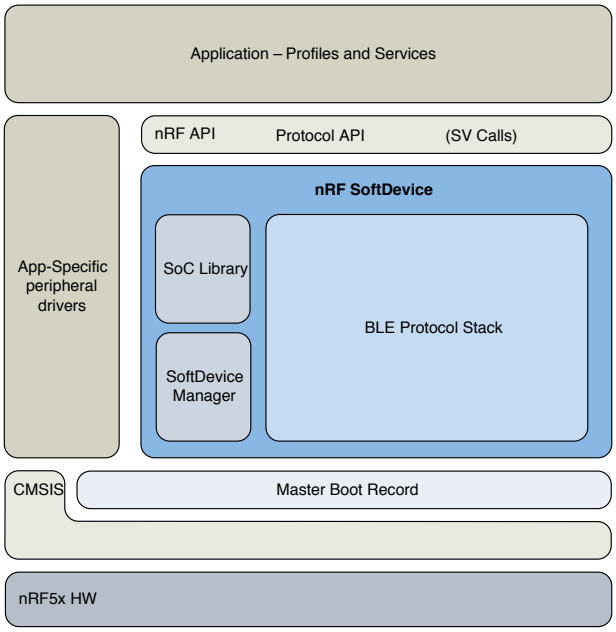

Figure 1: System on Chip application with the SoftDevice

Figure 1: System on Chip application with the SoftDevice on page 8 is a block diagram of the nRF51 series software architecture. It includes the standard ARM® CMSIS interface for nRF51 hardware, a master boot record, 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 API for shared hardware resource management (application coexistence).
- SoftDevice Manager - Implementation and nRF API for SoftDevice management (enabling/disabling the SoftDevice, etc.).
- *Bluetooth*® 4.2 low energy protocol stack - Implementation of protocol stack and API.

The Application Programming Interface (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. See Application Programming Interface (API) on page 9 for more details.

The SoftDevice enables the application developer to develop their code as a standard ARM® Cortex® -M0 project without having the need to integrate with proprietary IC vendor software frameworks. This means that any ARM® Cortex® -M0-compatible toolchain can be used to develop *Bluetooth*® low energy applications with the SoftDevice.

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

Chapter 5
Application Programming Interface (API)

The SoftDevice Application Programming Interface (API) is available to applications as a C programming language interface based on SuperVisor Calls (SVC) and defined in a set of header files.

In addition to a Protocol API enabling wireless applications, there is an nRF API that exposes the functionality of both the SoftDevice Manager and the SoC library.

> **Important:** When the SoftDevice is disabled, only a subset of the SoftDevice APIs is available to the application (see S130 SoftDevice v2.0.0 API). For more information about enabling and disabling the SoftDevice, see SoftDevice enable and disable on page 10.

SVCs are software triggered interrupts conforming to a procedure call standard for parameter passing and return values. Each SoftDevice API call triggers an 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 necessity 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.

> **Important:** 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 66.

5.1 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.

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

5.2 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.

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

Chapter 6
SoftDevice Manager

The SoftDevice Manager (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 low frequency clock (LFCLK) source, see Clock source on page 10.
- the interrupt management, see SoftDevice enable and disable on page 10.
- the embedded protocol stack.

In addition, it may enable the SoftDevice flash, RAM , and peripheral protection. See Memory isolation and runtime protection on page 11.

Detailed documentation of the SDM API is made available with the Software Development Kits (SDK).

6.1 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 51 for more details.

Once the SoftDevice has been enabled, more restrictions apply:

- Some RAM will be reserved. See Memory isolation and runtime protection on page 11 for more details.
- Some peripherals will be reserved. See Hardware peripherals on page 16 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 65 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 65

6.2 Clock source

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

The application must provide the selected clock source and some clock source characteristics, such as accuracy, when it enables the SoftDevice. The SoftDevice Manager is responsible for configuring the low frequency clock 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, clock calibration is required when a temperature change of more than 0.5 degrees has occured to adjust the RC oscillator frequency. The application may choose how often the SoftDevice will make a measurement to detect temperature change, depending on how frequently significant temperature changes are expected to occur in the intended environment of the end product.

6.3 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.

6.4 Memory isolation and runtime protection

The SoftDevice program memory, data memory and peripherals can be sandboxed and runtime protected to prevent the application from interfering with the SoftDevice execution, ensuring robust and predictable performance.

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

Sandboxing is enabled by writing the start address of the application program memory to UICR.CLENR0.

The program memory is divided into two regions at compile time. The SoftDevice Flash Region is located between addresses `0x00000000` and `APP_CODE_BASE - 1` and is occupied by the SoftDevice. The Application Flash Region is located between the addresses `APP_CODE_BASE` and the last valid address in the flash memory and is available to the application. The flash regions are defined when programming the SoftDevice by setting the SoftDevice Flash Region length in the UICR.CLENR0 register.

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.

Figure 2: Memory region designation on page 12 presents an overview of the regions.

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

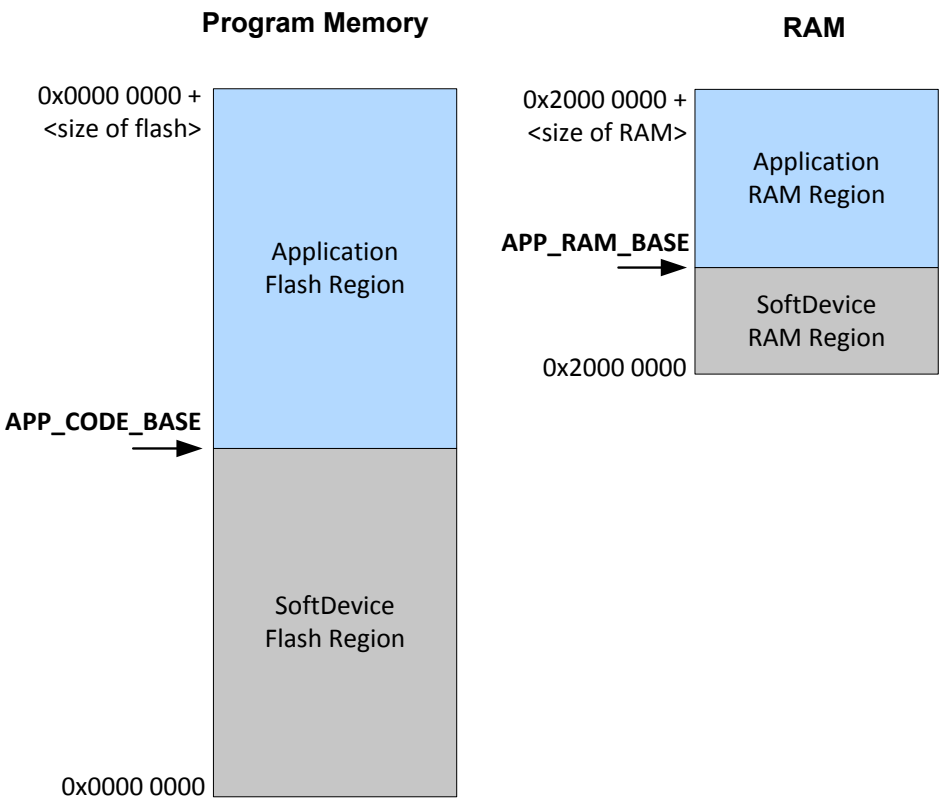

Figure 2: Memory region designation

The SoftDevice uses a fixed amount of flash (program) memory. By contrast, the size of the SoftDevice RAM Region depends on whether the SoftDevice is enabled or not, and on the selected BLE protocol stack configuration. See Role configuration on page 53 for more details.

The amout 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.

Example application program code address range:

`APP_CODE_BASE` ≤ Program ≤ <size of flash>

Example application RAM address range assuming call stack and heap location as shown in Figure 21: Memory resource map on page 51:

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

Sandboxing protects the SoftDevice Flash and RAM Regions. The SoftDevice Flash Region cannot be written or erased at runtime[2]. The SoftDevice RAM Region cannot be written to by an application at runtime. Violation of sandboxing rules, for example an attempt to write to the protected SoftDevice memory, will result in a system Hard Fault as defined by the ARM®Cortex®M0 architecture. There are debugging restrictions applied to these regions which are outlined in the “Memory Protection Unit (MPU)” chapter in the nRF51 Reference Manual that do not affect execution.

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 51 for more details. When the SoftDevice is enabled, RAM up to `APP_RAM_BASE` will be used by the SoftDevice and will be write protected.

[2] The only exception to this is when replacing the SoftDevice using MBR API functions. See Master Boot Record and bootloader on page 47 for details.

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.

Important:

- 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 be 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.

Chapter 7
System on Chip (SoC) library

The coexistence of Application and SoftDevice with safe sharing of common System on Chip (SoC) resources is ensured by the SoC library.

The features described in Table 2: System on Chip features on page 14 are implemented by the SoC library and can be used for accessing the shared hardware resources.

Table 2: System on Chip features

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®. **Important:** 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 while the SoftDevice is enabled:
	• Access to RESETREAS register
	• Set power modes
	• Configure power fail comparator
	• Control RAM block power
	• Use general purpose retention register
	• Configure DC/DC converter state:
	  • DISABLED
	  • ENABLED

Clock	Access to CLOCK block configuration while the SoftDevice is enabled. Allows the HFCLK Crystal Oscillator source to be requested by the application.
Wait for event	Simple power management call for the application to use to enter a sleep or idle state and wait for an application event.
PPI	Configuration interface for PPI channels and groups reserved for an application.

Feature	Description
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 23.
Radio Notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See Radio Notification Signals on page 39.
Block Encrypt (ECB)	Safe use of 128-bit AES encrypt HW accelerator.
Event API	Fetch asynchronous events generated by the SoC library.
Flash memory API	Application access to flash write, erase, and protect. Can be safely used during all protocol stack states. See Flash memory API on page 21.
Temperature	Application access to the temperature sensor.

Chapter 8
System on Chip resource requirements

This section describes how the SoftDevice, including the Master Boot Record (MBR), uses the System on Chip (SoC) resources. The SoftDevice requirements are shown for both when the SoftDevice is enabled and disabled.

The SoftDevice and MBR (see Master Boot Record and bootloader on page 47) 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 amout of RAM required by the SoftDevice see SoftDevice memory usage on page 51.

8.1 Hardware peripherals

SoftDevice access types are used to indicate the availability of hardware peripherals to the application. The availability varies per hardware peripheral and depends on whether the SoftDevice is enabled or disabled.

Table 3: Hardware access type definitions

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

Table 4: Peripheral protection and usage by SoftDevice

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
0	0x40000000	CLOCK	Restricted	Open
0	0x40000000	POWER	Restricted	Open
0	0x40000000	MPU	Restricted	Open
1	0x40001000	RADIO	Blocked[6]	Open
2	0x40002000	UART0	Open	Open
3	0x40003000	SPI0 / TWI0	Open	Open

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
4	0x40004000	SPI1 / SPIS1 / TWI1	Open	Open
...				
6	0x40006000	GPIOTE	Open	Open
7	0x40007000	ADC	Open	Open
8	0x40008000	TIMER0	Blocked[6]	Open
9	0x40009000	TIMER1	Open	Open
10	0x4000A000	TIMER2	Open	Open
11	0x4000B000	RTC0	Blocked	Open
12	0x4000C000	TEMP	Restricted	Open
13	0x4000D000	RNG	Restricted	Open
14	0x4000E000	ECB	Restricted	Open
15	0x4000F000	CCM	Blocked[6]	Open
15	0x4000F000	AAR	Blocked[6]	Open
16	0x40010000	WDT	Open	Open
17	0x40011000	RTC1	Open	Open
18	0x40012000	QDEC	Open	Open
19	0x40013000	LPCOMP	Open	Open
20	0x40014000	SWI0	Open	Open
21	0x40015000	SWI1 / Radio Notification	Restricted[7]	Open
22	0x40016000	SWI2 / SoftDevice Event	Blocked	Open
23	0x40017000	SWI3	Open	Open
24	0x40018000	SWI4	Blocked	Open
25	0x40019000	SWI5	Blocked	Open
...				
30	0x4001E000	NVMC	Restricted	Open
31	0x4001F000	PPI	Open[3]	Open
NA	0x10000000	FICR	Blocked	Blocked
NA	0x10001000	UICR	Restricted	Open
NA	0x50000000	GPIO P0	Open	Open
NA	0xE000E100	NVIC	Restricted[5]	Open

8.2 Application signals – software interrupts (SWI)

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

Table 5: Allocation of software interrupt vectors to SoftDevice signals

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

8.3 Programmable peripheral interconnect (PPI)

PPI 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 using the Cortex Microcontroller Software Interface Standard (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 PPI channels and groups, as described in the table below, are in use by the SoftDevice.

When the SoftDevice is enabled, the application program must not change the configuration of PPI channels or groups used by the SoftDevice. Failing to comply with this will cause the SoftDevice to not operate properly.

Table 6: Assigning PPI channels between the application and SoftDevice

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 - 13	Channels 0 - 15

[3] See section Programmable peripheral interconnect (PPI) on page 18 for limitations on the use of PPI when the SoftDevice is enabled.

[4] See section Memory isolation and runtime protection on page 11 and Peripheral runtime protection on page 19 for limitations on the use of MWU when the SoftDevice is enabled.

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

[6] Available to the application through the Radio Timeslot API, see Concurrent multiprotocol implementation using the Radio Timeslot API on page 23.

[7] Blocked only when Radio Notification signal is enabled. See Application signals – software interrupts (SWI) on page 18 for software interrupt allocation.

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
SoftDevice	Channels 14 - 15[8]	-

Table 7: Assigning preprogrammed channels between the application and SoftDevice

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	-	Channels 20 - 31
SoftDevice	Channels 20 - 31	-

Table 8: Assigning PPI groups between the application and SoftDevice

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

8.4 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.

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

Table 9: SVC number allocation

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

8.5 Peripheral runtime protection

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

Protected peripheral registers are readable by the application. An attempt to perform a write to a protected peripheral register will result in a Hard Fault. Note that the peripherals are only protected when the SoftDevice is enabled, otherwise they are available to the application. See Table 4: Peripheral protection and usage by SoftDevice on page 16 for an overview of the peripherals with access restrictions due to the SoftDevice.

8.6 External and miscellaneous requirements

For correct operation of the SoftDevice, it is a requirement that the 16 MHz crystal oscillator (16 MHz XOSC) startup time is less than 1.5 ms.

The external clock crystal and other related components must be chosen accordingly. Data for the crystal oscillator input can be found in the product specification (nRF51822 Product Specification) for the SoC.

[8] Available to the application in Radio Timeslot API timeslots, see Concurrent multiprotocol implementation using the Radio Timeslot API on page 23.

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 2), otherwise the behavior of the SoftDevice is undefined and the SoftDevice might malfunction.

Chapter 9
Flash memory API

The System on Chip (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 BLE 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 62 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 10: Behavior with BLE traffic and concurrent flash write/erase on page 21 to improve the probability of flash operation success.

> **Important:** Flash page (1024 bytes) erase takes approximately 22 ms and a 4 byte flash write takes approximately 48 µs. 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 probability of success, and reduce chances of affecting BLE performance.

Table 10: Behavior with BLE traffic and concurrent flash write/erase

BLE activity	Flash write/erase (flash write size is assumed to be four bytes)
High Duty cycle directed advertising	Does not allow flash operation while advertising is active (maximum 1.28 seconds). In this case, retrying flash operation will only succeed after the advertising activity has finished.
All possible BLE roles running concurrently (connections as a central, peripheral, advertiser, and scanner).	Low to medium probability of flash operation success Probability of success increases with: • Lower bandwidth configurations. • Increase in connection interval and advertiser interval. • Decrease in scan window. • Increase in scan interval.
8 high bandwidth connections as a central 1 high bandwidth connection as a peripheral All active connections fulfill the following criteria: • Supervision time-out > 6 x connection interval • Connection interval ≥ 85 ms • All central connections have the same connection interval	High probability of flash write success. Medium probability of flash erase success. (High probability if the connection interval is > 107 ms)
8 high bandwidth connections as a central	High probability of flash operation success.

BLE activity	Flash write/erase (flash write size is assumed to be four bytes)
All active connections fulfill the following criteria: • Supervision time-out > 6 x connection interval • Connection interval ≥ 80 ms • All connections have the same connection interval	
8 low bandwidth connections as a central All active connections fulfill the following criteria: • Supervision time-out > 6 x connection interval • Connection interval ≥ 40 ms • All connections have the same connection interval	High probability of flash operation success.
1 connection as a peripheral All active connections fulfill the following criteria: • Supervision time-out > 6 x connection interval	High probability of flash operation success.
Connectable Undirected Advertising Nonconnectable Advertising Scannable Advertising Connectable Low Duty Cycle Directed Advertising	High probability of flash operation success.
No BLE activity	Flash operation will always succeed.

Chapter 10
Multiprotocol support

Multiprotocol support allows a developer to implement their own 2.4 GHz proprietary protocol in the application; both when the SoftDevice is not in use (non-concurrent), or 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 BLE events.

10.1 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.

10.2 Concurrent multiprotocol implementation using the Radio Timeslot API

The Radio Timeslot API allows the nRF51 device to be part of a network using the SoftDevice protocol stack and an alternative network of wireless devices at the same time.

The 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(s) 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 16 and Table 4: Peripheral protection and usage by SoftDevice on page 16.

10.2.1 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.

> **Important:** 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.

10.2.2 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 cancelled due to an overlapping SoftDevice activity that has a higher scheduling priority.

10.2.3 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.

10.2.4 Scheduling

The SoftDevice includes a scheduler which manages radio timeslots, 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 62 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.

> **Important:** 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 39 for more information. If Radio Notifications are enabled, Radio Timeslots will be notified.

10.2.5 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. At this priority level only flash writes might be affected. 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 54 for more information on how priorities work together with other modules like the BLE 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 schedulled

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 my result in an unrecoverable fault..

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

> **Important:** 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.

10.2.6.1 API calls

These are the API calls defined for the S130 SoftDevice:

Table 11: API calls

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.

10.2.6.2 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, see Events - SoftDevice to application on page 9. The following events are defined:

Table 12: Radio Timeslot events

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 overlaping activity of higher priority.
NRF_EVT_RADIO_SIGNAL_ CALLBACK_INVALID_RETURN	Signal handler: The last signal hander return value contained invalid parameters and the timeslot was ended.

10.2.6.3 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 66.

Table 13: Radio Timeslot signals

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

10.2.6.4 Signal handler return actions

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

Table 14: Signal handler action return values

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.

10.2.6.5 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.

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

10.2.6.6 The signal handler runs at interrupt priority level 0

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 66), SVC calls are not available in the signal handler.

> **Important:** 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.

10.3 Radio Timeslot API usage scenarios

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

10.3.1 Complete session example

This section describes a complete Radio Timeslot session.

Figure 3: Complete Radio Timeslot session example on page 28 shows a complete Timeslot session. In this case, only timeslot requests from the application are being scheduled, there is no SoftDevice activity.

At start, the application calls the API to open a session and to request a first timeslot (which must be of type *earliest possible*). The SoftDevice schedules the timeslot. At the start of the timeslot, the SoftDevice calls the application signal hander with the START signal. After this, the application is in control and has access to the peripherals. The application will then typically set up TIMER0 to expire before the end of the timeslot, to get a signal 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 (see the middle timeslot in Figure 3: Complete Radio Timeslot session example on page 28). The signal handler is called with the START signal at the start of the timeslot.
The application then has control, but must arrange for a signal to come towards the end of the timeslot.
As the return value for the last signal in the timeslot, the signal handler requests a new timeslot using the REQUEST_AND_END action.

Eventually, the application does not require the radio any more. So, at the last signal in the last timeslot (Figure 3: Complete Radio Timeslot session example on page 28), the application returns END from the signal handler. The SoftDevice then sends an IDLE event to the application event handler. The application calls session_close, and the SoftDevice sends the CLOSED event. The session has now ended.

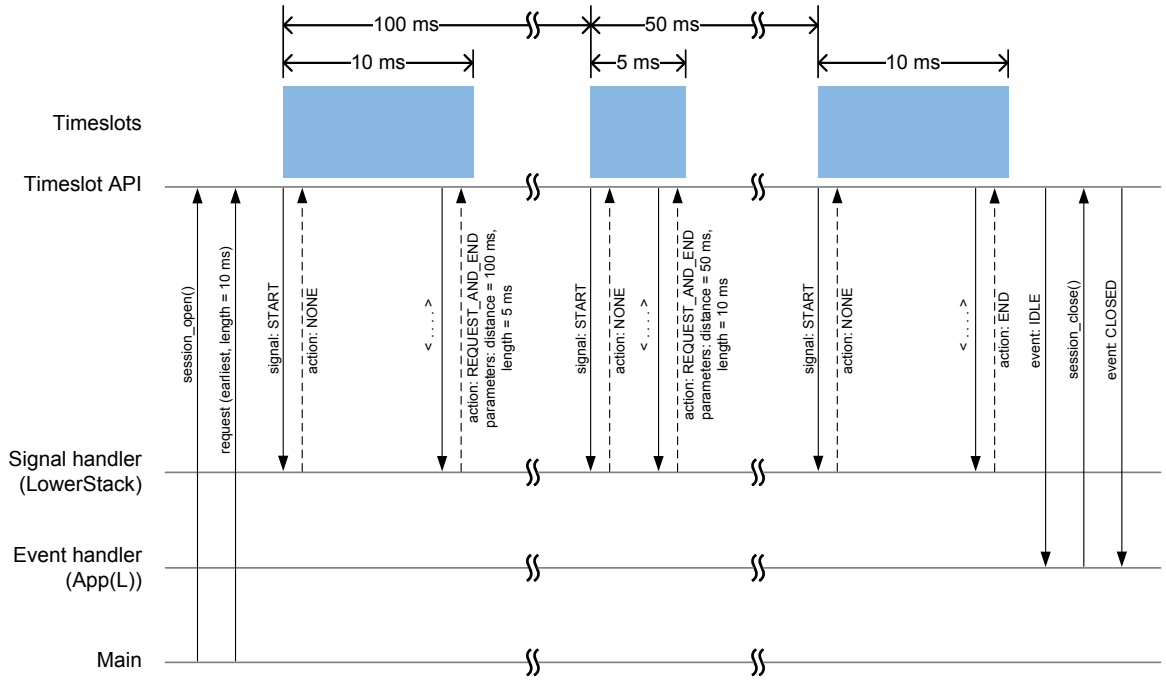

Figure 3: Complete Radio Timeslot session example

10.3.2 Blocked timeslot scenario

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

Figure 4: Blocked timeslot scenario on page 28 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.

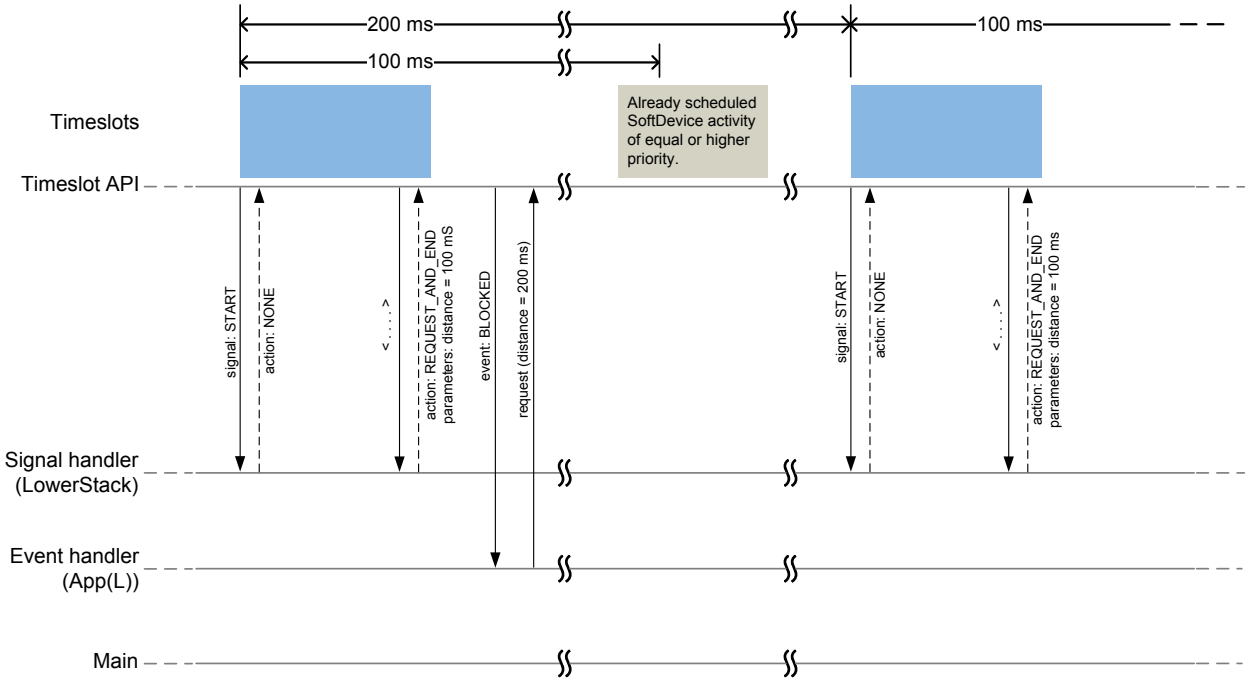

Figure 4: Blocked timeslot scenario

10.3.3 Canceled timeslot scenario

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

Figure 5: Canceled timeslot scenario on page 30 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.

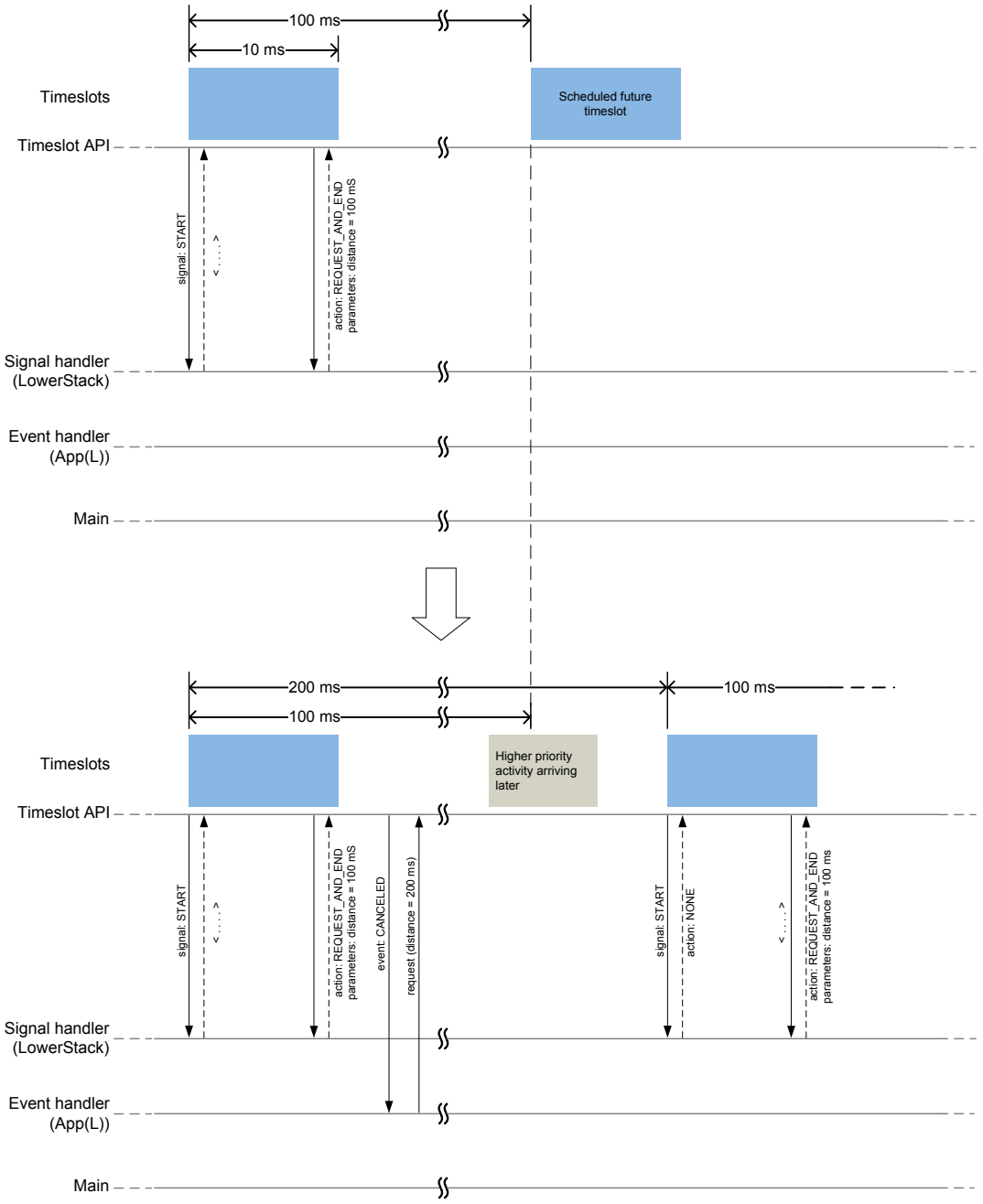

Figure 5: Canceled timeslot scenario

10.3.4 Radio Timeslot extension example

An application can use Radio Timeslot extension to create long continuous timeslots that will give the application as much radio time as possible while disturbing the SoftDevice activities as little as possible.

In the first timeslot in Figure 6: Radio Timeslot extension example on page 31 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.

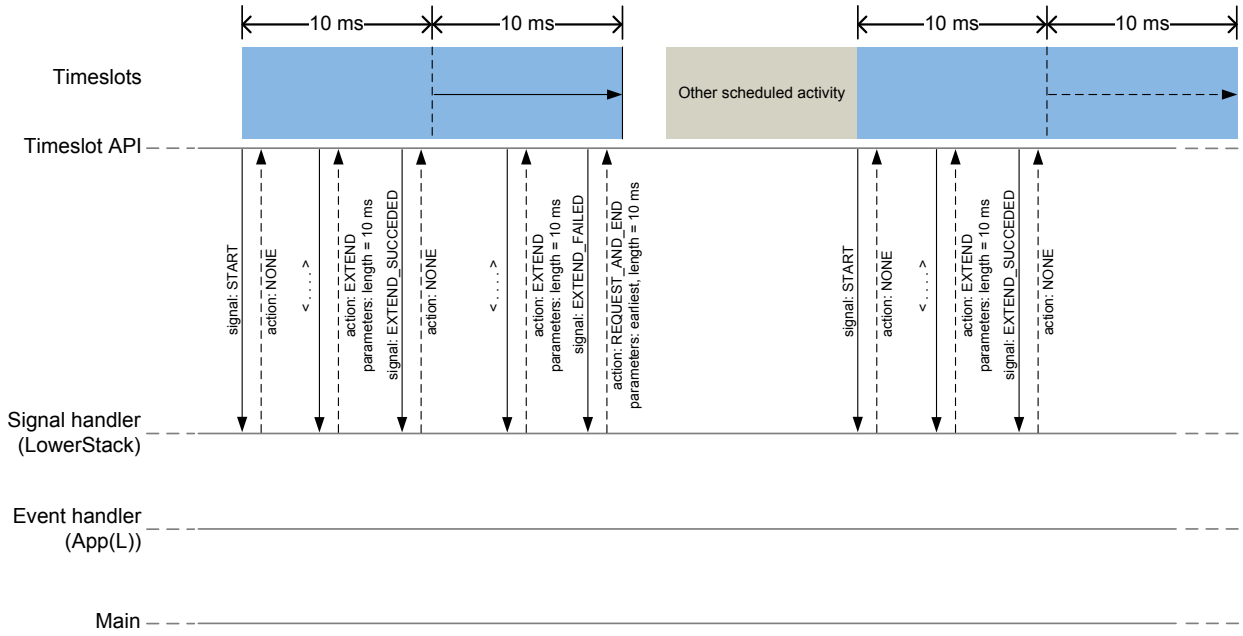

Figure 6: Radio Timeslot extension example

Chapter 11
Bluetooth® low energy protocol stack

The *Bluetooth*® 4.2 compliant low energy Host and Controller implemented by the SoftDevice are fully qualified with multi-role support (Central, Observer, Peripheral, and Broadcaster).

The SoftDevice allows applications to implement standard *Bluetooth*® low energy profiles as well as proprietary use case implementations. The API is defined above the Generic Attribute Protocol (GATT), Generic Access Profile (GAP), and Logical Link Control and Adaptation Protocol (L2CAP).

The nRF5 Software Development Kit (nRF5 SDK) complements the SoftDevice with Service and Profile implementations. Single-mode System on Chip (SoC) applications are enabled by the full BLE protocol stack and nRF51 Series SoC.

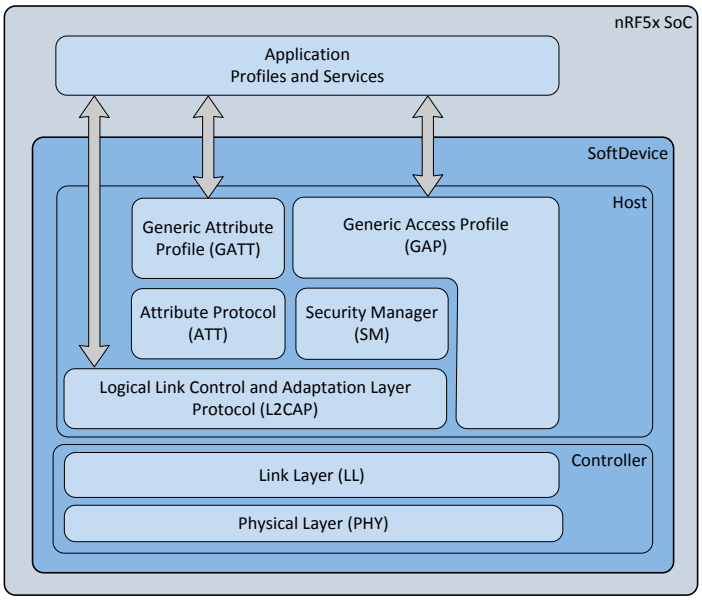

Figure 7: SoftDevice stack architecture

11.1 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 all profiles and services (with exceptions as noted in Table 15: Supported profiles and services on page 32) as well as additional proprietary profiles.

Table 15: Supported profiles and services

Adopted profile	Adopted services
HID over GATT	HID Battery Device Information

Adopted profile	Adopted services
Heart Rate	Heart Rate Device Information
Proximity	Link Loss Immediate Alert Tx Power
Blood Pressure	Blood Pressure Device Information
Health Thermometer	Health Thermometer Device Information
Glucose	Glucose Device Information
Phone Alert Status	Phone Alert Status
Alert Notification	Alert Notification
Time	Current Time Next DST Change Reference Time Update
Find Me	Immediate Alert
Cycling Speed and Cadence	Cycling Speed and Cadence Device Information
Running Speed and Cadence	Running Speed and Cadence Device Information
Location and Navigation	Location and Navigation
Cycling Power	Cycling Power
Scan Parameters	Scan Parameters
Weight Scale	Weight Scale Body Composition User Data Device Information
Continuous Glucose Monitoring	Continuous Glucose Monitoring Bond Management Device Information
Environmental Sensing	Environmental Sensing

Adopted profile	Adopted services
Pulse Oximeter	Pulse Oximeter Device Information Bond Management Battery Current Time
Object Transfer (Currently not supported by the SoftDevice)	Object Transfer
Automation IO	Automation IO
	Indoor Positioning
Internet Protocol Support (Currently not supported by the SoftDevice)	

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

11.2 *Bluetooth*® low energy features

The BLE protocol stack in the SoftDevice has been designed to provide an abstract but flexible interface for application development for *Bluetooth*® low energy devices.

GAP, GATT, SM, and L2CAP are implemented in the SoftDevice and managed through the API. The SoftDevice implements GAP and GATT procedures and modes that are common to most profiles such as the handling of discovery, connection, data transfer, and bonding.

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

Table 16: API features in the BLE stack

API features	Description
Interface to: GATT / GAP	Consistency between APIs 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.

Table 17: GAP features in the BLE stack

GAP features	Description
Multi-role:	Central, Peripheral, Observer, and Broadcaster can run concurrently with a connection.
Multiple bond support	Keys and peer information stored in application space. No restrictions in stack implementation.
Security Mode 1: Levels 1, 2 & 3	Support for all levels of SM 1.

Table 18: GATT features in the BLE stack

GATT features	Description
Full GATT Server	Support for three concurrent ATT server sessions. Includes configurable Service Changed support.
Support for authorization	Enables control points. Enables freshest data. Enables GAP authorization.
Full GATT Client	Flexible data management options for packet transmission with either fine control or abstract management.
Implemented GATT Sub-procedures	Discover all Primary Services. Discover Primary Service by Service UUID. Find included Services. Discover All Characteristics of a Service. Discover Characteristics by UUID. Discover All Characteristic Descriptors. Read Characteristic Value. Read using Characteristic UUID. Read Long Characteristic Values. Read Multiple Characteristic Values (Client only). Write Without Response. Write Characteristic Value. Notifications. Indications. Read Characteristic Descriptors. Read Long Characteristic Descriptors. Write Characteristic Descriptors.

GATT features	Description
	Write Long Characteristic Values. Write Long Characteristic Descriptors. Reliable Writes.

Table 19: Security Manager (SM) features in the BLE stack

Security Manager 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 MITM (Man in the middle) protection	Allows for per-link elevation of the encryption security level.
Pairing methods: Just works, Passkey Entry and Out of Band	API provides the application full control of the pairing sequences.

Table 20: Attribute Protocol (ATT) features in the BLE 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.
Max MTU size 23 bytes	Up to 20 bytes of user data available per packet.

Table 21: Controller, Link Layer (LL) features in the BLE stack

Controller, Link Layer features	Description
Master role Scanner/Initiator roles	The SoftDevice supports eight concurrent master connections and an additional Scanner/Initiator role. When the maximum number of simultaneous connections are established, the Scanner role will be supported for new device discovery. However, the initiator is not available at that time.
Master connection parameter update	
Channel map configuration	Setup of channel map for all master connections from the application. Accepting update for the channel map for a slave connection.
Slave role Advertiser/broadcaster role	Supports advertiser, or one peripheral connection and one additional broadcaster.
Connection parameter update	Central role may initiate connection parameter update. Peripheral role will accept connection parameter update.

Controller, Link Layer features	Description
Encryption	
RSSI	Signal strength measurements during advertising, scanning, and central and peripheral connections.

Table 22: Proprietary features in the BLE stack

Proprietary features	Description
TX Power control	Access for the application to change TX power settings anytime.
Enhanced Privacy 1.1 support	Synchronous and low power solution for *Bluetooth*® low energy enhanced privacy with hardware-accelerated address resolution for whitelisting.
Master Boot Record (MBR) for Device Firmware Update (DFU)	Enables over-the-air SoftDevice replacement, giving full SoftDevice update capability.

11.3 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 Security Manager (SM) Initiator is only instantiated once for all connections. The Link Layer also has concurrent procedure limitations that are handled inside the SoftDevice without requiring management from the application.

Table 23: Limitations on procedure concurrency

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 link layer control procedures (connection parameter update and encryption). In this case, the GAP module will queue the LL procedures and execute them in sequence.
SM	SM procedures cannot be executed in parallel for connections as a Central. That is, each SM procedure must run to completion before the next procedure begins across all connections as a Central. For example `sd_ble_gap_authenticate()`.
LL	The LL Disconnect procedure has no limitations and can be executed on any, or all, links simultaneously. The LL connection parameter update and encryption establishment procedure on a master link can only be executed on one master link at a time. Accepting connection parameter update and encryption establishment on a slave link is always

Protocol procedures	Limitation with multiple connections
	allowed irrespective of any control procedure running on master links.

11.4 BLE role configuration

The S130 SoftDevice stack supports concurrent operation in multiple *Bluetooth*® low energy roles. The roles available can be configured when the S130 SoftDevice stack is enabled at runtime.

The SoftDevice provides a mechanism for enabling the number of Central or Peripheral roles the application can run concurrently. The SoftDevice can be configured with one connections as a Peripheral and up to eight connections as a Central. The SoftDevice supports running one Advertiser or Broadcaster and one Scanner or Observer concurrently with the BLE connections.

An Initiator can only be started if there are less than eight connections established as a Central. Similarly, a connectable Advertiser can only be started if there are no connections as a Peripheral established. If there is a total of eight connections running, an Advertiser or Broadcaster cannot run simultaneously with a Scanner or Observer.

When the SoftDevice is enabled it will allocate memory for the connections the application has requested. See SoftDevice memory usage on page 51 chapter for more details.

The SoftDevice supports three predetermined bandwidth levels: low, mid(medium), and high. The bandwidth levels are defined per connection interval, so the connection interval of the connection will also affect the total bandwidth. Bandwidth configuration is an optional feature and if the application chooses to not use the feature the SoftDevice will use default bandwidth configurations. By default, connections as a Central will be set to medium bandwidth and connections as a Peripheral will be set to high bandwidth. The bandwidth can also be configured individually for both transmitting and receiving, allowing for more fine-grained control with asymmetric bandwidth.

When using the bandwidth feature of the SoftDevice the application needs to provide the SoftDevice with information about the bandwidth configurations the application intends to use. This is done when enabling the BLE stack and will allow the SoftDevice to allocate memory pools large enough to fit the bandwidth configurations of all connections. Otherwise connections can fail to be established because there is not enough memory available for the bandwidth requirement of the connection. If the application changes the bandwidth configuration when the link is disconnected it might fail to reconnect because of internal fragmentation of the SoftDevice memory pools. It is therefore recommended to disconnect all links when changing the bandwidth configurations of links.

Bandwidth and multilink scheduling can both affect each other. See Scheduling on page 54 for details. Knowledge about multilink scheduling can be used to get predictable performance on all links. Refer Suggested intervals and windows on page 62 for details about recommended configurations.

Chapter 12
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.

12.1 Radio Notification Signals

The Radio Notification signals are sent prior to and at the end of SoftDevice Radio Events or application Radio Events [9], i.e. defined time intervals of radio operation.

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 18.

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 24: Notation and terminology for the Radio Notification used in this chapter on page 40 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 39 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.

When there is not sufficent 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 where tgap is too small and the notification signals will not be available between the events on page 40.

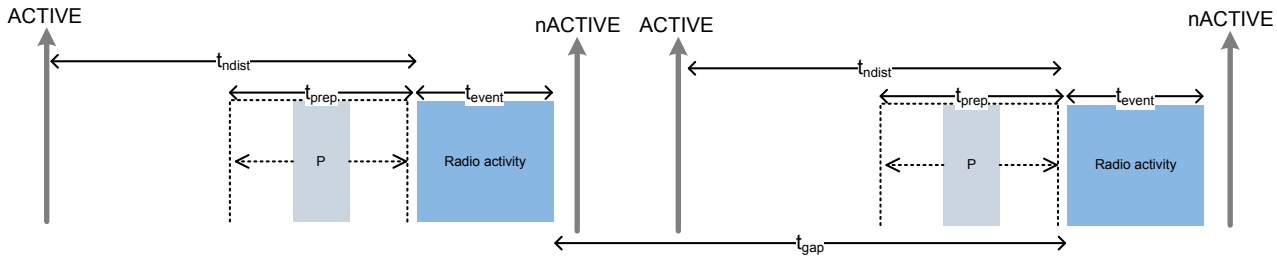

Figure 8: Two radio events with ACTIVE and nACTIVE signals

[9] Application Radio Events are defined as Radio Timeslots, see Multiprotocol support on page 23.

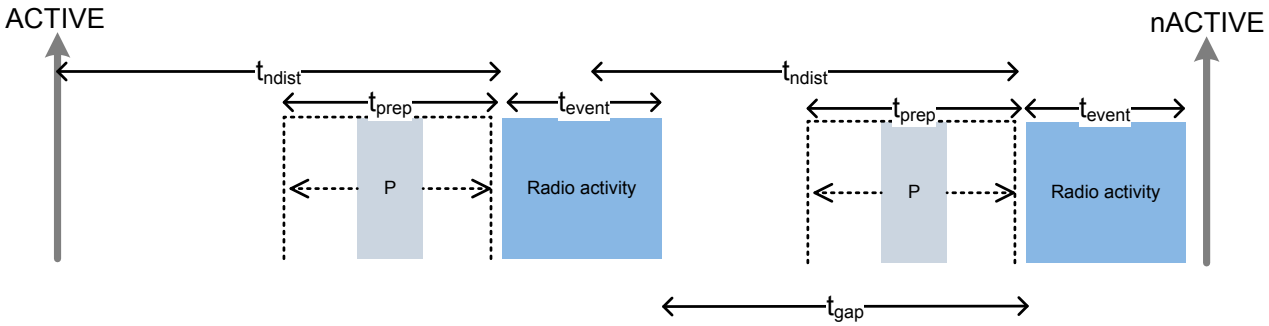

Figure 9: Two radio events where t_{gap} is too small and the notification signals will not be available between the events

Table 24: Notation and terminology for the Radio Notification used in this chapter

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_{event}	The total time of a Radio 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. **Important:** All packet data to send in an event should have been sent to the stack t_{prep} before the Radio Event starts.

Label	Description	Notes
t_P	Time used for preprocessing before the Radio Event.	
$t_{interval}$	Time period of periodic protocol Radio Events (e.g. BLE connection interval).	
t_{EEO}	Minimum time between central role Radio Events (Event-to-Event Offset).	The minimum time between the start of adjacent central role Radio events, and between the last central role radio event and the scanner. t_{EEO-C0} can be different for different connections, and it depends on the bandwidth configuration of the connection. See BLE role configuration on page 38 for more information on the BLE stack configuration.
$t_{ScanReserved}$	Reserved time needed by the SoftDevice for each ScanWindow	

Table 25: BLE Radio Notification timing ranges

Value	Range (µs)
t_{ndist}	800, 1740, 2680, 3620, 4560, 5500 (Configured by the application)
t_{event}	2750 to 5500 - Undirected and scannable advertising, 0 to 31 byte payload, 3 channels 2150 to 2950 - Non-connectable advertising, 0 to 31 byte payload, 3 channels 1.28 seconds - Directed advertising, 3 channels Slave bandwidth LOW : 2050 Slave bandwidth MID : 4000 Slave bandwidth HIGH : 6900 Master bandwidth LOW : 1900 Master bandwidth MID : 3850 Master bandwidth HIGH : 6725 Note: Master and Slave t_{event} values are for full duplex transfer of full packets.
t_{prep}	167 to 1542
t_P	≤ 165
t_{EEO}	Bandwidth LOW : 2100 Bandwidth MID : 4025 Bandwidth HIGH : 6900

Value	Range (µs)
	Note: These values are for full duplex transfer of full packets.
$t_{ScanReserved}$	1125

Based on the numbers from Table 25: BLE Radio Notification timing ranges on page 41, 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.

> **Important:** t_{prep} may be greater than t_{ndist} when t_{ndist} = 800. If time is required to handle packets or manage peripherals before interrupts are generated by the stack, t_{ndist} must be set larger than 1550.

12.2 Radio Notification on connection events as a Central

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

Refer to Table 24: Notation and terminology for the Radio Notification used in this chapter on page 40 for the notations used in the text and the figures of this section. For a comprehensive understanding of role scheduling see Scheduling on page 54.

For a central link multiple packets may be exchanged within a single Radio (connection) Event. This is shown in Figure 10: A BLE central link with multiple packet exchange per connection event on page 42.

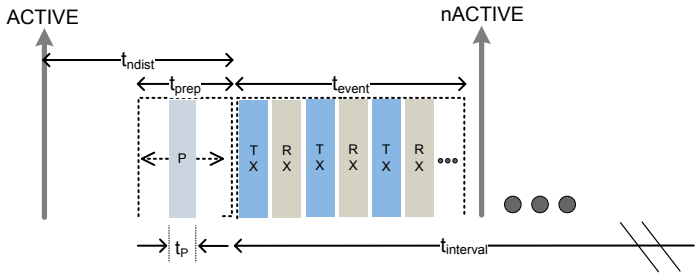

Figure 10: A BLE central link with multiple packet exchange per connection event

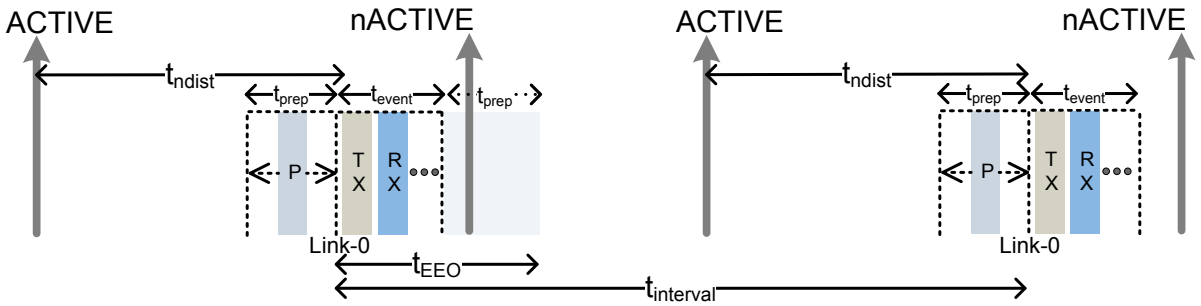

Figure 11: BLE Radio Notification signal in relation to a single active link

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: BLE Radio Notification signal in relation to a single active link on page 42), the following condition must hold:

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

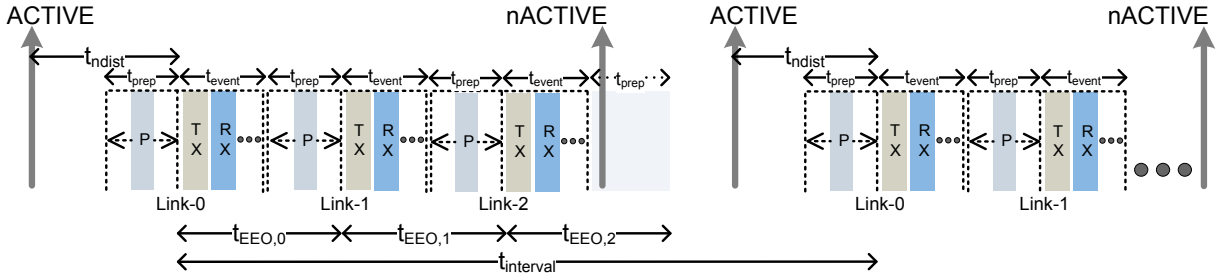

Figure 12: BLE Radio Notification signal in relation to 3 active links

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 3 links is shown in Figure 12: BLE Radio Notification signal in relation to 3 active links on page 43). To ensure that the ACTIVE notification signal will be available to the application at the configured time when 3 links are established as a Central, the following condition must hold:

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

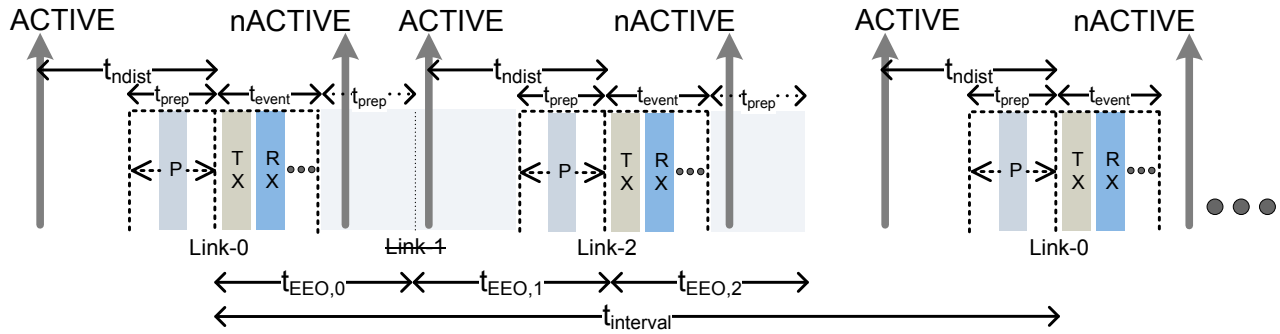

Figure 13: BLE Radio Notification signal when the number of active links as a Central is 2

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:

$\Sigma_{i=m,...n}\, t_{EEO,i} + t_{prep} > t_{ndist}$ where Link-m, ..., Link-n are consecutive inactive central links.

For example, in the scenario shown in Figure 13: BLE Radio Notification signal when the number of active links as a Central is 2 on page 43 Link-1 is not active, a gap of $t_{EEO,1}$ 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:

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

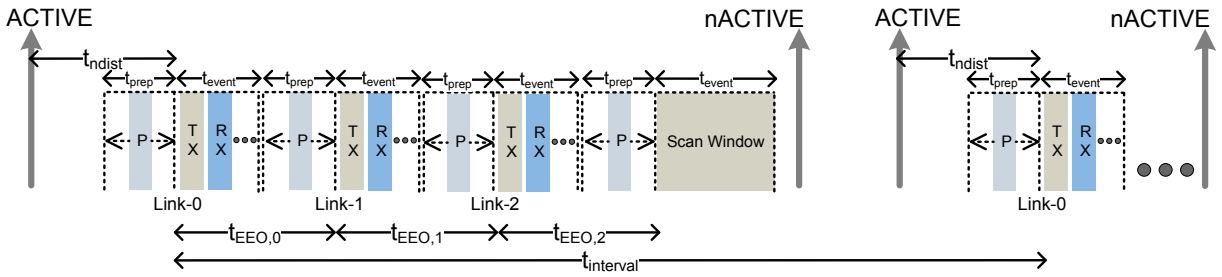

Figure 14: BLE Radio Notification signal in relation to 3 active connections as a Central while scanning

A SoftDevice may, additionally, run a scanner in parallel to the central links. This is shown in Figure 14: BLE Radio Notification signal in relation to 3 active connections as a Central while scanning on page 44, where 3 central links and a scanner have been established. To ensure in this case that the ACTIVE notification signal will be available to the application at the configured time, the following condition must hold:

$t_{ndist} + t_{EEO,0} + t_{EEO,1} + t_{EEO,2} + \text{Scan Window} + t_{ScanReserved} < t_{interval}$

12.3 Radio Notification on connection events as a Peripheral

Similar to central links, peripheral links may also include multiple packet exchange within a single Radio (connection) Event.

Radio Notification events are as shown in Figure 15: A BLE peripheral link with multiple packet exchange per connection event on page 44.

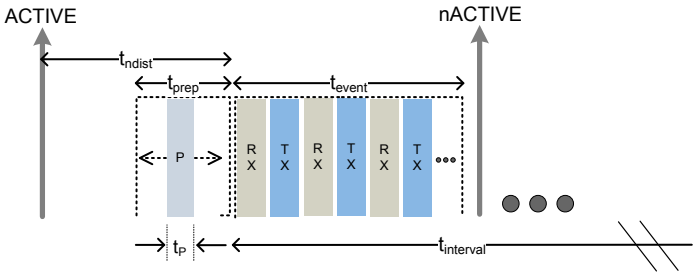

Figure 15: A BLE peripheral link with multiple packet exchange per connection event

To ensure that the ACTIVE notification signal is available to the application at the configured time when a single peripheral link is established, the following condition must hold (with one exception, see Table 26: Maximum Peripheral packet transfer per BLE Radio Event for given combinations of Radio Notification distances and connection intervals. Assumes full length packets and full-duplex, HIGH/HIGH BLE bandwidth configuration. on page 45):

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

The SoftDevice will limit the length of a Radio Event (t_{event}), thereby reducing the maximum number of packets exchanged, to accommodate the selected t_{ndist}. Figure 16: Consecutive peripheral Radio Events with BLE Radio Notification signals on page 45 shows consecutive Radio Events with Radio Notification signal and illustrates the limitation in t_{event} which may be required to ensure t_{ndist} is preserved.

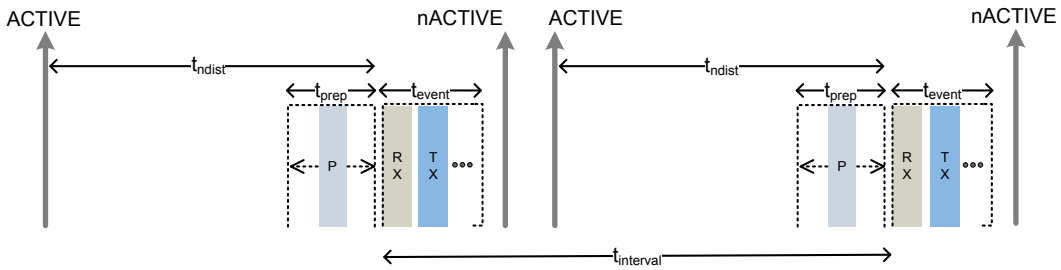

Figure 16: Consecutive peripheral Radio Events with BLE Radio Notification signals

Table 26: Maximum Peripheral packet transfer per BLE Radio Event for given combinations of Radio Notification distances and connection intervals. Assumes full length packets and full-duplex, HIGH/HIGH BLE bandwidth configuration. on page 45 shows the limitation on the maximum number of full length packets which can be transferred per Radio Event for given combinations of t_{ndist} and $t_{interval}$.

Table 26: Maximum Peripheral packet transfer per BLE Radio Event for given combinations of Radio Notification distances and connection intervals. Assumes full length packets and full-duplex, HIGH/HIGH BLE bandwidth configuration.

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

12.4 Radio Notification with concurrent Peripheral and central connection events

A peripheral link is arbitrarily positioned with respect to the central links. Therefore, if the peripheral connection event occurs too close to the central connection event, the notification signal before the peripheral connection event might not be available to the application, .

Figure 17: Example: the gap between the links as a Central and the Peripheral is too small to trigger the notification signal on page 45 shows an example where the time distance between the central and the peripheral events is too small to allow the SoftDevice to trigger the ACTIVE notification signal.

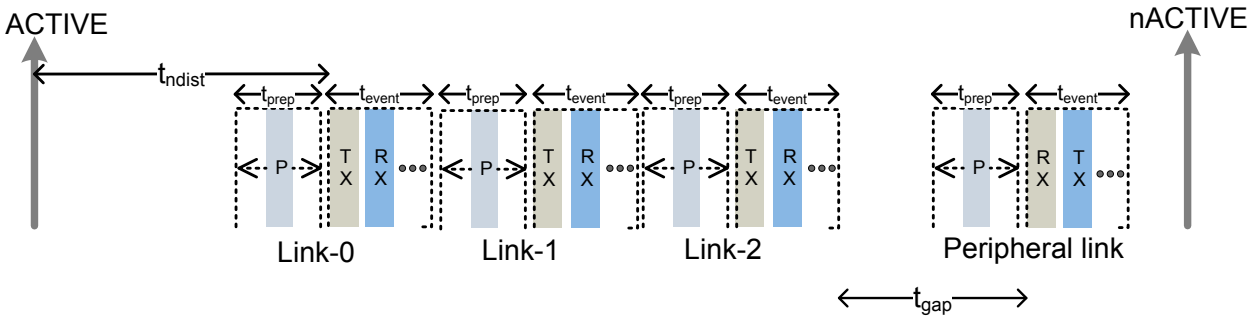

Figure 17: Example: the gap between the links as a Central and the Peripheral is too small to trigger the notification signal

If the following condition is met:

$t_{gap} > t_{ndist}$

the notification signal will arrive, as illustrated in Figure 18: Example: the gap between the links as a Central and the Peripheral is sufficient to trigger the notification signal on page 46.

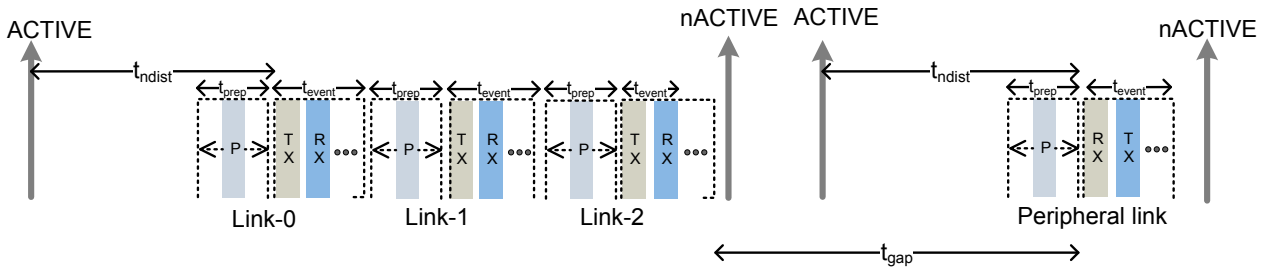

Figure 18: Example: the gap between the links as a Central and the Peripheral is sufficient to trigger the notification signal

Chapter 13
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 nRF51 software architecture includes a Master Boot Record (MBR) (Figure 1: System on Chip application with the SoftDevice on page 8). The MBR is necessary in order for the bootloader to update the SoftDevice, or to update the bootloader itself. The MBR is a required component in the system. The inclusion of a bootloader is optional.

13.1 Master Boot Record

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

The Master Boot Record (MBR) module occupies a defined region in the SoC program memory where the System Vector table resides.

All exceptions (reset, hard fault, interrupts, SVC) are, first, processed by the MBR and then are forwarded to the appropriate handlers (for example the bootloader or the SoftDevice exception handlers). See Interrupt model and processor availability on page 65 for more details on the interrupt forwarding scheme.

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.

13.2 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 BLE communication.

The bootloader is supported in the SoftDevice architecture by using a configurable base address for the bootloader in the application Flash Region. The base address is configured by setting the UICR.BOOTLOADERADDR register. The bootloader is responsible for determining the start address of the application. It uses `sd_softdevice_vector_table_base_set(uint32_t address)` to tell the SoftDevice where the application starts.

The bootloader is also responsible for keeping track of, and verifying the integrity of the SoftDevice. If an unexpected reset occurs during an update of the SoftDevice, it is the responsibility of the bootloader to detect this and resume the update procedure.

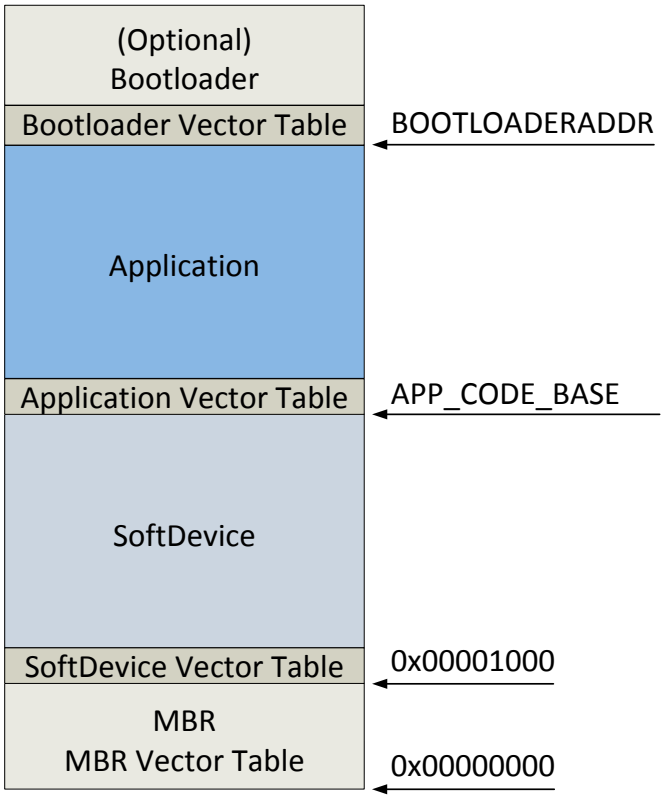

Figure 19: MBR, SoftDevice and bootloader architecture

13.3 Master Boot Record (MBR) and SoftDevice reset procedure

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

The MBR and SoftDevice reset behavior is as follows:

- 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.

13.4 Master Boot Record (MBR) and SoftDevice initialization procedure

The SoftDevice can be enabled by the bootloader.

The bootloader can enable the SoftDevice through the following step-by-step 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 through the following step-by-step 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.

Chapter 14
SoftDevice information structure

The SoftDevice binary file contains an information structure.

The structure is illustrated in Figure 20: SoftDevice information structure on page 50. The location of the structure, the SoftDevice size, and the firmware_id can be obtained at run time by the application using macros defined in the `nrf_sdm.h` header file. Accessing this structure requires that the SoftDevice is not read back protected. The information structure can also be accessed by parsing the binary SoftDevice file.

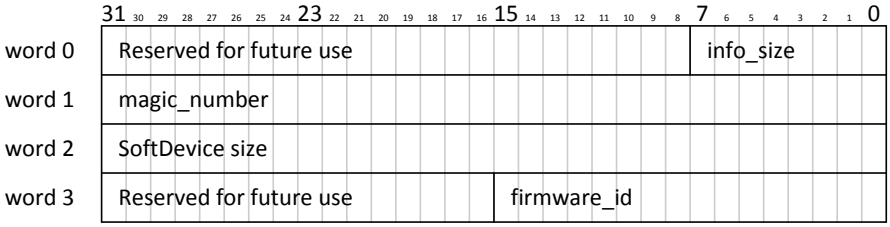

Figure 20: SoftDevice information structure

Chapter 15
SoftDevice memory usage

The SoftDevice shares the available flash memory and RAM on the nRF51 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 Table 27: S130 Memory resource requirements for flash on page 52 and Table 28: S130 Memory resource requirements for RAM on page 52. In addition, depending on the runtime configuration, the SoftDevice will require:

- Additional RAM for *Bluetooth*® low energy (BLE) roles and bandwidth (see Role configuration on page 53)
- Attributes (see Attribute table size on page 53)
- Security (see Security configuration on page 53)
- UUID storage (see Vendor specific UUID counts on page 53)

15.1 Memory resource map and usage

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

Figure 21: Memory resource map on page 51 illustrates the memory usage of the SoftDevice alongside an 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 8 bytes of RAM when not enabled. Once enabled, the RAM usage of the SoftDevice increases: the RAM requirements of an enabled SoftDevice are detailed in Table 28: S130 Memory resource requirements for RAM on page 52. With the exception of the call stack, the RAM usage for the SoftDevice is always isolated from the application usage; thus 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, see Table 28: S130 Memory resource requirements for RAM on page 52 for minimum RAM requirements.

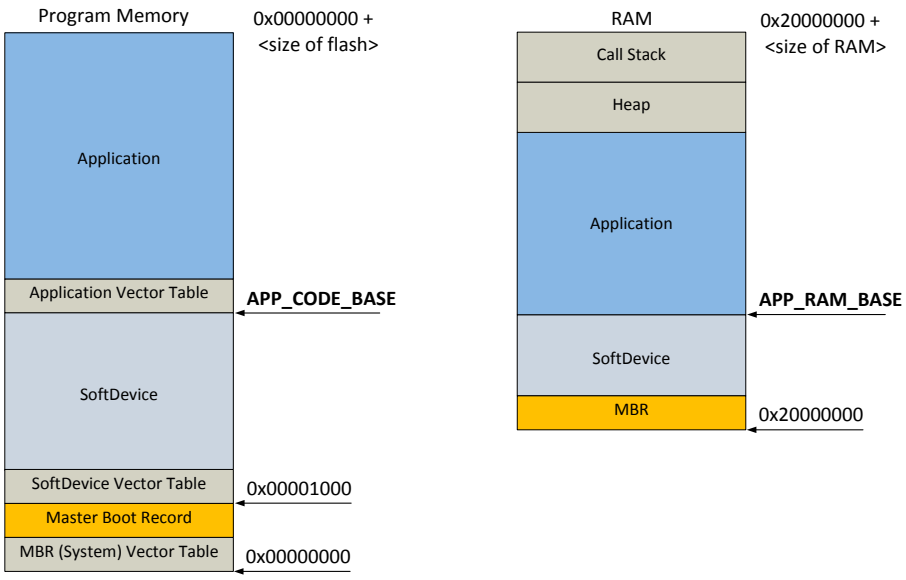

Figure 21: Memory resource map

15.1.1 Memory resource requirements

The tables below show the memory resource requirements both when the S130 SoftDevice is enabled and disabled.

Flash

Table 27: S130 Memory resource requirements for flash

Flash	Value
Required by the SoftDevice	104 kB[10]
Required by the MBR	4 kB
APP_CODE_BASE address (absolute value)	0x0001B000

RAM

Table 28: S130 Memory resource requirements for RAM

RAM	S130 Enabled	S130 Disabled
Required by the SoftDevice (in bytes)	0x1288 + Configurable Resources Minimum: 0x13C8 (5064)	8
APP_RAM_BASE address (minimum required value)	0x20000000 + SoftDevice RAM consumption Minimum: 0x200013C8	0x20000008

Call stack

By default, the nRF51 SoC will have a shared call stack with both application stack frames and SoftDevice stack frames, managed by the main stack pointer (MSP).

The call stack configuration is done by the application, and the MSP gets initialized on reset to the address specified by the application vector table entry 0. The application may, in its reset vector, configure the CPU to use the process stack pointer (PSP) in thread mode. This configuration is optional but may be required by an operating system (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.

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

With each release of an nRF51 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 nRF51 SoC has no hardware for detecting stack overflow, and the application is responsible for leaving enough space both for the application itself and the nRF51 SoftDevice stack requirements.

Table 29: S130 Memory resource requirements for call stack on page 53 depicts the maximum call stack size that may be consumed by the SoftDevice. The application call stack memory usage must be added to the SoftDevice call stack size, in order to determine the total call stack size, and configure it in the user application.

[10] 1 kB = 1024 bytes

Table 29: S130 Memory resource requirements for call stack

Call stack	S130 Enabled	S130 Disabled
Maximum usage	1536 bytes (0x600)	0 bytes

Heap

There is no heap required by nRF51 SoftDevices. The application is free to allocate and use a heap without disrupting the SoftDevice functionality.

15.2 Attribute table size

The size of the attribute table can be configured through the SoftDevice API when enabling the BLE stack.

The attribute table size, `ATTR_TAB_SIZE`, has a default value of 0x580 bytes. 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 minimum attribute table size is 0xD8 bytes. The amount of RAM reserved by the SoftDevice, and the minimum required start address for the application RAM, `APP_RAM_BASE`, will then change accordingly.

For more information on how to configure the attribute table size, refer to the S130 SoftDevice API.

15.3 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, their role and bandwidth 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 BLE role configuration on page 38 for more details on role configuration.

15.4 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 an 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 the `ble_gap_enable_params_t` type on `sd_ble_enable`.

15.5 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 `ble_common_enable_params_t` type on `sd_ble_enable`. The minimum number of vendor specific UUID count can be 1, whereas the default value is 10.

Chapter 16
Scheduling

The S130 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 BLE role events (central roles, peripheral roles), Flash memory API usage, and Radio Timeslot API timeslots.

If timing-events collide, their scheduling is determined by a priority system. If timing-activity A needs 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 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 timing-event at overlapping time with the same priority, and 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.

16.1 SoftDevice timing-activities and priorities

The SoftDevice supports up to eight connections as a central, up to one connection as a peripheral, an advertiser or broadcaster and an observer or scanner simultaneously. In addition to these BLE roles, Flash memory API and Radio Timeslot API can also run simultaneously.

An Initiator can only be started if there are less than eight connections established as a central. Similarly, a connectable advertiser can only be started if there is no connection as a peripheral established. See BLE role configuration on page 38 for more information on the BLE stack configuration.

Central link timing-events are added relative to already running central link timing-events. Advertiser and broadcaster timing-events are scheduled as early as possible. Peripheral link timing-events follow the timings dictated by the connected peer. Peripheral role timing-events (Peripheral link timing-event, Advertiser/Broadcaster timing-event) and central role timing-events (Central link timing-event, Initiator/Scanner timing-event) are scheduled independently and so may occur at the same time and collide. Similarly Flash access timing-event and Radio Timeslot timing-event are scheduled independently and so may occur at the same time and collide.

The different timing-activities have different priorities at different times, dependent upon their state. As an example, if a connection as a peripheral is close to its 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.Table Table 30: Scheduling priorities on page 54 summarizes the priorities:

Table 30: *Scheduling priorities*

Priority (Decreasing order)	Role state
First priority	• Connection as a peripheral during connection update procedure. • Connection setup as a peripheral (waiting for ack from peer)

Priority (Decreasing order)	Role state
	• Connection as a peripheral that is about to time-out
Second priority	• Central connections that are about to time out
Third priority	• Central connection setup (waiting for ack from peer) • Initiator • Advertiser/Broadcaster/Scanner which has been blocked consecutively for a few times. **Important:** An advertiser which is started while a link as a peripheral is active, does not increase its priority at all.
Fourth priority	• All BLE roles in states other than above run with this priority. • Flash access after it has been blocked consecutively for a few times. • Radio Timeslot with high priority.
Last priority	• Flash access • Radio Timeslot with normal priority

16.2 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 1.25 ms after the connect request was sent, as shown in Figure 22: Initiator - first connection on page 55.

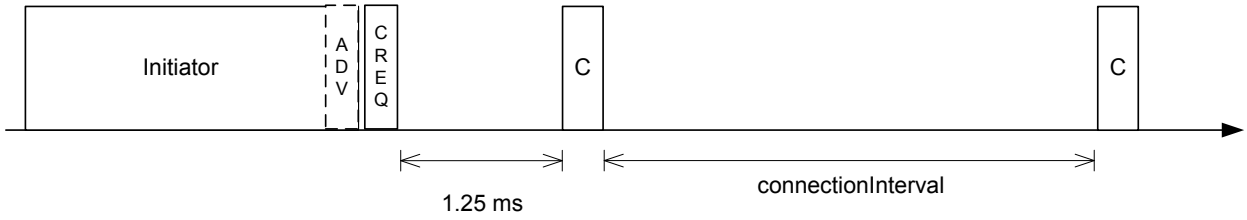

Figure 22: Initiator - first connection

When establishing a new connection with other connections already made as a central, the initiator will start asynchronously to the connected link timing-events and position 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 placed close to each other (without any free time between them). This minimum time between start of two central role timing-event is referred as t_{EEO}. t_{EEO} is propotional to the number of packets exchanged (bandwidth configuration) in timing-event, because more time is required to exchange more packets. Refer Table 25: BLE Radio Notification timing ranges on page 41 for t_{EEO} timing ranges. Figure 23: Initiator - one central connection running on page 56 illustrates the case of establishing a new central connection with one central connection already running.

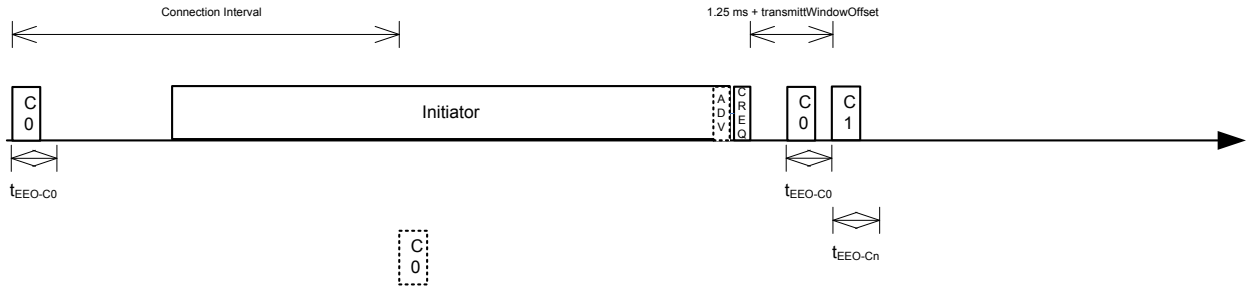

Figure 23: Initiator - one central connection running

> **Important:** The inititator is scheduled asynchronously to any other role (and any other timing-activity) and assigned higher priority to ensure faster connection setup.

When a central link disconnects, the timings of other central link timing-events remain unchanged. Figure 24: Initiator - free time due to disconnection on page 56 illustrates the case when central link C1 is disconnected, which results in free time between C0 and C2.

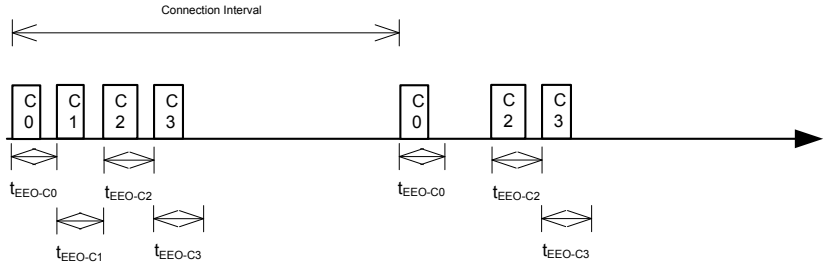

Figure 24: Initiator - free time due to disconnection

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 25: Initiator - one or more connections as a central on page 56 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. Timing-event for new connection, Cn, is positioned in the available time between C2 - C3 because that is the best fit for Cn.

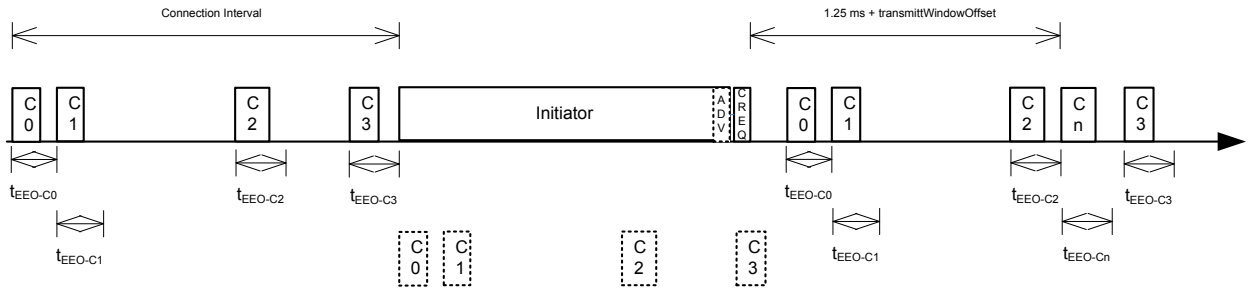

Figure 25: Initiator - one or more connections as a central

Figure 26: Initiator - free time not enough on page 57 illustrates the case when any free time between existing central link timing-events is not big enough to fit the new connection. The new central link timing-event is placed after all running central link timing-events in this case.

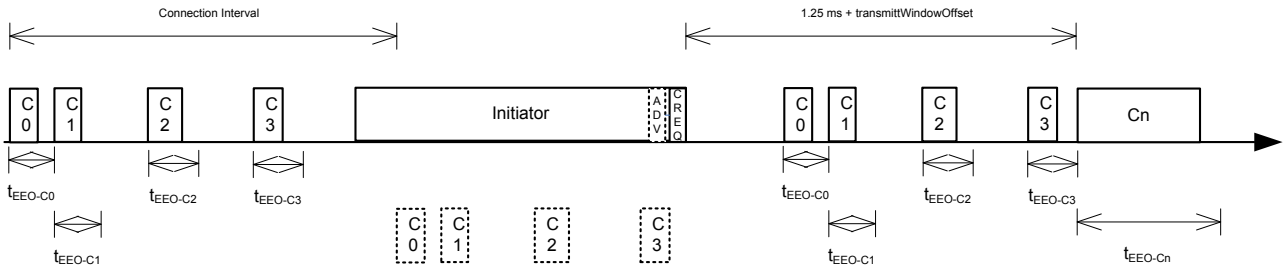

Figure 26: Initiator - free time not enough

When establishing connections to newly discovered devices, the scanner may be used for discovery followed by the initiator. In Figure 27: Initiator - fast connection on page 57, the initiator is started directly after discovering a new device to connect as fast as possible to that device. The initiator will always start asynchronously to the connected link events. The result is some link timing-events being dropped while the initiator timing-event runs. Link timing-events scheduled in the transmit window offset will not be dropped (C1). In this case time between C0 - C1 is available, and is allocated for the new connection (Cn).

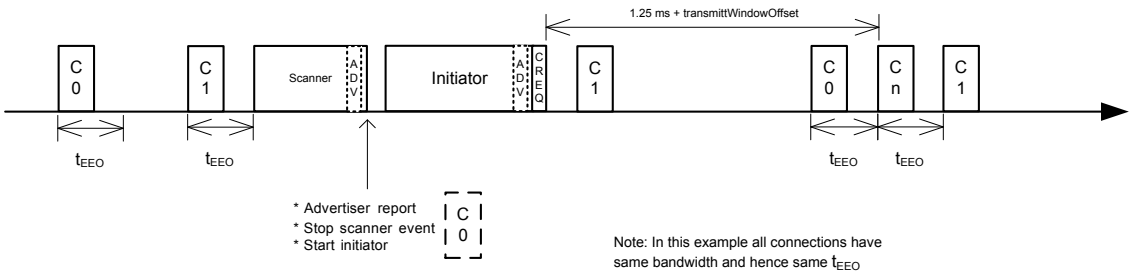

Figure 27: Initiator - fast connection

16.3 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_{EEO} depending on the bandwidth requirement of links. Refer Initiator timing on page 55 for details about t_{EEO} and Table 25: BLE Radio Notification timing ranges on page 41 for its timing ranges.

Figure 28: Multilink scheduling - one or more connections as a central, factored intervals on page 58 shows a scenario where there are two links as a central established. C0 timing-events correspond to the first connection as a central made and C1 timing-events correspond to the second connection made. C1 timing-events are initially offset from C0 timing-events by t_{EEO-C0}. C1, in this example, have 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_{EEO-C0}.

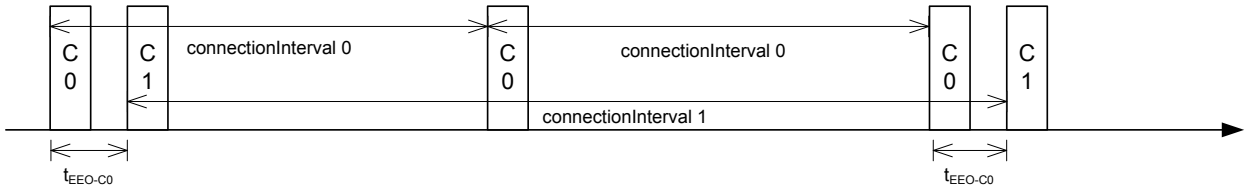

Figure 28: Multilink scheduling - one or more connections as a central, factored intervals

In Figure 29: Multilink scheduling - one or more connections as a central, unfactored intervals on page 58 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.

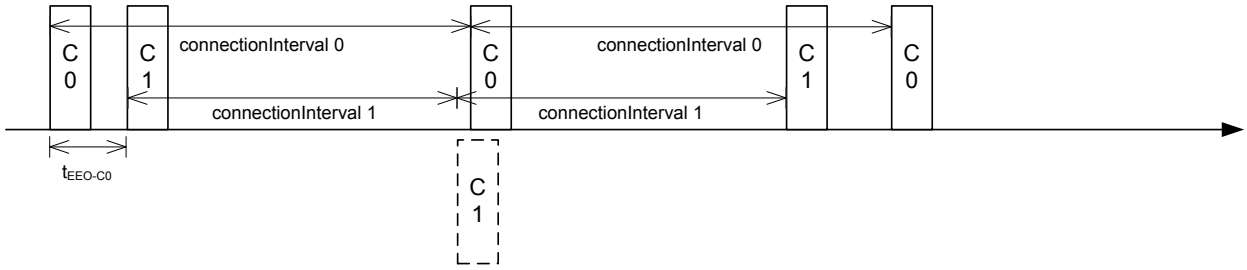

Figure 29: Multilink scheduling - one or more connections as a central, unfactored intervals

Figure 30: Multilink scheduling with maximum connections as a central and minimum interval on page 58 shows the maximum possible number of links as a Central (8) at the same time with minimum bandwidth (LOW bandwidth configuration) and with the minimum connection interval (17.5 ms), without having timing-event collisions and dropped packets. In this case, all available time is used for the links as a central.

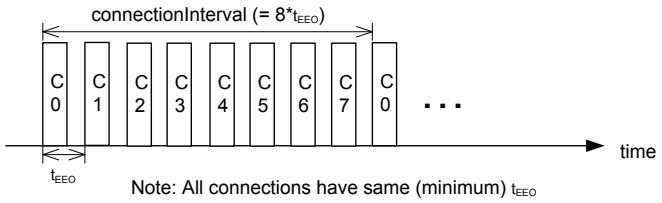

Figure 30: Multilink scheduling with maximum connections as a central and minimum interval

Figure 31: Multilink scheduling of connections as a central and interval > min on page 58 shows a scenario similar to the one illustrated above except the connectionInterval is longer than the minimum, and Central 1 and 4 has been disconnected or does not have an timing-event in this time period. It shows the idle time during connection interval, and also shows that the timings of central link timing-events are not affected if other central links disconnects.

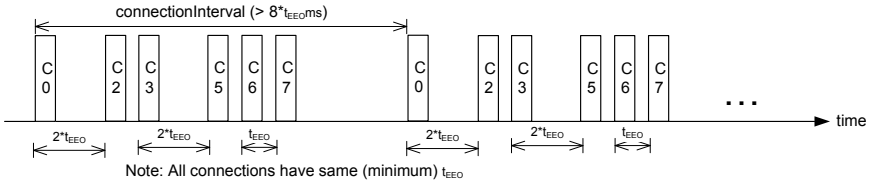

Figure 31: Multilink scheduling of connections as a central and interval > min

16.4 Scanner timing

This section describes scanner timing with different connections.

Figure 32: Scanner timing - no active connections on page 59 shows that when scanning for advertisers with no active connections, the scan interval and window can be any value within the *Bluetooth*® Core Specification.

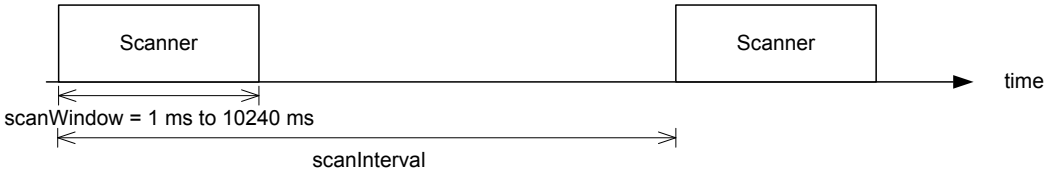

Figure 32: Scanner timing - no active connections

Scanner timing-event is always placed after the Central link timing-events. Figure 33: Scanner timing - one or more connection as a central on page 59 shows that when there is 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 ≤ (connectionInterval - ($\sum t_{EEO}$+ tScanReserved)), scanning will proceed without packet loss.

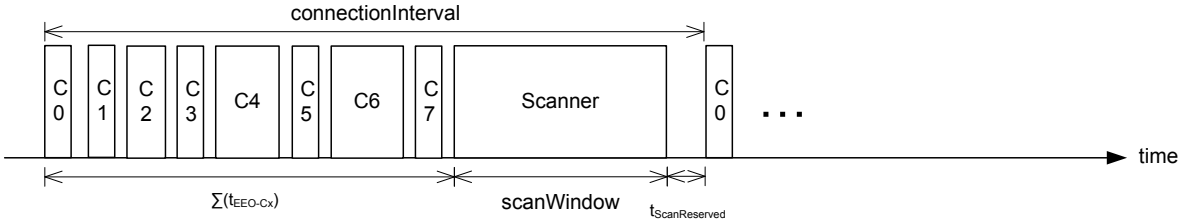

Figure 33: Scanner timing - one or more connection as a central

Figure 34: Scanner timing - always after connections on page 59 shows a scenario where free time is available between link timing-events, but still the scanner timing-event is placed after all connections.

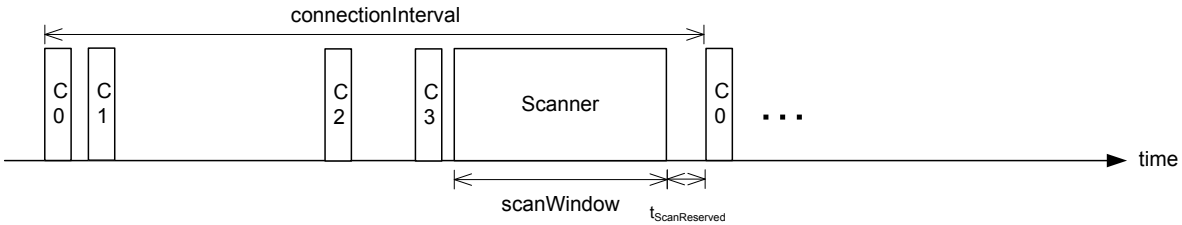

Figure 34: Scanner timing - always after connections

Figure 35: Scanner timing - one connection, long window on page 59 shows a scanner with a long scanWindow which will cause some connection timing-events to be dropped.

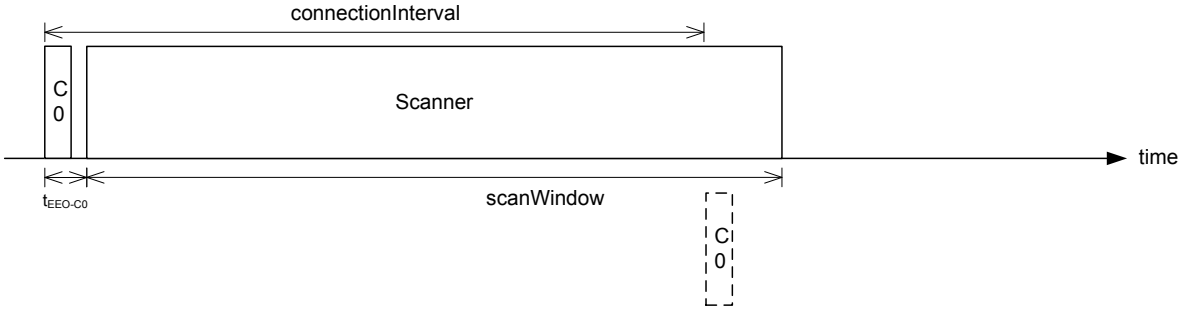

Figure 35: Scanner timing - one connection, long window

16.5 Advertiser (connectable and non-connectable) timing

Advertiser is started as early as possible, asynchronously to any other role timing-events. If no roles are running, advertiser timing-evens are able to start and run without any collision.

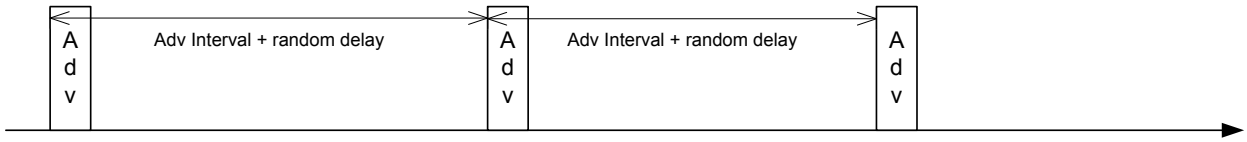

Figure 36: Advertiser

When other role timing-events are running in addition, the advertiser role timing-event may collide with those. Figure 37: Advertiser collide on page 60 shows a scenario of advertiser colliding with peripheral (P).

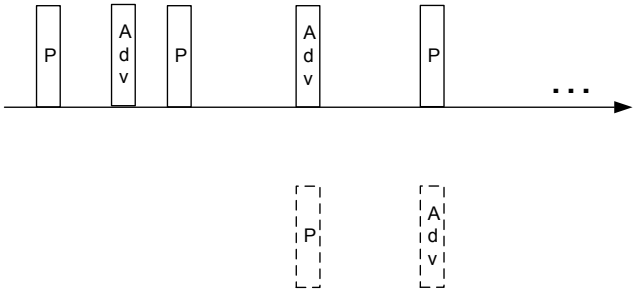

Figure 37: Advertiser collide

Directed advertiser is different compared to other advertiser types because it is not periodic. The scheduling of the single timing-event required by directed advertiser is done in the same way as other advertiser type timing-events. Directed advertiser timing-event is also started as early as possible, and its priority (refer to Table 30: Scheduling priorities on page 54) is raised if it is blocked by other role timing-events multiple times.

16.6 Peripheral connection setup and connection timing

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

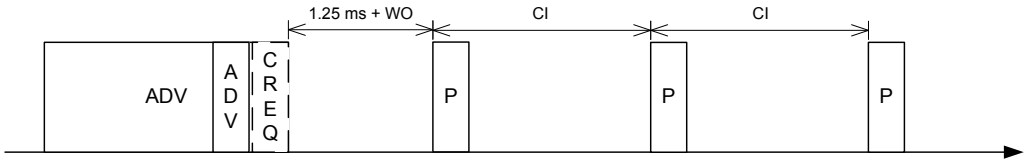

Figure 38: Peripheral connection setup and connection

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

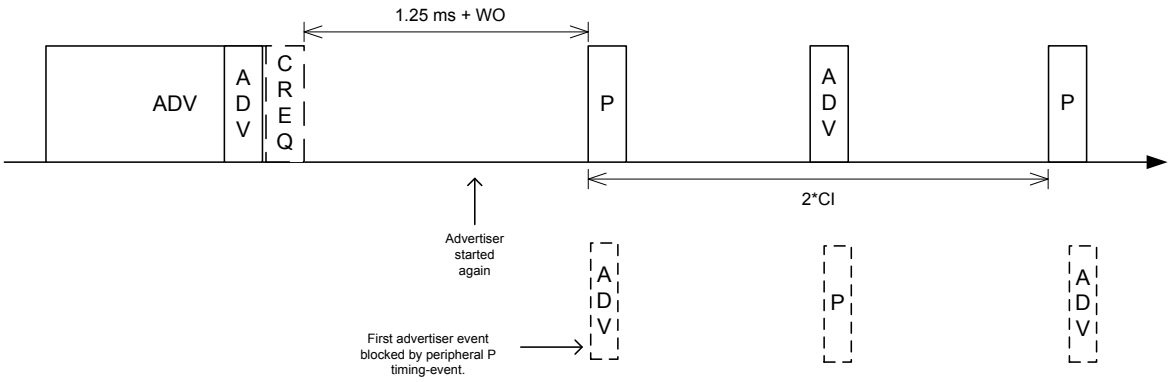

Figure 39: Peripheral connection setup and connection with collision

Table 31: Peripheral role timing ranges

Value	Description	Value (µs)
$t_{SlaveNominalWindow}$	Listening window on slave to receive first packet in a connection event.	1000 (assuming 250 ppm sleep clock accuracy on both slave and master with 1 second connection interval)
$t_{SlaveEventNominal}$	Nominal event length for slave link.	t_{prep}(max) + $t_{SlaveNominalWindow}$ + $t_{event\ (max\ for\ slave\ role)}$ Refer to Table 24: Notation and terminology for the Radio Notification used in this chapter on page 40 and Table 25: BLE Radio Notification timing ranges on page 41.
$t_{SlaveEventMax}$	Maximum event length for slave link.	$t_{SlaveEventNominal}$ + 7 ms Where 7 ms is added for the maximum listening window for 500 ppm sleep clock accuracy on both master and slave with 4-second connection interval.
$t_{AdvEventMax}$	Maximum event length for advertiser (all types except directed advertiser) role.	t_{prep}(max) + $t_{event\ (max\ for\ adv\ role\ except\ directed\ adv)}$ Refer to Table 24: Notation and terminology for the Radio Notification used in this chapter on page 40 and Table 25: BLE Radio Notification timing ranges on page 41.

16.7 Flash API timing

Flash timing-activity is a one time activity with no periodicity, as opposed to BLE 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 lowest possible priority. Other timing-activities running in higher priority can collide with flash timing-events. Refer Table 30: Scheduling priorities on page 54 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 timing-event at all if other timing-events occupy most of the time and use priority higher than flash timing-activity. To avoid long waiting while using Flash API, flash timing-activity will fail in case it cannot get timing-event before a timeout.

16.8 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 Table 30: Scheduling priorities on page 54 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 have two configurable priorities. To run efficiently with other timing-activities, the Timeslot API should run in lowest possible priority. It can be configured to use higher priority if it has been blocked many times by other timing-activities and is in a critical state.

16.9 Suggested intervals and windows

The time required to fit one timing-event of all active central links is equal to the sum of t_{EEO} of all active central links.

Therefore eight link timing-events can complete in maximum $\sum t_{EEO\text{-}Cx}$,which is around 17.5 ms for LOW bandwidth configuration Refer Table 24: Notation and terminology for the Radio Notification used in this chapter on page 40 and Table 25: BLE Radio Notification timing ranges on page 41 for timing ranges in central role.

Note that 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 eight). 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 or equal to $\sum t_{EEO\text{-}Cx}$. Note that this sum depends on number of connections and their respective bandwidth. In the case of eight connections with LOW bandwidth it is 17.5 ms , for 3 connections with MID bandwidth it is 12.5 ms. In case of of using 17.5 ms as the common factor, all connections would have an interval of 17.5 ms or a multiple of 17.5 ms like 35 ms, 53.5 ms, etc.

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 $\sum t_{EEO\text{-}Cx}$ plus the scanWindow plus $t_{ScanReserved}$. Note that the inititator is scheduled asynchronously to any other role (and any other timing-activity), hence initiator timing-event might collide with other timing-events even if above recommendation is followed.

As an example, setting the connection interval to 45 ms will allow three connection events with MID bandwidth 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:

- All central role intervals (i.e. connection interval, scanner/observer/initiator intervals) should have a common factor. This common factor should be ≥ $\sum t_{EEO\text{-}Cx}$ + scanWindow + $t_{ScanReserved}$.

Peripheral roles use the same time space as all other roles (including any other peripheral and central roles), hence a collision free schedule cannot be guaranteed if a peripheral role is running along with any other role. 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 ≥ $\sum t_{EEO\text{-}Cx}$ + ($t_{ScanReserved}$ + ScanWindow) + $t_{SlaveEventNominal}$ + $t_{AdvEventMax}$

  Important: $t_{SlaveEventNominal}$ can be used in above equation in most cases, but should be replaced by $t_{SlaveEventMax}$ for cases where links as a peripheral can have worst sleep clock accuracy and longer connection interval.
- Broadcaster and Advertiser roles also follow the constraint of interval which can be factored by the smallest connection interval.

  Important: Directed advertiser is not considered here because that is not a periodic event.

If only BLE role events are running and the above conditions are met, the worst case collision scenario will be broadcaster, connection as a peripheral, initiator and one or more connection 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 it will be two connections as central if all connections have same bandwidth and both the initiator scan window and the t_{event} for the broadcaster are approximately equal to t_{EEO}. Figure 40: Worst case collission of BLE roles on page 63 shows this case of collision.

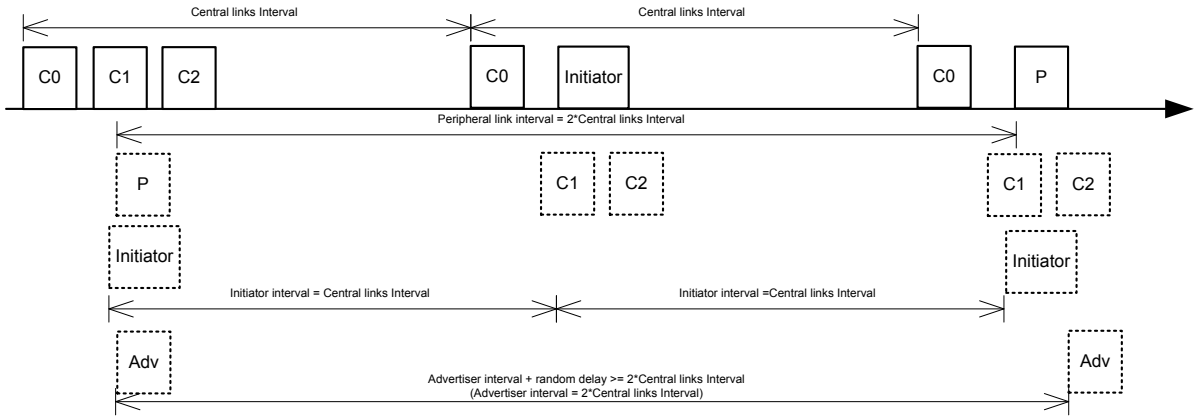

Figure 40: Worst case collission of BLE roles

These collisions will result in collision resolution via priority mechanism. The worst case collision will be reduced if any of the above roles are not running. For example, in the case of only connections as a central and slave connection is running, in the worst case each role will get a timing-event half the time because they both run with the same priority (Refer to Table 30: Scheduling priorities on page 54). Figure 41: Three links running as a central and one peripheral on page 64 shows this case of collision.

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

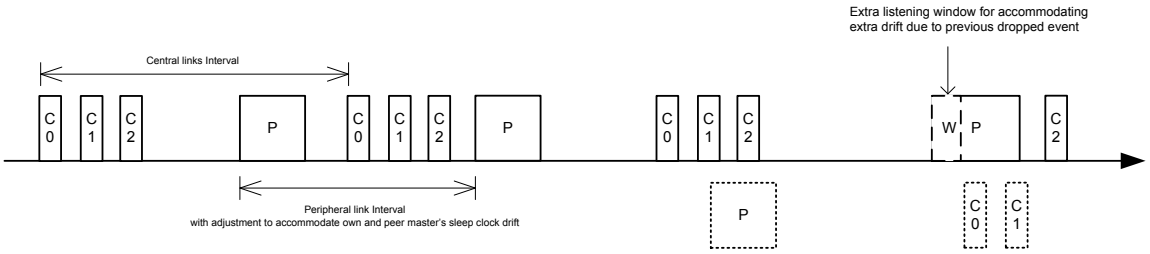

Figure 41: Three links running as a central and one peripheral

Timing-activities other than BLE 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.

Packet drops might happen due to collision between different roles, as it is 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. For example, in case when only 3 central connections and one peripheral connection are running, in the worst case each role will get a timing-event half the time. To accommodate this packet drop, the application should set the supervision time-out to twice the size it would have set if only either Central or Peripheral role was running.

Chapter 17
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.

17.1 Exception model

As the SoftDevice, including the Master Boot Record (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 47.

17.1.1 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 47.

Some peripherals and their respective interrupt numbers are reserved for use by the SoftDevice (see Hardware peripherals on page 16). 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 21: Memory resource map on page 51. 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® -M0 application project with the compiler creating the interrupt vector table.

17.1.2 Interrupt latency due to System on Chip (SoC) framework

Latency, additional to ARM® Cortex® -M0 hardware architecture latency, is introduced by SoftDevice logic to manage interrupt events.

This latency occurs when an interrupt is forwarded to the application from the SoftDevice and is part of the minimum latency for each application interrupt. 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 67.

Table 32: Additional latency due to SoftDevice and MBR forwarding interrupts

Interrupt	SoftDevice enabled	SoftDevice disabled
Open peripheral interrupt	< 4 µs	< 2 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	N/A	< 3 µs
Application SVC interrupt	< 4 µs	< 4 µs

17.2 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 SuperVisor Calls (SVCs, see Application Programming Interface (API) on page 9) 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 interleaved. 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 ARM® Cortex® -M0 processor has four configurable interrupt priorities ranging from 0 to 3 (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 2 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.

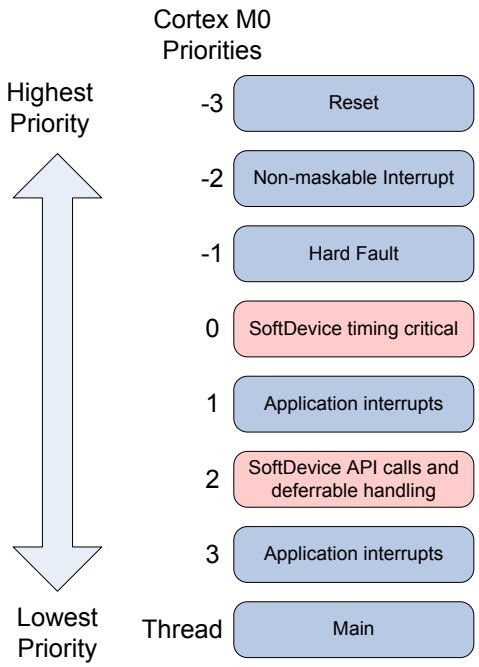

Figure 42: Exception model

As seen from Figure 42: Exception model on page 66, the application has available priority level 1, located between the priority levels reserved by the SoftDevice. This enables a low-latency application interrupt to support fast sensor interfaces. An application interrupt at this priority will only experience latency from SoftDevice interrupts at priority level 0, while application interrupts at priority level 3 can experience latency from both SoftDevice priority levels in addition to its own priority level 1 interrupts.

> **Important:** 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 2 (lower numerical priority value) have neither access to SoftDevice functions nor to application specific SVCs or RTOS functions running at lower priority levels (higher numerical priority values).

Figure 43: SoftDevice exception examples (some priority levels left out for clarity) on page 67 shows an example of how interrupts with different priorities may run and preemt each other.

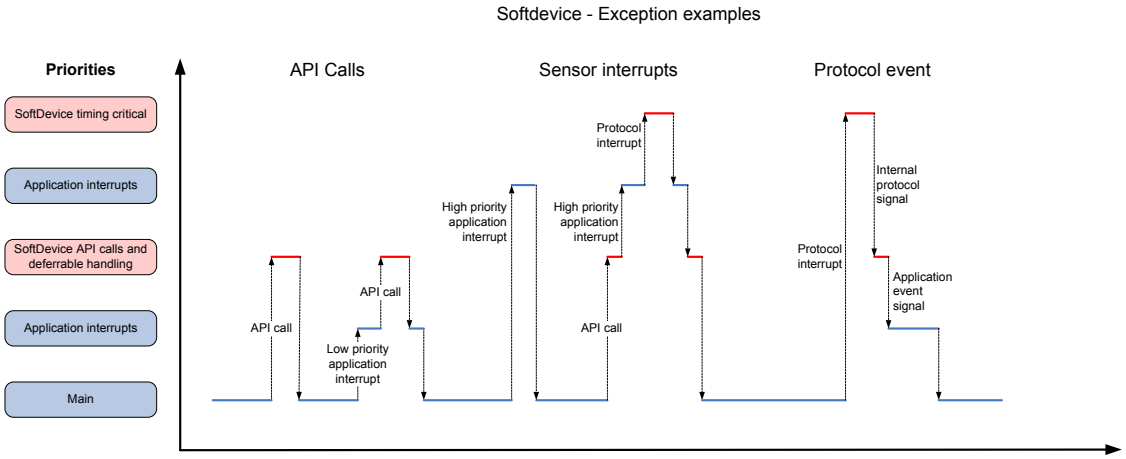

Figure 43: SoftDevice exception examples (some priority levels left out for clarity)

17.3 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 scenarios to follow, $t_{ISR(x)}$ denotes interrupt processing time at priority level x, and $t_{nISR(x)}$ denotes time between interrupts at priority level x.

17.3.1 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.

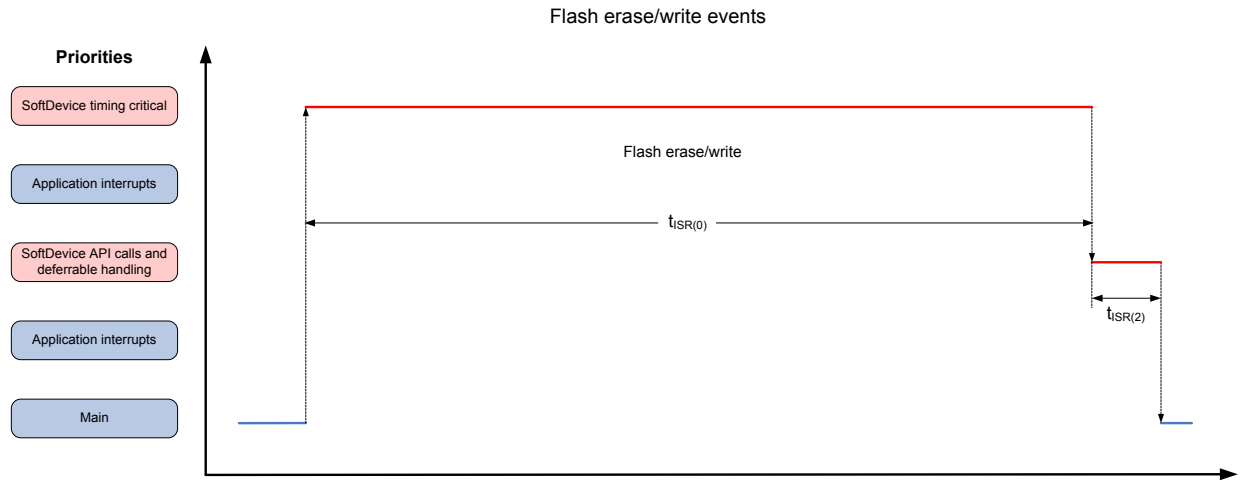

Figure 44: Flash API activity (some priority levels left out for clarity)

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

- First, there is an interrupt at priority level 0 that sets up and performs the flash activity. The CPU is halted for most of the time in this interrupt.
- After the first interrupt is finished, there is another interrupt at priority level 2 that does some cleanup after the flash operation.

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

Table 33: Processor usage for the Flash API

Parameter	Description	Min	Typical	Max
$t_{ISR(0),FlashErase}$	Interrupt processing when erasing a flash page. Note that the CPU is halted for most of the length of this interrupt.			22.6 ms
$t_{ISR(0),FlashWrite}$	Interrupt processing when writing one or more words to flash. Note that the CPU is halted for 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 to find out how long time is needed for per word to write, and add to the Max time provided in this table.			550 µs
$t_{ISR(2)}$	Priority level 2 interrupt at the end of flash write or erase.		30 µs	

17.3.2 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 25 for more information on the Radio Timeslot API.

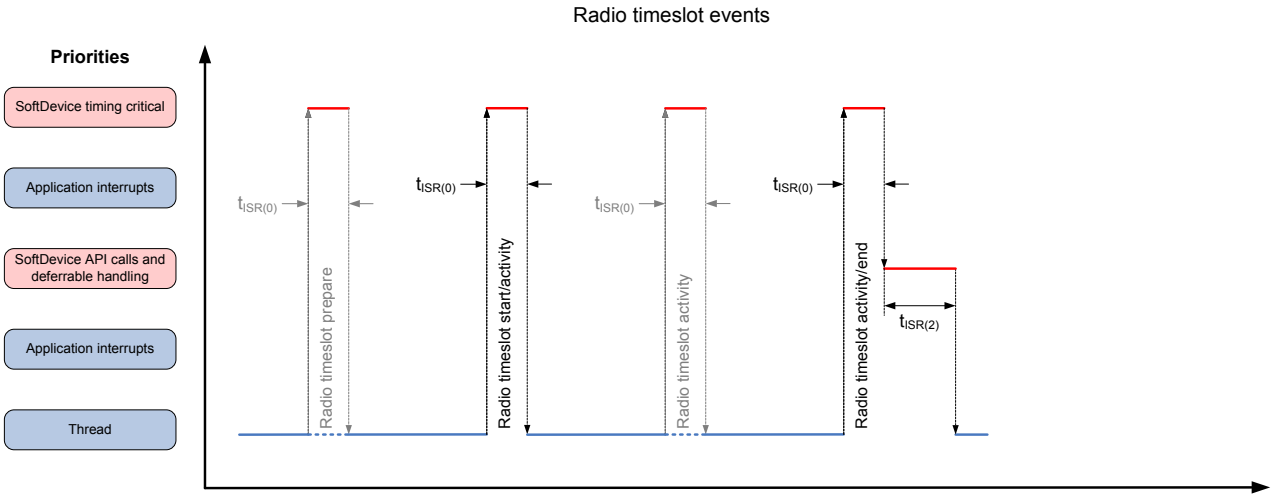

Figure 45: Radio Timeslot API activity (some priority levels left out for clarity)

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

- 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.
- Then there are one or more radio timeslot activities. How many and how long these are, is application dependent.
- When the last of the radio timeslot activities are finished, there is another interrupt at priority level 2 that does some cleanup after the Radio Timeslot operation.

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

Table 34: Processor usage for the Radio Timeslot API

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioTimeslotPrepare}$	Interrupt processing when starting up the high frequency crystal.			21 µs
$t_{ISR(0),RadioTimeslotActivity}$	The application's processing in the timeslot. The length of this is application dependent.			
$t_{ISR(2)}$	Priority level 2 interrupt at the end of the timeslot.		23 µs	

17.3.3 BLE processor usage patterns

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

17.3.3.1 BLE advertiser (broadcaster) processor usage patterns

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

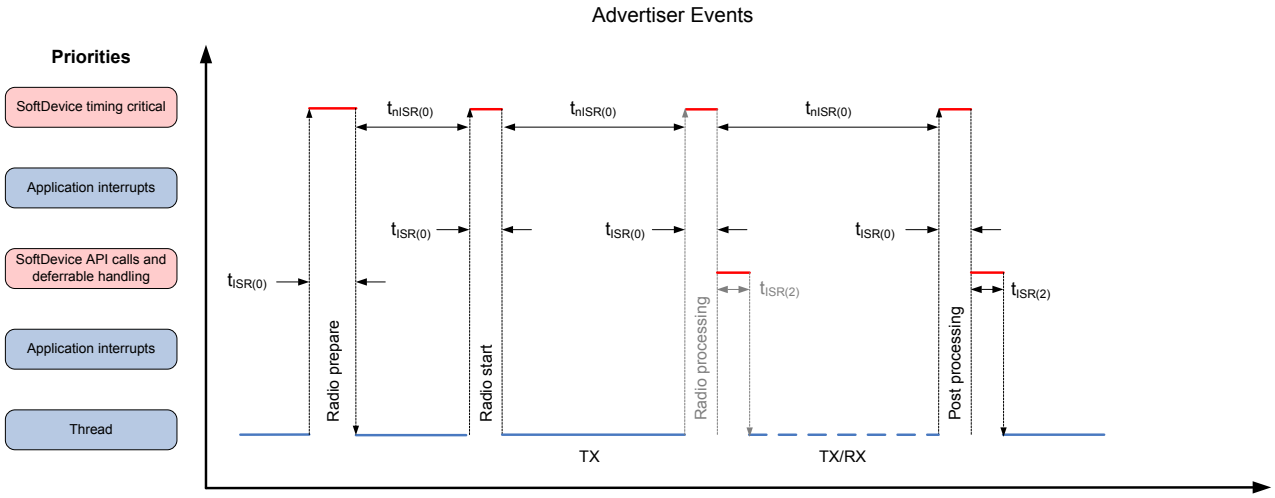

Figure 46: Advertising events

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

- First, there is an interrupt (Radio prepare) that sets up and prepares the software and hardware for this advertising event.
- Then there is a short interrupt when the radio starts sending the first advertising packet.
- Depending on the type of advertising, there may be one or more instances of Radio processing (including processing in priority level 2) and further receptions/transmissions.
- Finally, advertising ends with Post processing at interrupt priority level 0 and some interrupt priority level 2 activity.

SoftDevice processing activity in the different priority levels when advertising is outlined in Table 35: Processor usage when advertising on page 70. The typical case is seen when advertising without using a whitelist and without receiving scan or connect requests. The max case can be seen when advertising with a full whitelist, receiving scan and connect requests while having a maximum number of connections and utilizing the Radio Timeslot API and Flash memory API at the same time.

Table 35: Processor usage when advertising

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing preparing the radio for advertising.		76 µs	130 µs
$t_{ISR(0),RadioStart}$	Processing when starting the advertising.		36 µs	40 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending/receiving a packet.		50 µs	80 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of an advertising event.		200 µs	700 µs
$t_{nISR(0)}$	Distance between interrupts during advertising.	40 µs	>150 µs	
$t_{ISR(2)}$	Priority level 2 interrupt at the end of an advertising event.		270 µs	

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

$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + 2 * t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + t_{ISR(2)} = 682\ \mu s$

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

17.3.3.2 BLE peripheral connection processor usage patterns

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

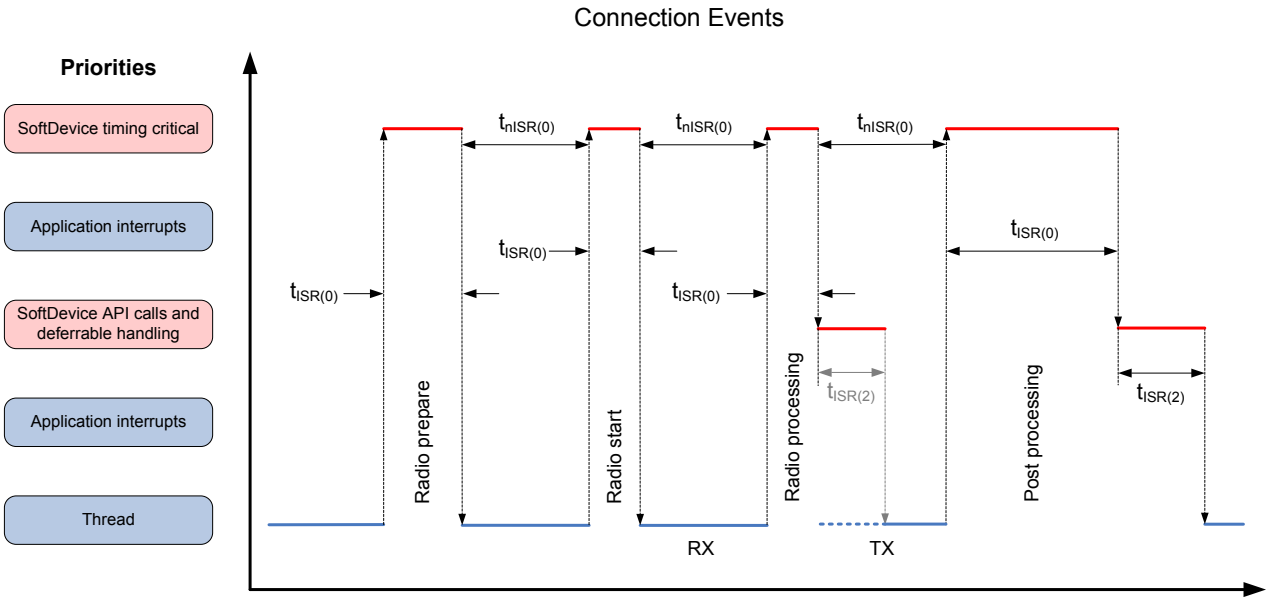

Figure 47: Peripheral connection events

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

- First, there is an interrupt (Radio prepare) that sets up and prepares the software and hardware for the connection event.
- Then there is a short interrupt when the radio starts listening for the first packet.
- When the reception is finished, there is Radio processing that processes the received packet and switches the radio to transmission.
- When the transmission is finished, there is either a Radio processing including a switch back to reception and possibly a new transmission after that, or the event ends with post processing.
- After the radio and post processings in priority level 0, 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 2. 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 peripheral connection events is outlined in Table 36: Processor usage when connected on page 71. The typical case is seen when sending GATT write commands writing 20 bytes. The max case can be seen when sending and receiving maximum length packets and at the same time initiating encryption, while having a maximum number of connections and utilizing the Radio Timeslot API and Flash memory API at the same time.

Table 36: Processor usage when connected

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing preparing the radio for a connection event.		105 µs	130 µs

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioStart}$	Processing when starting the connection event.		44 µs	50 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending or receiving a packet.		80 µs	110 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of a connection event.		255 µs	775 µs
$t_{nISR(0)}$	Distance between interrupts during a connection event.	30 µs	> 160 µs	
$t_{ISR(2)}$	Priority level 2 interrupt after a packet is sent or received.		160 µs	

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

$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + 2 * t_{ISR(2)} = 804$ µs

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

17.3.3.3 BLE scanner and initiator processor usage patterns

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

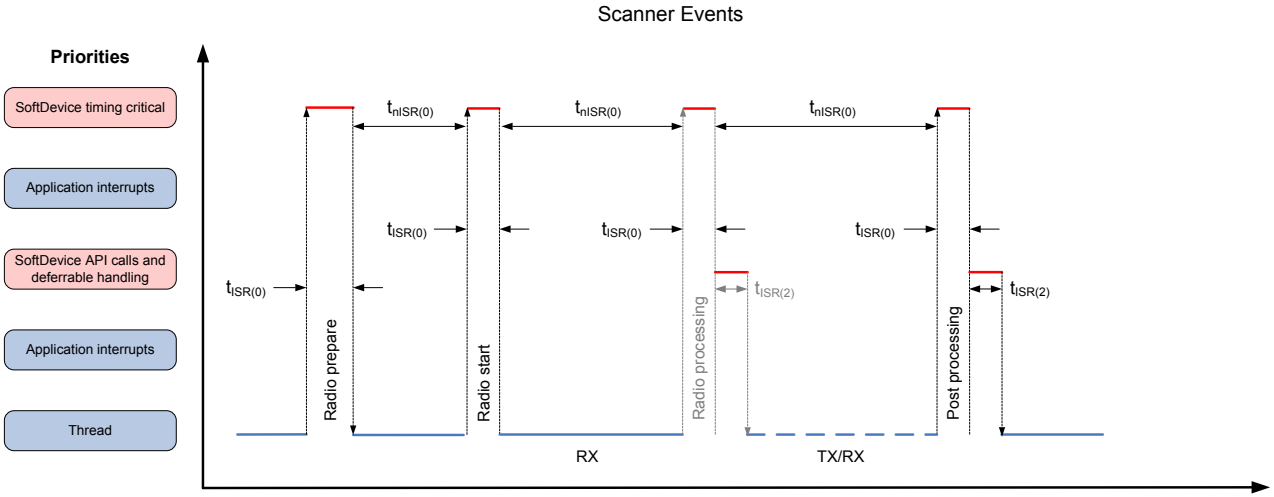

Figure 48: Scanning or initiating

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

- First, there is an interrupt (Radio prepare) that sets up and prepares the software and hardware for this scanner or initiator event.
- Then there is a short interrupt when the radio starts listening for advertisement packets.
- During scanning, there will be zero or more instances of Radio processing, depending upon whether the active role is a scanner or an initator, 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 2.
- When the event ends (either by timeout, or if the initator receives a connectable advertisement packet it accepts), the SoftDevice does some Post processing, which may be followed by processing at interrupt priority level 2.

SoftDevice processing activity in the different priority levels when scanning or initiating is outlined in Table 37: 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, sending scan or connect requests while having a maximum number of connections and utilizing the Radio Timeslot API and Flash memory API at the same time.

Table 37: Processor usage for scanning or initiating

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing preparing the radio for scanning or initiating.		55 µs	128 µs
$t_{ISR(0),RadioStart}$	Processing when starting the scan or initiation.		60 µs	80 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending/receiving packet.		82 µs	145 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of a scanner or initiator event.		186 µs	620 µs
$t_{nISR(0)}$	Distance between interrupts during scanning.	30 µs	>1.5 ms	
$t_{ISR(2)}$	Priority level 2 interrupt at the end of a scanner or initiator event.		250 µs	

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

$$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + t_{ISR(2)} = 633\ \mu s$$

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

17.3.3.4 BLE central connection processor usage patterns

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

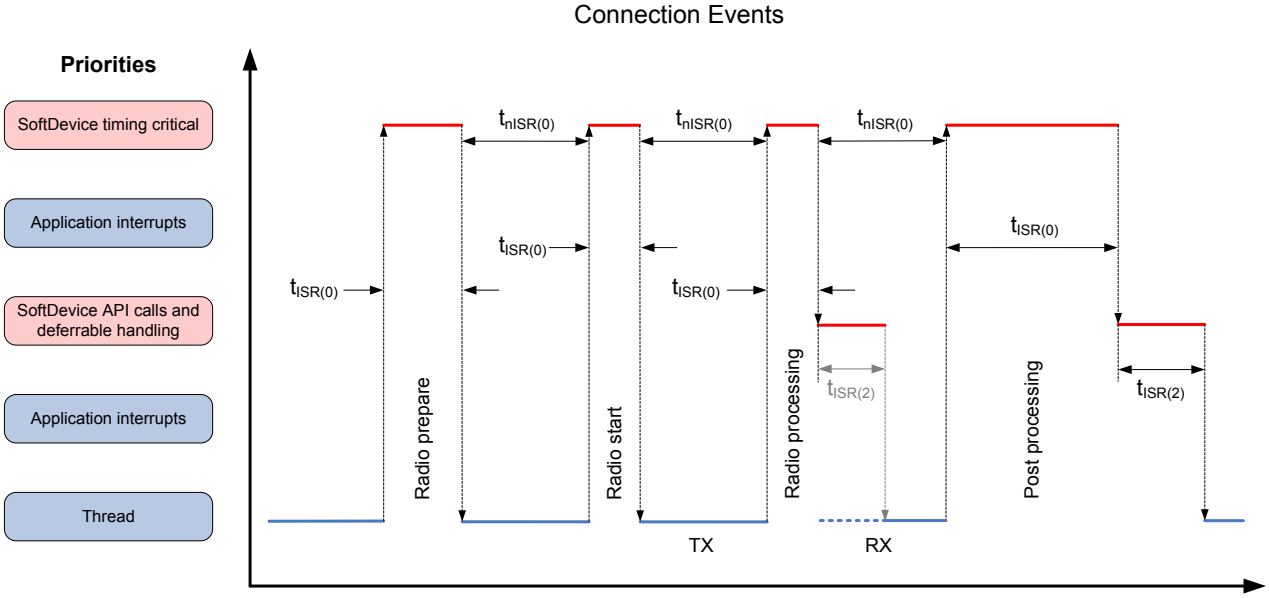

Figure 49: Central connection events

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

- First, there is an interrupt (Radio prepare) that sets up and prepares the software and hardware.
- Then there is a short interrupt when the radio starts transmitting the first packet in the connection event.
- When the transmission is finished, there is Radio processing that switches the radio to reception.
- When the reception is finished, there is a Radio processing that processes the received packet, and either including a switch back to transmission and possibly a new reception after that, or the event ends with post processing.
- After the radio and post processings in priority level 0, 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 2. 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 38: Processor usage latency when connected on page 74. 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 at the same time initiating encryption, while having a maximum number of connections and utilizing the Radio Timeslot API and Flash memory API at the same time.

Table 38: Processor usage latency when connected

Parameter	Description	Min	Typical	Max
$t_{ISR(0),RadioPrepare}$	Processing preparing the radio for a connection event.		96 µs	120 µs
$t_{ISR(0),RadioStart}$	Processing when starting the connection event.		61 µs	70 µs
$t_{ISR(0),RadioProcessing}$	Processing after sending or receiving a packet.		80 µs	150 µs
$t_{ISR(0),PostProcessing}$	Processing at the end of a connection event.		270 µs	730 µs
$t_{nISR(0)}$	Distance between connection event interrupts.	30 µs	> 155 µs	
$t_{ISR(2)}$	Priority level 2 interrupt after a packet is sent or received.		160 µs	

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

$t_{ISR(0),RadioPrepare} + t_{ISR(0),RadioStart} + t_{ISR(0),RadioProcessing} + t_{ISR(0),PostProcessing} + 2 * t_{ISR(2)} = 827$ µs

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

17.3.4 Interrupt latency when using multiple modules and roles

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

The same interrupt priority levels are used by all Flash API, Radio Timeslot API and BLE roles. When using more than one of these concurrently, their respective events can be scheduled back-to-back (see Scheduling on page 24 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 combination of roles that are used.

Example: If the application uses the Radio Timeslot API while having a BLE advertiser running, the worst case interrupt latency or interruption for an application interrupt is the largest of

- The worst case interrupt latency of the Radio Timeslot API
- The worst case interrupt latency of the BLE advertiser role
- The sum of the max time of the first interrupt of the Radio Timeslot API and the last interrupt of the BLE advertiser role
- The sum of the max time of the first interrupt of the BLE advertiser role and the last interrupt of the Radio Timeslot API

for the SoftDevice interrupts with higher priority level (lower numerical value) as the application interrupt.

Chapter 18
BLE data throughput

This chapter gives an indication of achievable BLE connection throughput for GATT procedures used to send and receive data in stated SoftDevice configurations.

Maximum throughput will only be possible when the application reads data packets as they are received, and provides new data as packets are transmitted, without delay. The SoftDevice may transfer more packets than reserved by the bandwidth configuration when data transfer is simplex (read or write only), because extra time is available in the event to transfer data.

All data throughput values apply to packet transfers over an encrypted connection using maximum payload sizes.

Table 39: Maximum data throughput with a single Peripheral or Central connection and a connection interval of 7.5 ms

Protocol	BW Config	Method	Maximum data throughput
GATT Client	HIGH	Receive Notification	149.2 kbps
		Send Write command	149.2 kbps
		Send Write request	10.6 kbps
		Simultaneous receive Notification and send Write command	127.9 kbps (each direction)
GATT Server	HIGH	Send Notification	149.2 kbps
		Receive Write command	149.2 kbps
		Receive Write request	10.6 kbps
		Simultaneous send Notification and receive Write command	127.9 kbps (each direction)
GATT Client	MID	Receive Notification	63.9 kbps
		Send Write command	63.9 kbps
		Send Write request	10.6 kbps
		Simultaneous receive Notification and send Write command	63.9 kbps (each direction)
GATT Server	MID	Send Notification	63.9 kbps
		Receive Write command	63.9 kbps
		Receive Write request	10.6 kbps
		Simultaneous send Notification and receive Write command	63.9 kbps (each direction)

Protocol	BW Config	Method	Maximum data throughput
GATT Client	LOW	Receive Notification	21.3 kbps
		Send Write command	21.3 kbps
		Send Write request	10.6 kbps
		Simultaneous receive Notification and send Write command	21.3 kbps (each direction)
GATT Server	LOW	Send Notification	21.3 kbps
		Receive Write command	21.3 kbps
		Receive Write request	10.6 kbps
		Simultaneous send Notification and receive Write command	21.3 kbps (each direction)

The following table Table 40: Maximum data throughput for each connection, up to 8 connections on page 77 shows maximum data throughput at a connection interval of 17.5 ms that allows up to 8 LOW bandwidth concurrent connections per interval.

Only throughput for LOW bandwidth configuration is indicated. For higher bandwidth configurations, a longer connection interval would need to be used for each connection to prevent connection events from overlapping. See Scheduling on page 54 for more information on how connections can be configured.

Throughput may get reduced if a peripheral link is running because peripheral links are not synchronized with central links. If a peripheral link is running, throughput may decrease to half for up to two central links and the peripheral link.

Table 40: Maximum data throughput for each connection, up to 8 connections

Protocol	BW Config	Method	Maximum data throughput
GATT - Client	LOW	Receive Notification	9.1 kbps
		Send Write command	9.1 kbps
		Send Write request	4.6 kbps
		Simultaneous receive Notification and send Write command	9.1 kbps (each direction)
GATT - Server	LOW	Send Notification	9.1 kbps
		Receive Write command	9.1 kbps
		Receive Write request	4.5 kbps
		Simultaneous send Notification and receive Write command	9.1 kbps (each direction)

Important: 1 kbps = 1000 bits per second

Chapter 19
BLE power profiles

Power profiles give a detailed overview of the stages within a *Bluetooth*® low energy Radio Event, the approximate timing of stages within the event, and how to calculate the peak current at each stage using data from the product specification.

This section provides power profiles for MCU activity during *Bluetooth*® low energy Radio Events implemented in the SoftDevice.

The CPU profile for interrupts at priority level 0 during the event is shown separately. These profiles are based on typical events with empty packets.

19.1 Advertising event

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

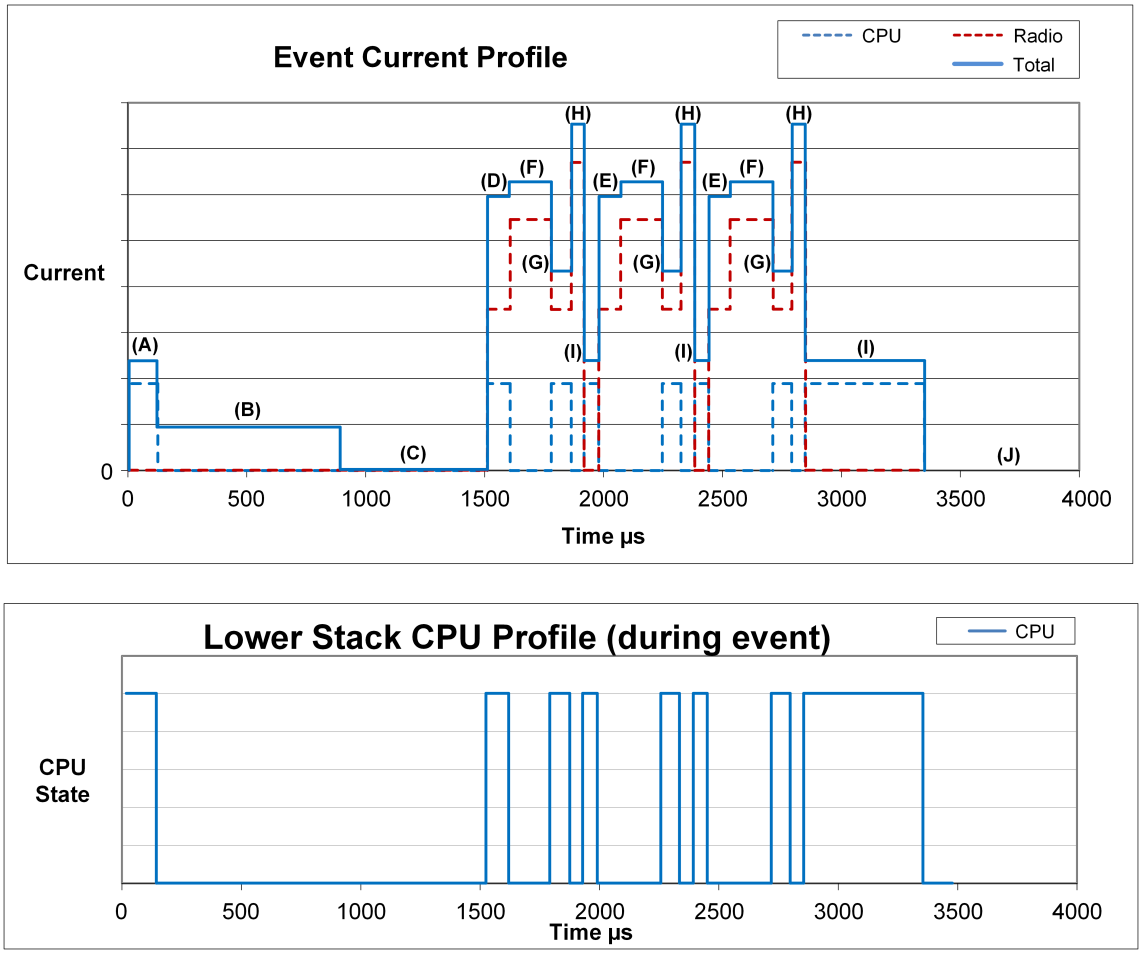

Figure 50: Advertising event

Table 41: Advertising event

Stage	Description	Current calculation[11]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash} + I_{START,X16M}$

Stage	Description	Current calculation[11]
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start/switch	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#I(_{START,TX}) + I_{CPU,Flash}$
(E)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX})$
(F)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM}$
(G)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(H)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(I)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(J)	Idle	$I_{ON} + I_{RTC} + I_{X32k}$

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

19.2 Peripheral connection event

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

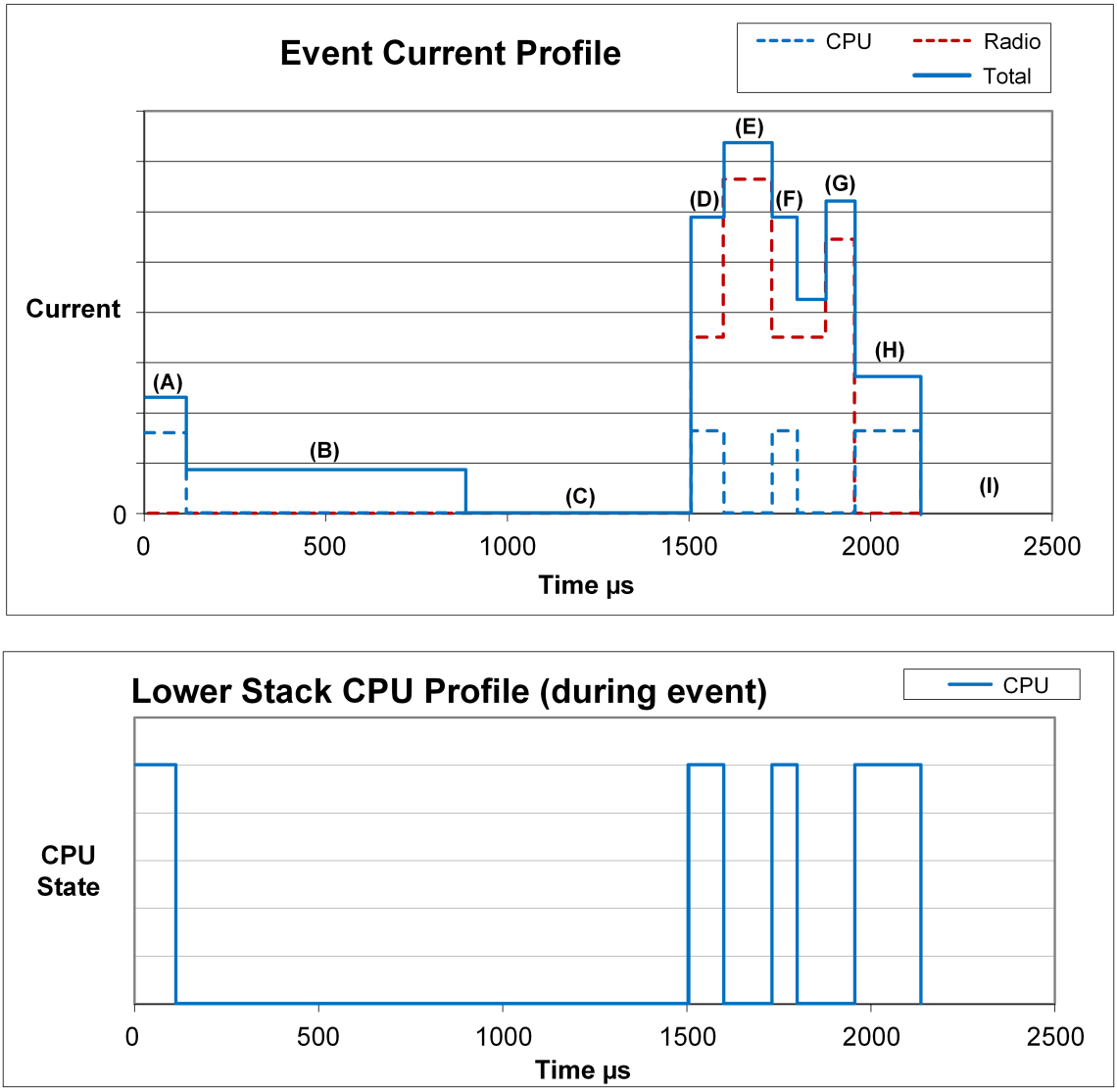

Figure 51: Peripheral connection event

Table 42: Peripheral connection event

Stage	Description	Current Calculation[12]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash} + I_{START,X16M}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(E)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX} + I_{CRYPTO}$
(F)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX}) + I_{CPU,Flash}$
(G)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM} + I_{CRYPTO}$
(H)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

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

19.3 Scanning event

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

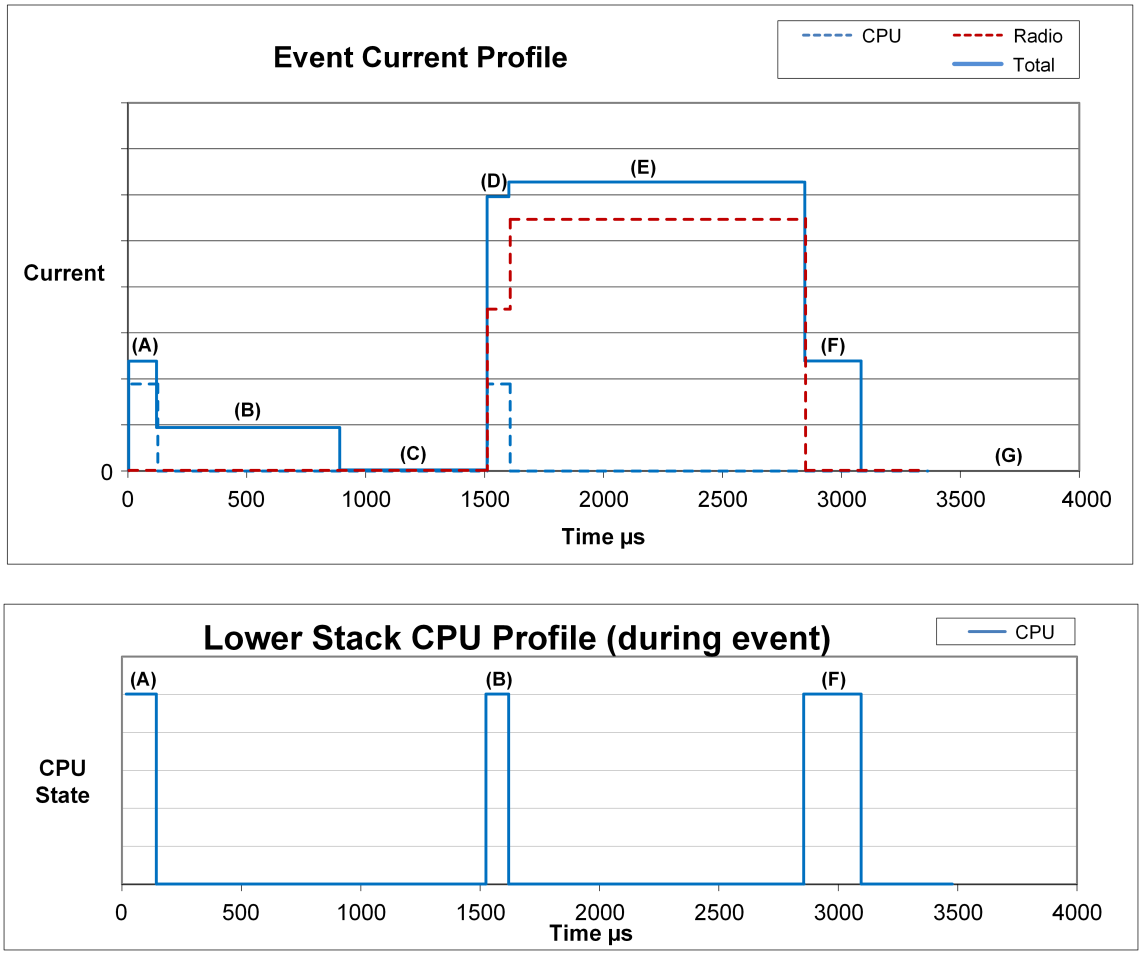

Figure 52: Scanning event

Table 43: Scanning event

Stage	Description	Current Calculation[13]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(E)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(F)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(G)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

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

19.4 Central connection event

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

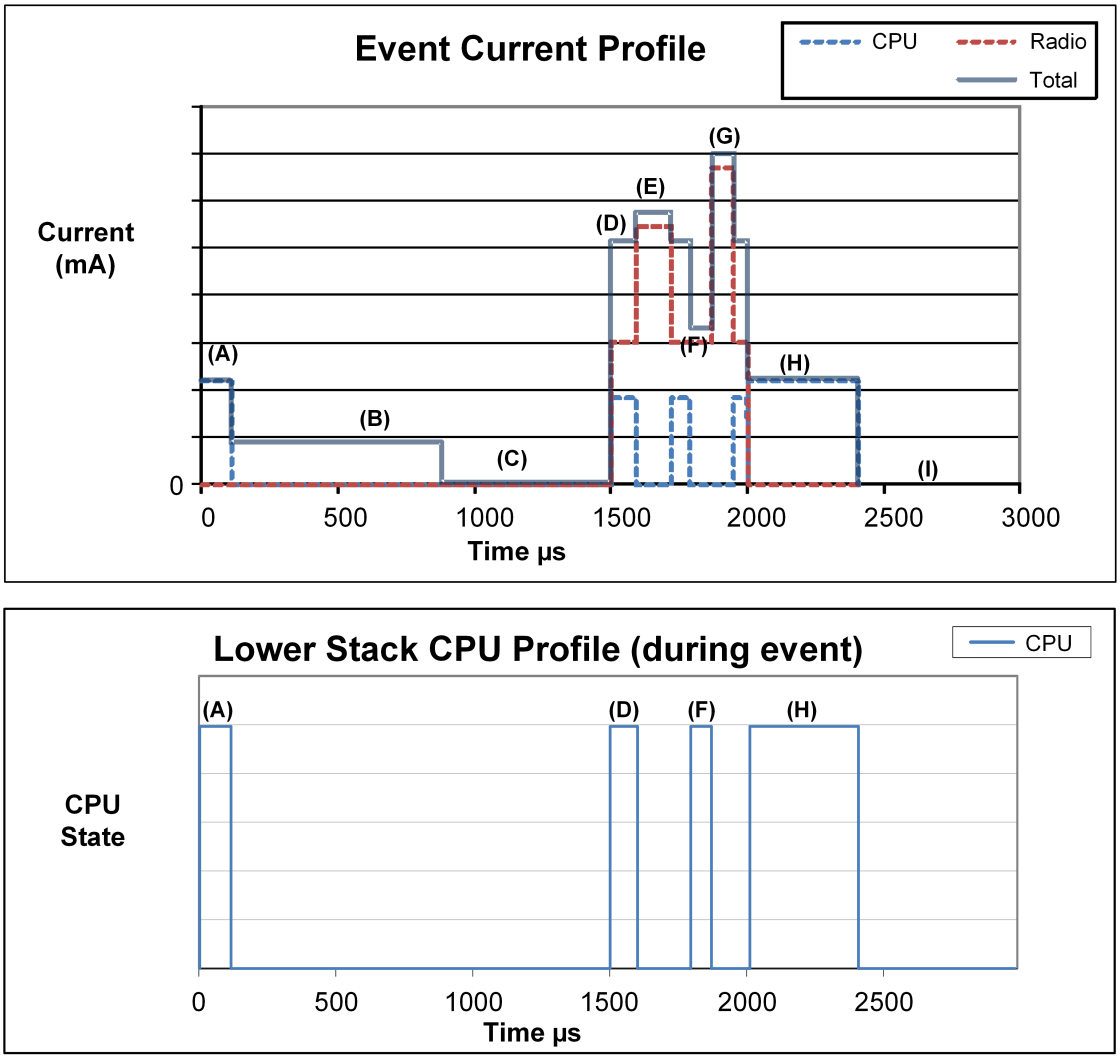

Figure 53: Central connection event

Table 44: Central connection event

Stage	Description	Current Calculation[14]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX}) + I_{CPU,Flash}$
(E)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM}$
(F)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(G)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$

Stage	Description	Current Calculation[14]
(H)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

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

Chapter 20
SoftDevice identification and revision scheme

The SoftDevices are identified by the SoftDevice part code, a qualified IC partcode (for example, nRF51822), 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 major = 1, minor = 2, revision = 3, build number = 4 and test qualification level is alpha. Additional examples are given in table Table 46: SoftDevice revision examples on page 85.

Table 45: Revision scheme

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 46: SoftDevice revision examples

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

Sequence number	Description
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 47: Test qualification levels

Qualification	Description
Alpha	• Development release suitable for prototype application development. • Hardware integration testing is not complete. • Known issues may not be fixed between alpha releases. • Incomplete and subject to change.

Beta	• Development release suitable for application development.
	• In addition to alpha qualification:
	  • Hardware integration testing is complete.
	  • Stable, but may not be feature complete and may contain known issues.
	  • Protocol implementations are tested for conformance and interoperability.

Production	• Qualified release suitable for production integration.
	• In addition to beta qualification:
	  • Hardware integration tested over supported range of operating conditions.
	  • Stable and complete with no known issues.
	  • Protocol implementations conform to standards.

20.1 MBR distribution and revision scheme

The MBR is distributed in each SoftDevice hex file.

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