™

SoftDevice Specification

S212 SoftDevice v2.0

Copyright Information and Usage Notice

Revision History

Revision	Effective Date	Description
1.0	June 2016	Initial SDS creation
2.0	September 2016	High Duty Search and Time Sync added

Table of Contents

List of Figures

List of Tables

1　S212 SoftDevice

The S212 SoftDevice is an ANT protocol stack solution. The S212 SoftDevice integrates an ANT Master/Slave stack and provides a complete API that allows up to 15 channels and can be configured into a flexible network solution that covers everything from point-to-point to mesh networks. The S212 SoftDevice is designed for use with the nRF52 System on Chip (SoC) solutions.

Table 1-1. Summary of key features and applications

Key features	Applications
• ANT compliant low energy wireless communications 　• Simple to complex network topologies supported 　• Logical channels individually configurable for channel type, ID, period, RF frequency, and network 　• Broadcast, acknowledged or burst data transfer 　• Device search, pairing, and proximity support 　• 8 byte data payload per message 　• Advanced ANT Features: 　　• Configurable for up to 15 ANT channels 　　• AES-128 data encryption option for data transfers on all ANT channels 　　• Advanced Burst Transfer mode (up to 60 kbps) 　　• Up to 8 public, managed, and/or private ANT network keys 　　• Event Filtering and Selective Data Updates 　　• Asynchronous Transmission 　　• Fast Channel Initiation 　　• Search Uplink 　　• Group Transmitter Initiation 　　• High Duty Search 　　• Time Sync • Master Boot Record for over-the-air device firmware update 　• ANT updater available 　• 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 behaviour • No RTOS dependency 　• Any RTOS can be used • No link-time dependencies 　• Standard ARM® Cortex®-M4 project configuration for application development • Support for concurrent and non-concurrent multiprotocol operation 　• Concurrent with the ANT stack using the Radio Timeslot API 　• Alternate protocol stack in application space • Support for control of external Power Amplifiers and Low Noise Amplifiers	• Sports and fitness devices 　• Sports watches 　• Bike computers 　• Wearables • Personal Area Networks 　• Health and fitness sensor and monitoring devices 　• Medical devices 　• Key fobs and wrist watches • Home automation • Remote control toys • Computer peripherals and I/O devices 　• Mice 　• Keyboards 　• Multi-touch trackpads • Interactive entertainment devices 　• Remote controls 　• Gaming controllers • Environment sensor networks • High density networking and monitoring • Logistics and goods tracking • Smart RF tags • Internet of Things

2   Documentation

Additional recommended reading for developing applications using the SoftDevice on the nRF52 SoC is listed in Table 2-1. These documents can be downloaded from www.thisisant.com, and www.infocenter.nordicsemi.com.

Table 2-1. S212 SoftDevice core documentation

Documentation	Description
nRF52832 Product Specification	Contains a description of the hardware, peripherals, and electrical specifications specific to the nRF52832 IC.
nRF52832 Errata	Contains information on anomalies related to the nRF52832 IC.
nRF52 Series Compatibility Matrix	Contains information on the compatibility between nRF52 Integrated Circuit (IC) revisions, SoftDevices and SoftDevice Specifications, SDKs, development kits, documentation, and Qualified Design Identifications (QDIDs).
ANT Message Protocol and Usage	Contains information on ANT serial messages and ANT usage
Nordic S210 SoftDevice Specification v3.0	Contains information on ANT features and APIs for previous ANT SoftDevice

3   Product Overview

The S212 SoftDevice is a precompiled and linked binary image implementing an ANT protocol stack for the nRF52 Series of SoCs.
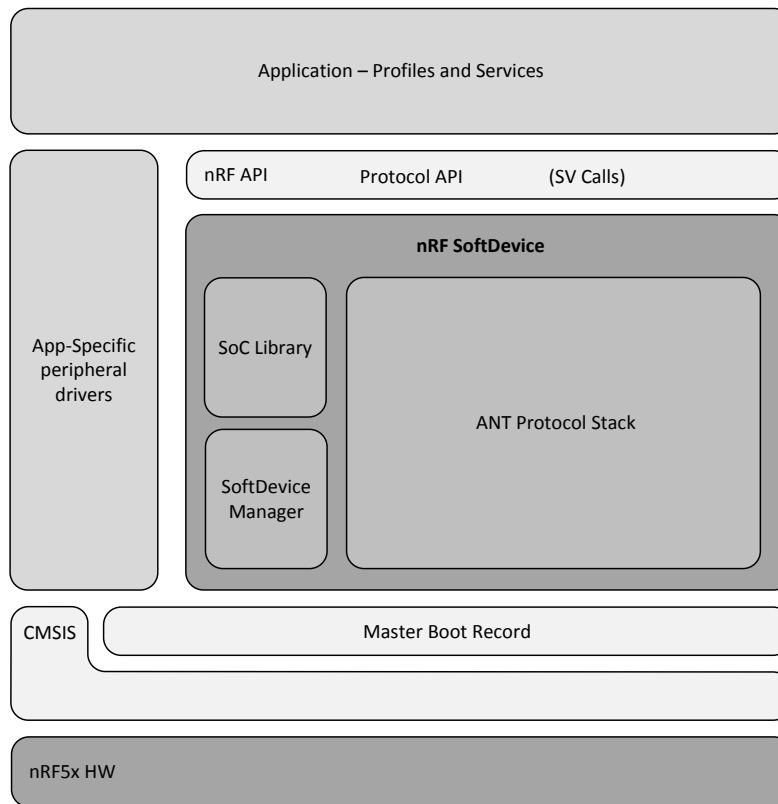
Figure 3-1. System on Chip application with the SoftDevice

Figure 3-1 is a block diagram of the nRF52 series software architecture. It includes the standard ARM® CMSIS interface for nRF52 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 four 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.).

- ANT protocol stack - Implementation of ANT protocol stack and API.

Each Application Programming Interface (API) is a set of standard C language functions and data types, which are provided as a series of header files. These give the application complete compiler and linker independence from the SoftDevice implementation. See section 4 for more details.

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

The SoftDevice can be programmed onto compatible nRF52 Series ICs during both development and production.

4   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 API is available to the application (see S212 SoftDevice v1.0.2 API). For more information about enabling and disabling the SoftDevice, see section 5.1.

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 to 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 section 16.2.

4.1   Events – SoftDevice to application

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

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

4.2   Error handling

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

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

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

- SoftDevice assertions.

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

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

5   SoftDevice Manager

The SoftDevice Manager (SDM) API allows the application to control the SoftDevice state and configure the behaviour of certain SoftDevice core functionality.

When enabling the SoftDevice, the SDM configures the following:

- The low frequency clock (LFCLK) source (section 5.2).

- The interrupt management (section5.1).

- The embedded protocol stack.

In addition, it enables the SoftDevice RAM and peripheral protection. See section 5.4.

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

5.1   SoftDevice enable and disable

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

When the SoftDevice is not enabled, most of the SoCs 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 (section 14.1)

Once the SoftDevice has been enabled, more restrictions apply:

- Some RAM will be reserved (section 5.4)

- Some peripherals will be reserved (section 7.1).

- 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 (section 16.1.1).

- 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 (section 16).

5.1.1   ANT License Key

The S332 SoftDevice requires a license key in order to operate. An evaluation key is included in the SoftDevice which will enable the full stack and is to be used for NON-COMMERCIAL USE ONLY. Further information about the license key and/or obtaining a commercial license key can be found at: www.thisisant.com/developer/ant/licensing.

License validation may cause the enable time of the SoftDevice to extend by up to 100 ms in some configurations.

5.2   Clock source

The SoftDevice can use one of two available low frequency clock sources: the internal RC Oscillator, or an 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 periodically and when a temperature change of more than 0.5°C has occurred, to adjust the RC oscillator frequency. See the nRF52832 Product Specification for more information. The SoftDevice will perform this function automatically. The application may choose how often the SoftDevice will make a measurement to detect temperature change based on how frequently significant temperature changes are expected to occur in the intended environment of the end product. A temperature polling interval of 4 seconds and a forced clock calibration every second interval (8 seconds) is recommended (`.ctiv=32, .temp_ctiv=2`).

5.3   Power management

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

The application shall 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.

When waiting for application events using the Power API, the CPU goes to an IDLE state whenever the SoftDevice is not using the CPU. 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 Power API ensures the SoftDevice services are stopped before powering down.

5.4   Memory isolation and runtime protection

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

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

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

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

The RAM is also 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.

---

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

Figure 5-1 presents an overview of the regions.

Program Memory

RAM

0x0000 0000 +
<size of flash>

Application
Flash Region

APP_CODE_BASE

SoftDevice
Flash Region

0x0000 0000

0x2000 0000 +
<size of RAM>

Application
RAM Region

APP_RAM_BASE

SoftDevice
RAM Region

0x2000 0000
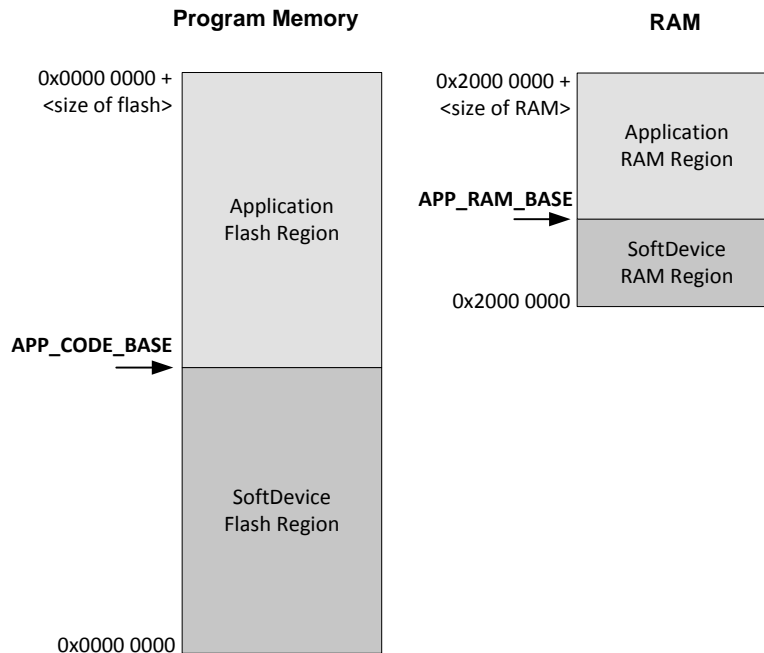
Figure 5-1. Memory region designation

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.

The amount of flash and RAM available to the application is determined by region size (kilobytes or bytes) and the base addresses of the application code and RAM: `APP_CODE_BASE` and `APP_RAM_BASE` 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 of an application program's 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 14-1:

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

Sandboxing protects the SoftDevice RAM Region so that it cannot be written to by the application at runtime. Violation of sandboxing rules, for example an attempt to write to the protected SoftDevice memory, will result in the triggering of a fault (unrecoverable error handled by the application). See section 4.2 for more information.

When the SoftDevice is disabled, all RAM, with the exception of a few bytes, is available to the application. See section 14.1 for more details. When the SoftDevice is enabled, RAM up to `APP_RAM_BASE` will be used by the SoftDevice and will be write protected.

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

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.

6   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 6-1 are implemented by the SoC Library and can be used for accessing the shared hardware resources.

Table 6-1. 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.
Radio Timeslot API	Schedule other radio protocol activity, or periods of radio inactivity. See section 9.2.
Radio Notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See section 11.1.
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 section 8.
Temperature	Application access to the temperature sensor.

7   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 (section 12) are designed to be installed on the nRF SoC in the lower part of the code memory space. After a reset, the MBR will use some RAM to store state information. When the SoftDevice is enabled, it uses resources on the SoC including RAM and hardware peripherals like the radio. For the amount of RAM required by the SoftDevice see section 14.

7.1   Hardware peripherals

Hardware 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 7-1. Hardware access type definitions

Access type	Description
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 7-2. 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	BPROT	Restricted	Open
1	0x40001000	RADIO	Blocked[2]	Open
2	0x40002000	UART0 / UARTE0	Open	Open
3	0x40003000	TWIM0 / TWIS0 / SPIM0 / SPIS0 / SPI0 /TWI0	Open	Open

---

2 Available to the application through the Radio Timeslot API, see section 9.2.

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
4	0x40004000	SPI1 / TWIS1 / SPIM1 / TWI1 / TWIM1 / SPIS1	Open	Open
…				
6	0x40006000	GPIOTE	Open	Open
7	0x40007000	SAADC	Open	Open
8	0x40008000	TIMER0	Blocked[3]	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[4]	Open
15	0x4000F000	AAR	Blocked[5]	Open
16	0x40010000	WDT	Open	Open
17	0x40011000	RTC1	Open	Open
18	0x40012000	QDEC	Open	Open
19	0x40013000	LPCOMP / COMP	Open	Open
20	0x40014000	EGU0 / SWI0	Open	Open

[3] Available to the application through the Radio Timeslot API, see section 9.2.

[4] Available to the application through the Radio Timeslot API, see section 9.2.

[5] Available to the application through the Radio Timeslot API, see section 9.2.

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
21	0x40015000	EGU1 / SWI1 / Radio Notifications	Restricted[6]	Open
22	0x40016000	EGU2 / SWI2 / SoftDevice Event	Blocked	Open
23	0x40017000	EGU3 / SWI3	Open	Open
24	0x40018000	EGU4 / SWI4	Blocked	Open
25	0x40019000	EGU5 / SWI5	Blocked	Open
		…		
30	0x4001E000	NVMC	Restricted	Open
31	0x4001F000	PPI	Open[7]	Open
32	0x40020000	MWU	Restricted[8]	Open
33	0x40021000	PWM1	Open	Open
34	0x40022000	PWM2	Open	Open
35	0x40023000	SPI2 / SPIS2 / SPIM2	Open	Open
36	0x40024000	RTC2	Open	Open
37	0x40025000	I2S	Open	Open
38	0x40026000	FPU	Open	Open
NA	0x10000000	FICR	Blocked	Blocked
NA	0x10001000	UICR	Restricted	Open
NA	0x50000000	GPIO P0	Open	Open

---

6 Blocked only when Radio Notification signal is enabled. See section 7.2 for software interrupt allocation.

7 See section 7.3 for limitations on the use of PPI when the SoftDevice is enabled.

8 See section 5.4 and section 7.5for limitations on the use of MWU when the SoftDevice is enabled.

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
NA	0xE000E100	NVIC	Restricted[9]	Open

7.2   Application signals – Software Interrupts (SWI)

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

Table 7-3. 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.

---

9 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.

7.3   Programmable Peripheral Interconnect (PPI)

The Programmable Peripheral Interconnect 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 shall 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 7-4. Assigning PPI channels between the application and SoftDevice

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 − 16	Channels 0 − 19
SoftDevice	Channels 17 − 19[10]	-

Table 7-5. Assigning preprogrammed channels between the application and SoftDevice

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

Table 7-6. Assigning PPI groups between the application and SoftDevice

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 − 3	Channels 0 − 5
SoftDevice	Channels 4 − 5	-

[10] Available to the application in Radio Timeslot API timeslots (section 9.2)

7.4 SVC number ranges

Application programs and SoftDevices use certain SVC numbers.

The table below shows which SVC numbers an application program can use and which numbers are used by the SoftDevice.

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

Table 7-7. SVC number allocation

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	0x00-0x0F	0x00-0x0F
SoftDevice	0x10-0xFF	0x10-0xFF

7.5 Peripheral runtime protection

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

Protected peripheral registers are readable by the application. An attempt to perform a write to a protected peripheral register will result in a Hard Fault. See section 4.2 for more details on faults due to prohibited memory access. Note that the peripherals are only protected when the SoftDevice is enabled; otherwise they are available to the application. See Table 7-2 for an overview of the peripherals with access restrictions due to the SoftDevice.

7.6 External and miscellaneous requirements

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

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 for the SoC (nRF52832 Product Specification).

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

8   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 ANT communication using the Flash Memory API of the SoC Library.

The flash memory accesses are scheduled to not disturb radio events. See section 15.2 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 timeout 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.

Important: Flash page (4096 bytes) erase can take up to 90ms and a 4 byte flash write can take up to 338µs. A flash write must be made in chunks smaller than or equal to the flash page size. Make flash writes in the smallest chunks possible to increase the probability of success, and reduce the chances of affecting ANT performance.

8.1   Using flash with ANT activity

The SoC Library API can safely be used during ANT radio activities to perform asynchronous flash memory operations.

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

Table 8-1. Behaviour with ANT traffic and concurrent flash write/erase

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

9 Multiprotocol support

Multiprotocol support allows a developer to implement a custom 2.4 GHz proprietary protocol in the application; both while the SoftDevice is not in use (non-concurrent), and while the SoftDevice protocol stack is in use (concurrent). For concurrent multiprotocol implementations, the Radio Timeslot API allows the application protocol to safely schedule radio usage between ANT events.

9.1 Non-concurrent multiprotocol implementation

For non-concurrent operation a proprietary 2.4 GHz protocol can be implemented in the application program area and can access all hardware resources when the SoftDevice is disabled. The SoftDevice may be disabled and enabled without resetting the application in order to switch between a proprietary protocol stack and ANT communication.

9.2 Concurrent multiprotocol implementation using the Radio Timeslot API

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

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

The Timeslot feature is part of the SoC Library. The feature works by having the SoftDevice time-multiplex access to peripherals between the application and itself. Through the SoC Library 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 7-1 and Table 7-2.

9.2.1 Request types

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

Timeslots may be requested as *earliest possible*, in which case the timeslot occurs at the first available opportunity. In the request, the application can limit how far into the future the timeslot may be placed.

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 an extension to an ongoing timeslot. Extension requests may be repeated, prolonging the timeslot even further.

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

9.2.2 Request priorities

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

9.2.3   Timeslot length

A Radio Timeslot is requested for a given length of time. 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 100µs to 100ms. 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 128s.

9.2.4   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.

Section 15.3 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 11.1 for more information. If Radio Notifications are enabled, Radio Timeslots will be notified.

9.2.5   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 section 9.2.4 for more information on how priorities work together with other modules e.g. the ANT protocol stack, the Flash API etc.

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

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

9.2.6   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.

9.2.6.1   API calls

These are the API calls defined for the S212 SoftDevice:

Table 9-1. 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.

9.2.6.2   Radio Timeslot events

Events are generated by 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 section 4.1. The following events are defined:

Table 9-2. Radio Timeslot events

Events	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_CANCELLED	Timeslot status: The scheduled timeslot was cancelled due to an overlapping 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.

9.2.6.3   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 16.2.

Table 9-3. 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 nRF52 Reference Manual.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_TIMER0	Timer interrupt, for more information, see chapter Timer/counter (TIMER) in the nRF52 Reference Manual.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_SUCCEEDED	The most recent extend action succeeded.
NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_FAILED	The most recent extend action failed.

9.2.6.4   Signal handler return actions

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

Table 9-4. 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.

9.2.6.5   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 1MHz. 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.

9.2.6.6   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 (section 16.2), 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 behaviour of the SoftDevice is undefined and the SoftDevice may malfunction.

The signal handler may be called several times during a timeslot. It is recommended that the signal handler is only used 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) while waiting.

9.3   Radio Timeslot API usage scenarios

This section describes several Radio Timeslot API usage scenarios and the sequence of events within them.

9.3.1   Complete session example

This section describes a complete Radio Timeslot session, shown in Figure 9-1. 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 100ms after the first.

All subsequent timeslots are similar (see the middle timeslot in Figure 9-1). 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 9-1), 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.
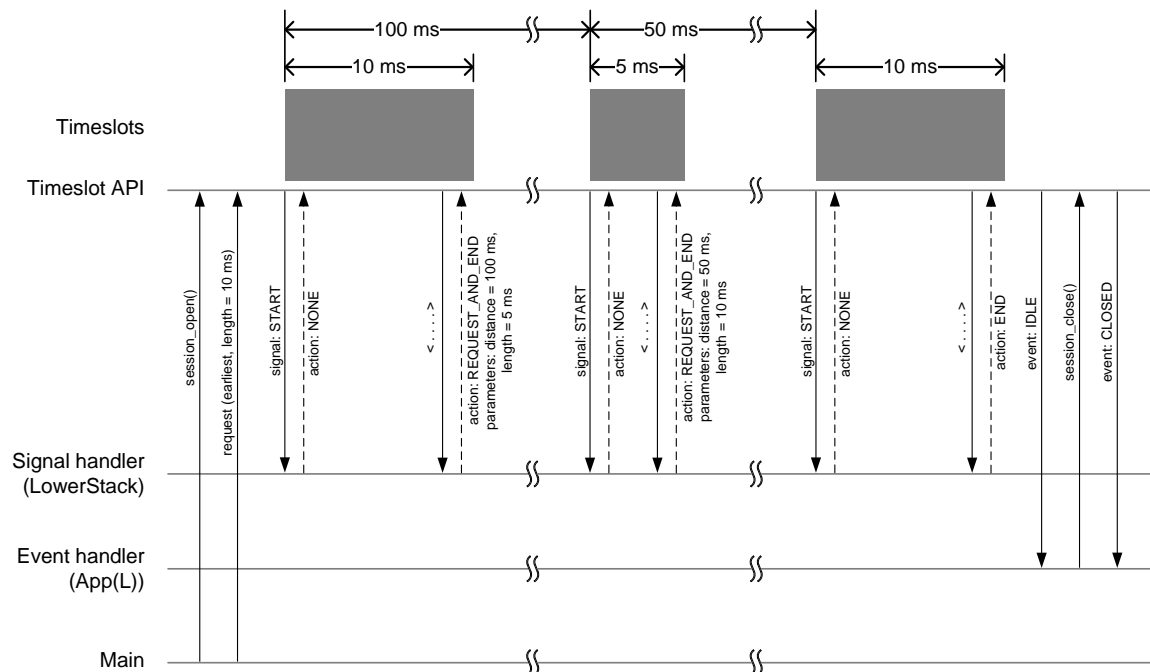
Figure 9-1. Complete Radio Timeslot session example

9.3.2 Blocked timeslot scenario

Radio Timeslot requests may be blocked due to an overlap with activities already scheduled by the SoftDevice. Figure 9-2 shows a situation in the middle of a session where this occurs. At the end of the first timeslot, 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 a scheduled SoftDevice activity. The application is notified about this by a BLOCKED event and makes a new request for a later time. This request does not collide with anything, so a new timeslot is successfully scheduled.
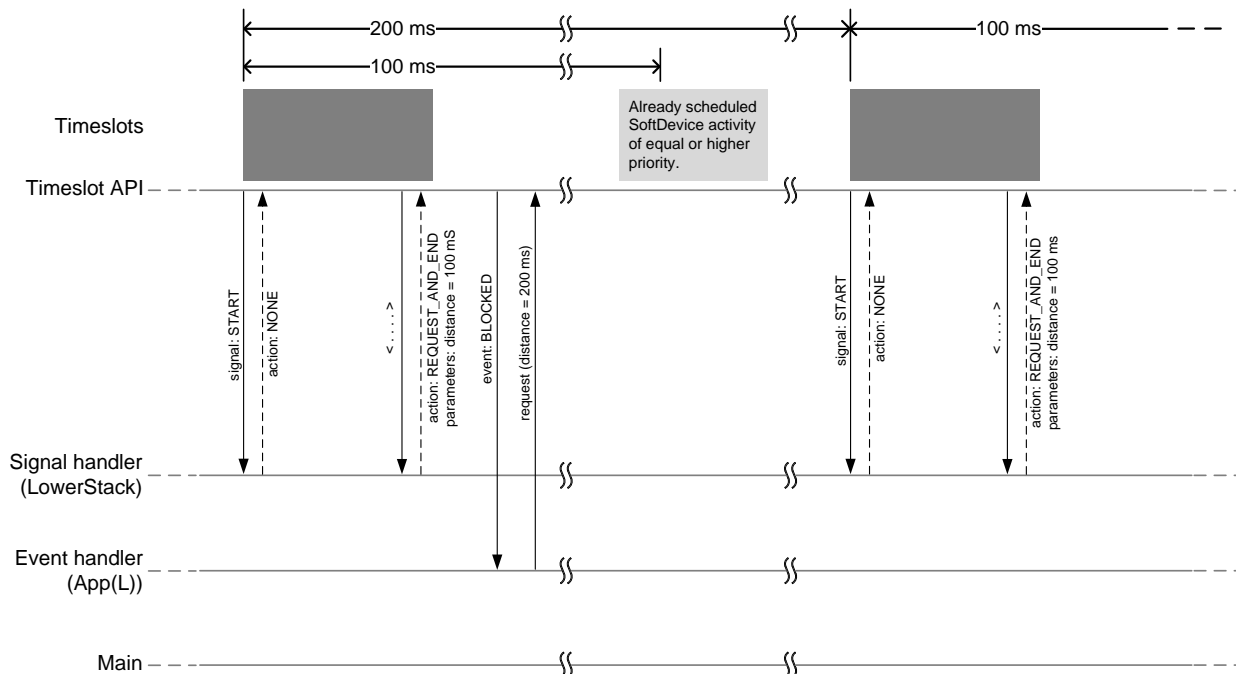
Figure 9-2. Blocked timeslot scenario

9.3.3   Cancelled timeslot scenario

Situations may occur in the middle of a session where a requested and scheduled application radio timeslot is revoked. Figure 9-3 shows a situation in the middle of a session where this occurs. 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 CANCELLED event sent to the application event handler. The application then makes a new request at a later point in time. This request does not collide with anything, so a new timeslot is successfully scheduled.
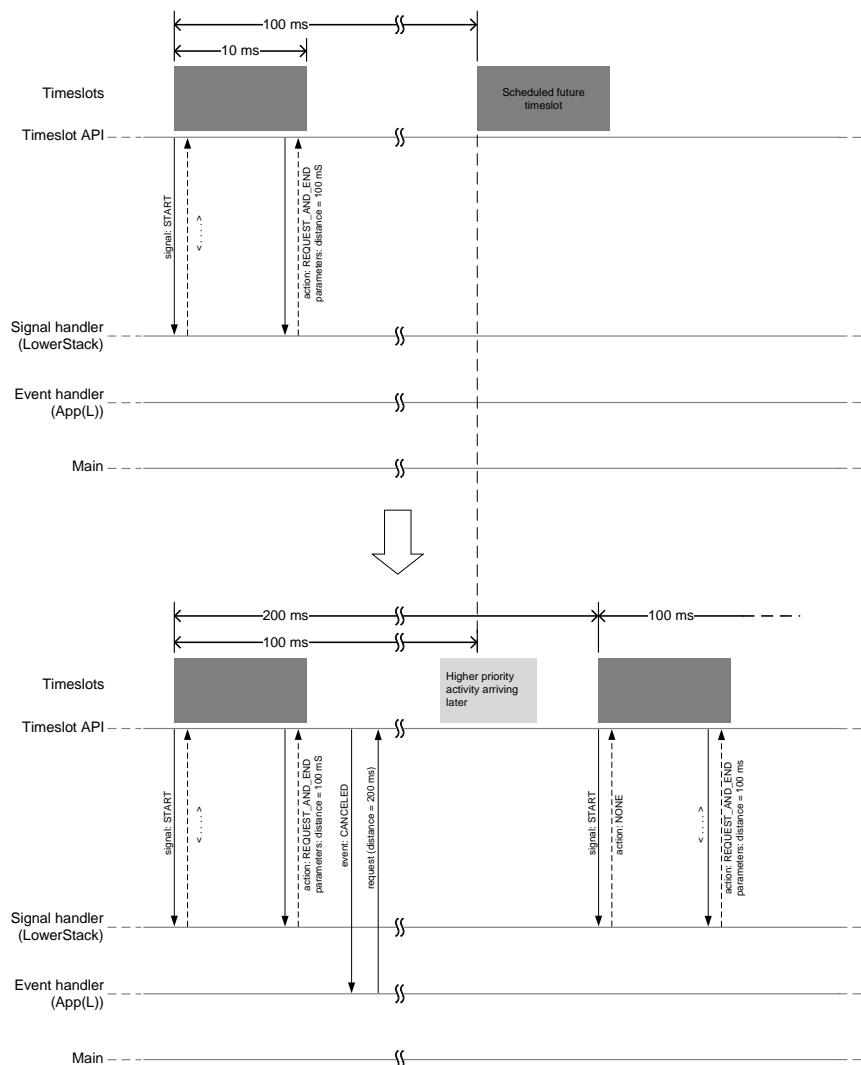
Figure 9-3. Cancelled Timeslot Scenario

9.3.4   Radio Timeslot extension example

An application can use Radio Timeslot extensions to create long continuous timeslots that will give the application as much radio time as possible while minimising any disturbance to SoftDevice activities.

In the first timeslot in Figure 9-4 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.
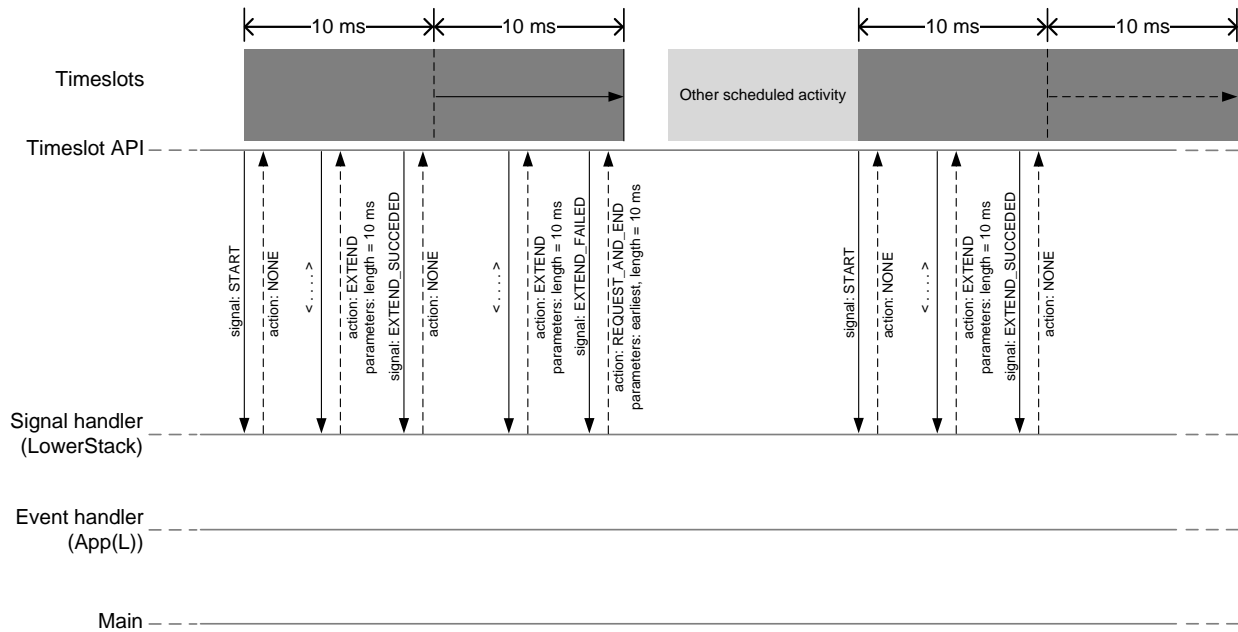
Figure 9-4. Radio Timeslot Extension Example

10   ANT Protocol Stack

The SoftDevice is a fully qualified ANT compliant stack that supports all ANT core functionality, including the implementation of ANT+ device profiles. See www.thisisant.com for further information regarding ANT.

Note: ANT+ device profile implementations must pass the ANT+ certification process to receive ANT+ Compliance Certificates.

10.1   Overview

The nRF52 Software Development Kit (SDK) supplements the ANT protocol stack with complete peripheral drivers, example applications, and ANT+ profile implementations.

Note: ANT+ network keys are needed to make ANT+ compliant products. The keys can be obtained by registering as an ANT+ adapter at www.thisisant.com.
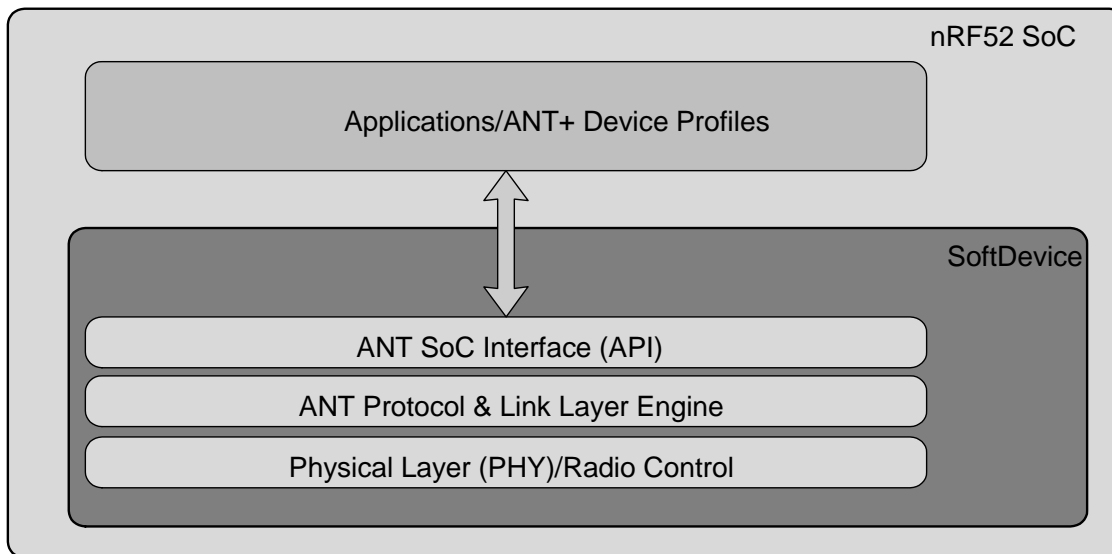
Figure 10-1. ANT Stack Architecture

10.2   ANT feature support

The S212 SoftDevice supports all applicable ANT commands described in the S210 SoftDevice Specification v3.0. Features available in the S210 SoftDevice are supported in the S212 SoftDevice. In addition, the SoftDevice includes the following enhanced ANT features described below.

10.2.1   Search Uplink

Search Uplink allows ANT channels in searching state to send uplink data to a master device. A slave channel in search mode or a background scan channel can now send broadcast or acknowledged data to a master channel while in searching state, reducing latency. In addition to this, channels in searching state can now also receive acknowledged or burst data from a master. This feature is enabled by default. For background scan channels, assigning the channel as Rx-Only will generate the legacy behaviour.

10.2.2   Group Transmitter Initiation

Group Transmitter Initiation provides the ability to configure the start-up behaviour of ANT synchronous transmission channels for high density group transmitter setups. The property of the transmission search window is adjusted to detect for adjacent interference or possible collision sources before starting transmission. Using this feature will help minimize transmitter interference when adding transmitters to a high density group transmitter setup at the cost of increased power consumption and start-up latency.

10.2.3 High Duty Search

High Duty Search uses the entire available resources of the radio to search for a master device. The effect is that latency to acquire the master device is significantly reduced to an average of ½ of the elapsed time between transmitted messages assuming ideal RF conditions. I.e. for a message period of 4Hz, the average acquisition latency is ½ of 0.25 seconds: 0.125 seconds. This mode of operation has high power consumption while in search and should only be used in applications that have considerable power resources available. This feature can be enabled for standard searches and for background scanning.

10.2.4 Time Sync

Time Sync allows independent devices to synchronize the time at which an event was generated on the transmitting device in the time base of the receiving device(s). Using this method multiple devices can synchronize themselves within two clock ticks. Some examples of usage could include synchronizing data between sensors, display elements on devices, or movement in autonomous robotics. Time sync can be used concurrently on multiple channels.

11 Radio Notification

The Radio Notification signals are sent prior to and at the end of SoftDevice Radio Events or application Radio Events[11], i.e. defined time intervals of radio operation.

11.1 Radio Notification Signals

To ensure that the Radio Notification signals behave in a consistent way, configuring Radio Notification signals shall only be done when the SoftDevice is in an idle state with no protocol stack or other SoftDevice activity in progress. Therefore, it is recommended that the Radio Notification signals be configured immediately 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 a software interrupt, as specified in Table 7-3. Refer to Table 11-1 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 11-1 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 sufficient time between the Radio Events ($t_{gap} < t_{ndist}$), the ACTIVE and nACTIVE notification signals will be skipped. There will still be an ACTIVE signal before the first event and a nACTIVE signal after the last event. This is shown in Figure 11-2.
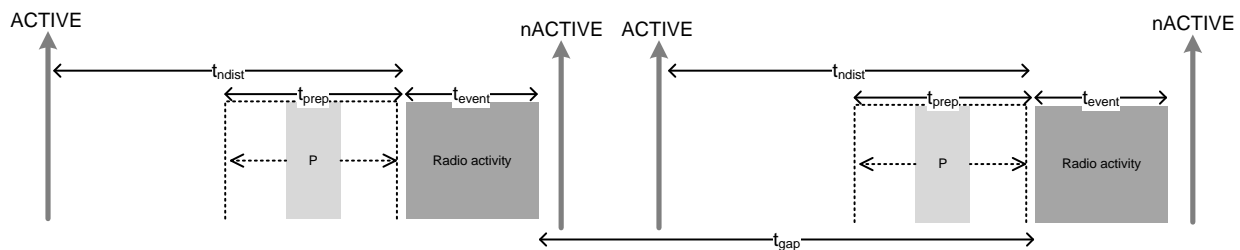
Figure 11-1. Two Radio Events with ACTIVE and nACTIVE Signals

---

[11] Application Radio Events are defined as Radio Timeslots (section 9).
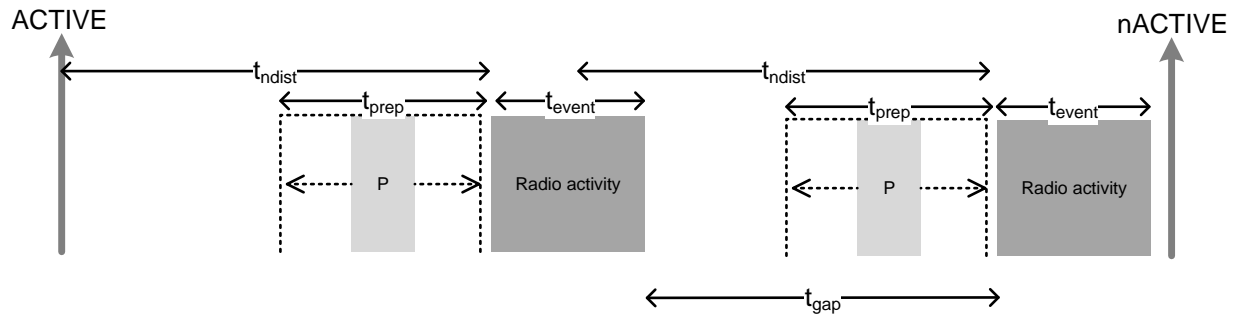
Figure 11-2. Two radio events where t_{gap} is too small and the notification signals will not be available between the events

Table 11-1. 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 a nACTIVE signal.
P	SoftDevice CPU processing at interrupt priority level 0 between the ACTIVE signal and the start of the Radio Event.	The CPU processing may occur anytime within the t_{prep} interval, 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.
t_P	Time used for preprocessing before the Radio Event.	
$t_{interval}$	Time period of periodic protocol Radio Events (e.g. ANT channel period).	

Label	Description	Notes
$t_{ScanReserved}$	Reserved time needed by the SoftDevice for each ScanWindow	

11.2 ANT traffic Radio Notifications

Radio Notifications are recommended for synchronizing application logic with radio activity, but not for synchronizing with packet transfers. Instead, ANT event messages should be used as triggers for data loading.

11.2.1 ANT Broadcast traffic

There are two cases for ANT channels that transmit Broadcast messages.
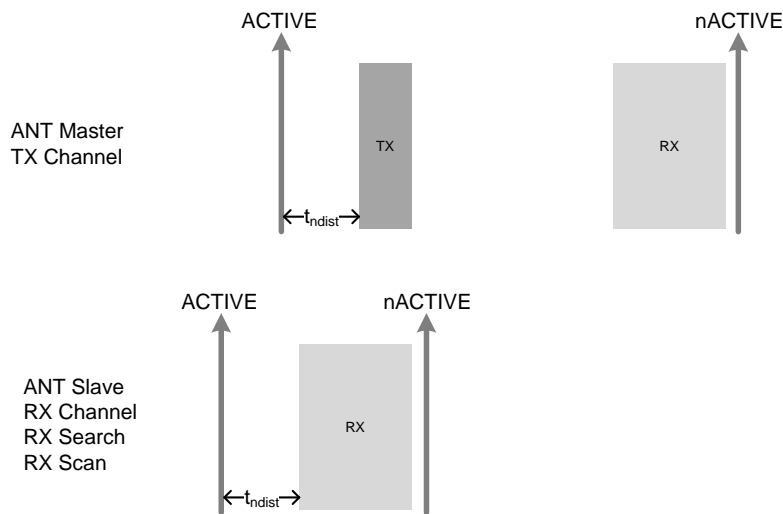
Figure 11-3. ANT Broadcast with t_{ndist} >= 1740μs

When the Radio Notifications are configured with a distance of 1740μs or greater the Master channel will only receive one notification before the transmit window and one after the receive window as shown in Figure 11-3. The Slave will only receive one pair of notifications.
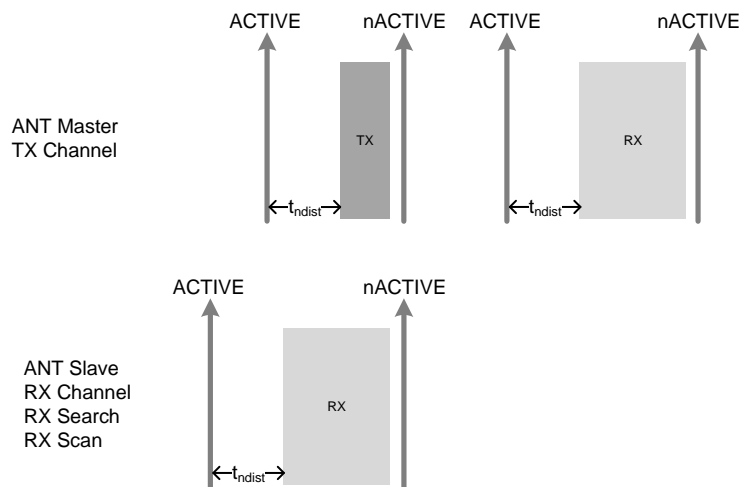
Figure 11-4. ANT Broadcast with t_{ndist} = 800μs

When the Radio Notifications are configured with a distance of 800μs the Master channel will receive an extra pair of Radio Notifications between the transmit and receive windows. The Slave will still only receive one pair of notifications.

11.2.2 ANT Burst traffic

There are three cases for ANT Burst Transfers.
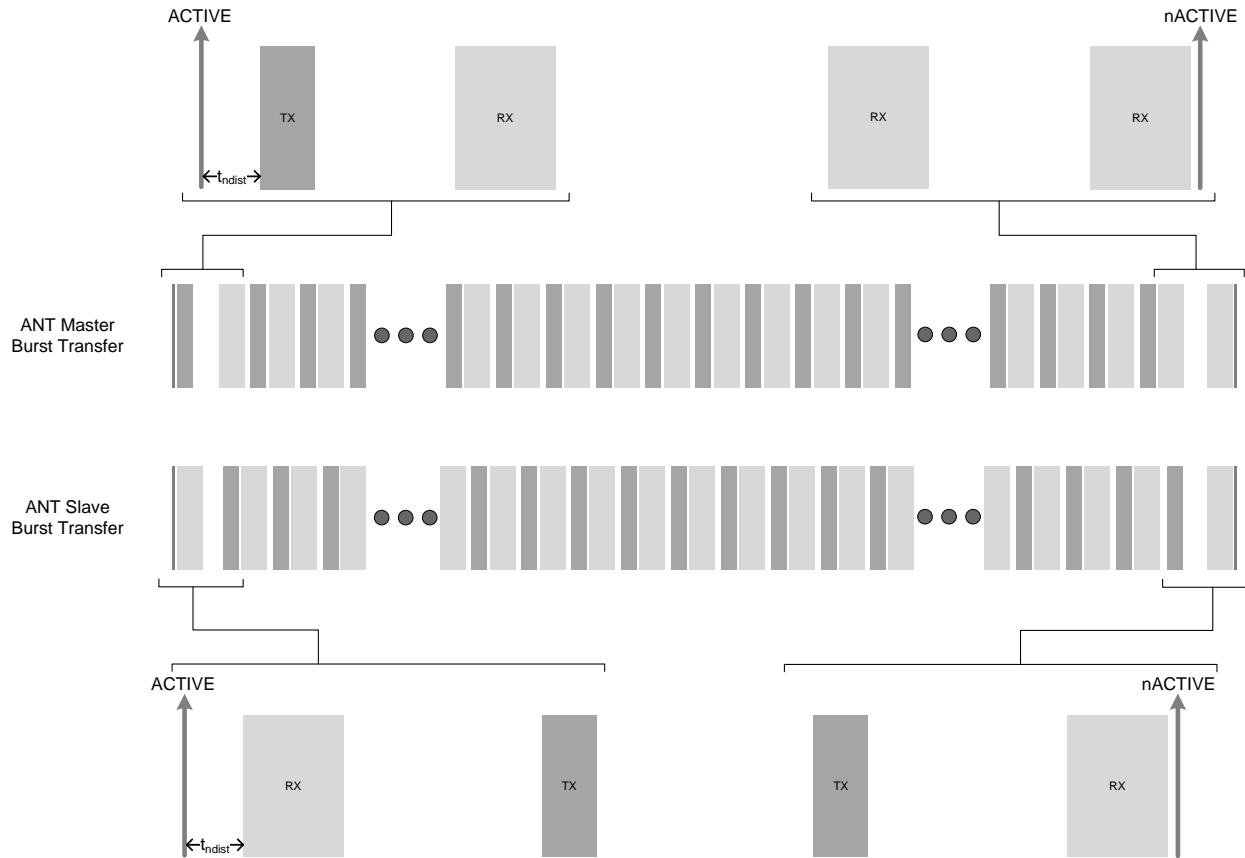
Figure 11-5. ANT Burst with $t_{ndist} >= 2680\mu s$

When the Radio Notifications are configured with a distance of 2680μs or greater the Master Channel and Slave Channel will only receive one notification before and one after the burst (Figure 11-5).
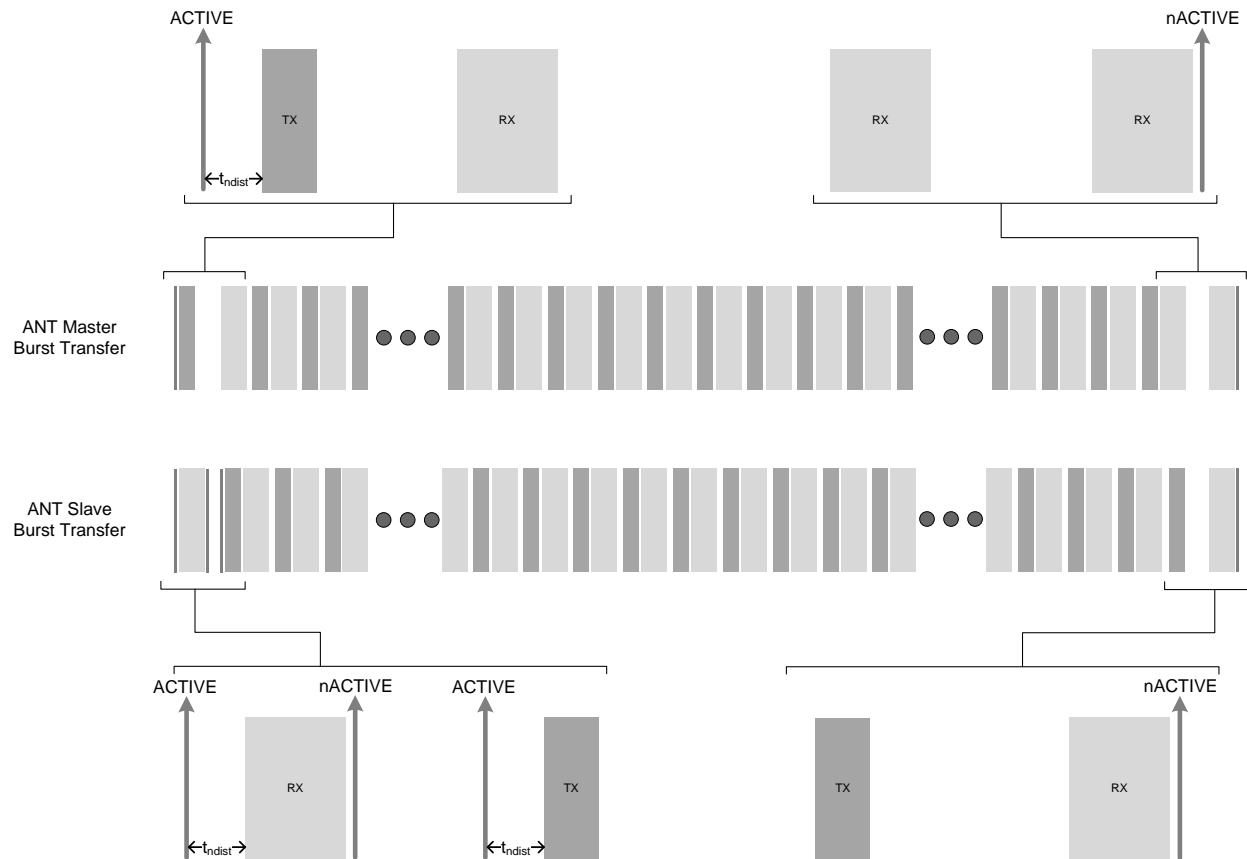
Figure 11-6. ANT Burst with ndist = 1740µs

When the Radio Notifications are configured with a distance of 1740µs the Master channel will still only receive a notification before and after the burst. The Slave channel will receive an extra pair of notifications after the initial receive window as shown in Figure 11-6.
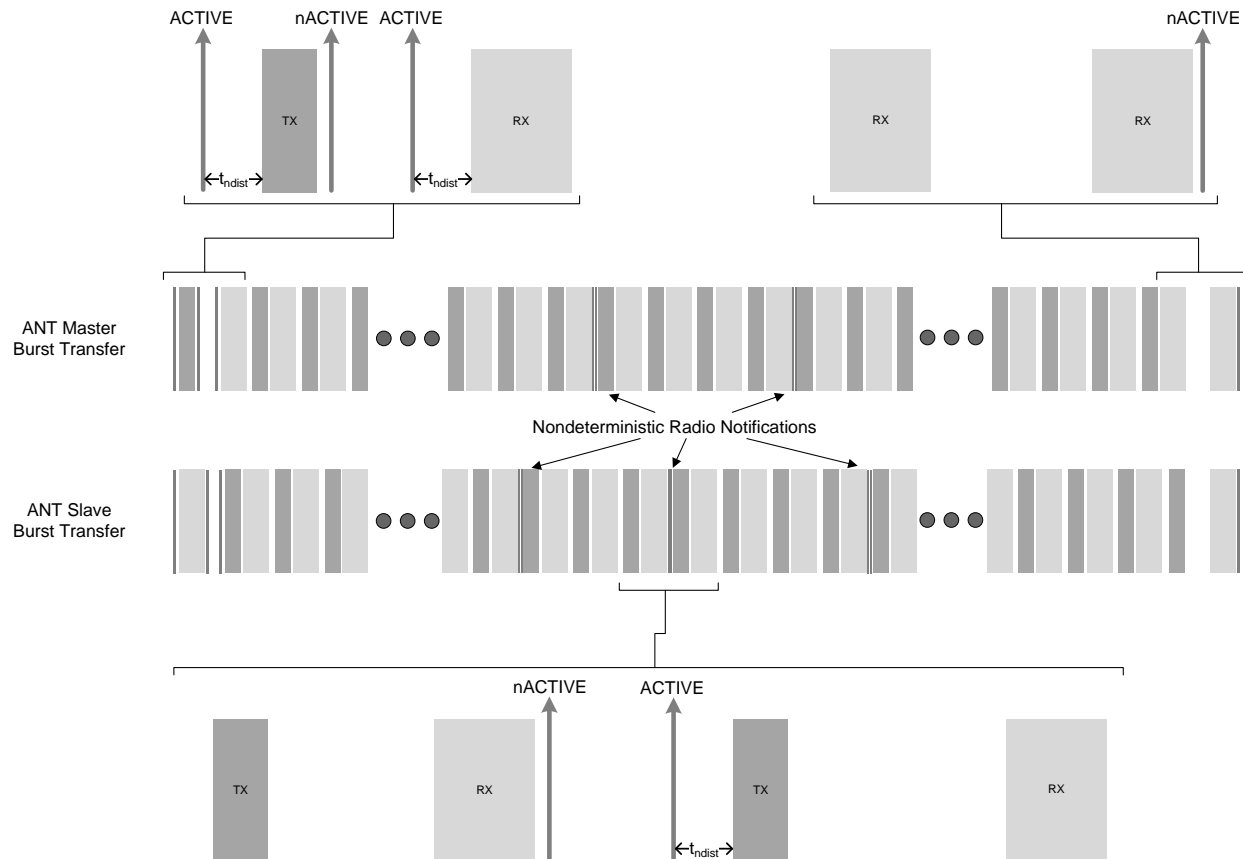
Figure 11-7. ANT Burst with ndist = 800µs

When the Radio Notifications are configured with a distance of 800µs the Master channel and Slave channel will receive notification pairs between the burst transmissions occasionally. This is due to slight time variances between the burst transmissions. The burst transmission window size is on the edge of detection for Radio Notifications and these variations will cause notifications sometimes but not others. 800µs is not a recommended distance setting for Radio Notifications.

11.3 Power Amplifier and Low Noise Amplifier control configuration (PA/LNA)

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

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

Important: In order to be used along with proprietary radio protocols that make use of the Timeslot API, the PA/LNA control functionality needs to be implemented as part of the proprietary radio protocol stack.
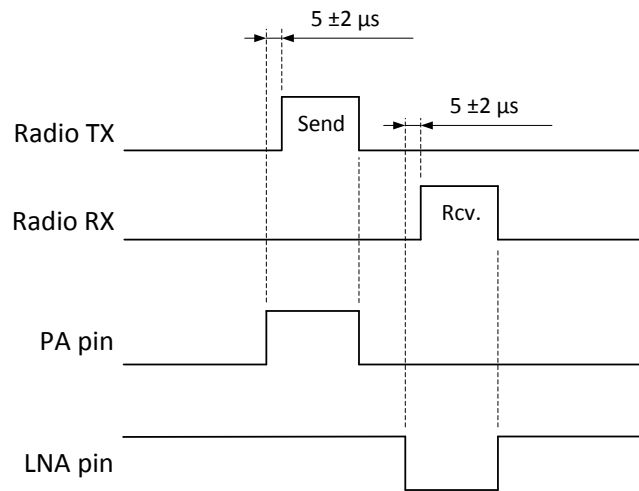
Figure 11-8. Example of timings when the PA/LNA control is enabled. The PA pin is configured active high, and the LNA pin is configured active low.

The PA and the LNA are controlled by one GPIO pin each, which are activated, respectively, during radio transmission and radio reception. The pins can be configured to be active low or active high. The SoftDevice will guarantee that the pins will be set to active 5±2μs before the EVENTS_READY signal on the RADIO using a GPIOTE connected with a PPI channel to a timer. The selected time difference allows for a sufficient ramp up time for the amplifiers, while it avoids activating them too early during the radio start up procedure (which results in amplifying carrier noise etc.). The pins are restored to inactive state using a PPI connected to the EVENTS_DISABLED event on the RADIO. See nRF52832 Product Specification for more details on the nRF52 RADIO notification signals.

12 Master Boot Record and bootloader

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

The nRF52 software architecture includes a Master Boot Record (MBR) (Figure 3-1). 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.

12.1 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 flash 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 section 16 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.

In order to issue the `SD_MBR_COMMAND_COPY_BL` or `SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET` commands to the MBR, the `UICR.NRFFW[1]` register must be set to an address (`MBRPARAMADDR` address in Figure 12-1) corresponding to a page in the Application Flash Region (section 5.4). If `UICR.NRFFW[1]` is not set the commands will return `NRF_ERROR_NO_MEM`. This page will be cleared by the MBR and used to store parameters before chip reset. When the `UICR.NRFFW[1]` register is set, the page it refers to must not be used by the application. If the application does not want to reserve a page for the MBR parameters, it must leave the `UICR.NRFFW[1]` register set to 0xFFFFFFFF (its default value).

12.2 Bootloader

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

The bootloader has full access to the SoftDevice API and can be implemented like any application that uses the SoftDevice. In particular, the bootloader can make use of the SoftDevice API for ANT 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.
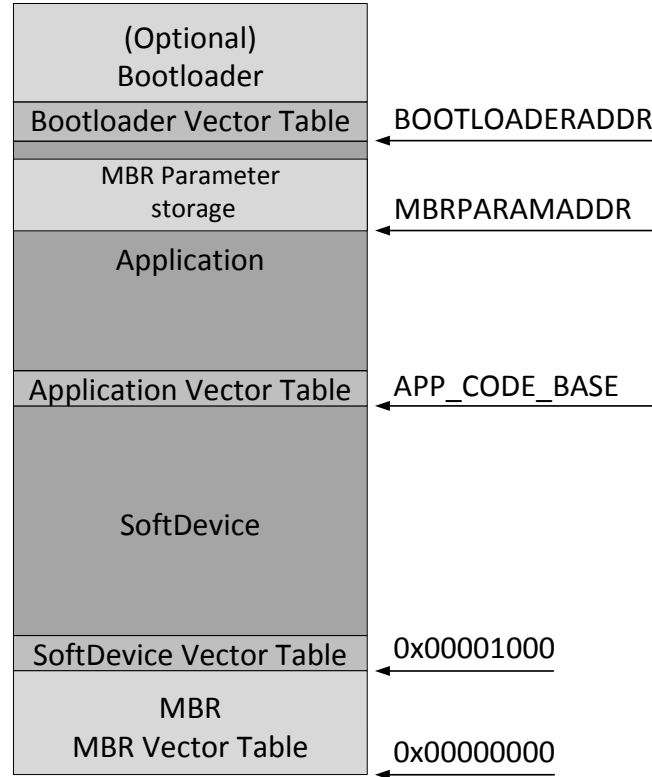
Figure 12-1. MBR, SoftDevice and bootloader architecture

12.3 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 behaviour 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.

12.4 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.

13   SoftDevice information structure

The SoftDevice binary file contains an information structure.

The structure is illustrated in Figure 13-1. 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.

	31 ... 23 ... 15 ... 7 ... 0
word 0	Reserved for future use · · · · info_size
word 1	magic_number
word 2	SoftDevice size
word 3	Reserved for future use · firmware_id
word 4	sd_id
word 5	sd_version
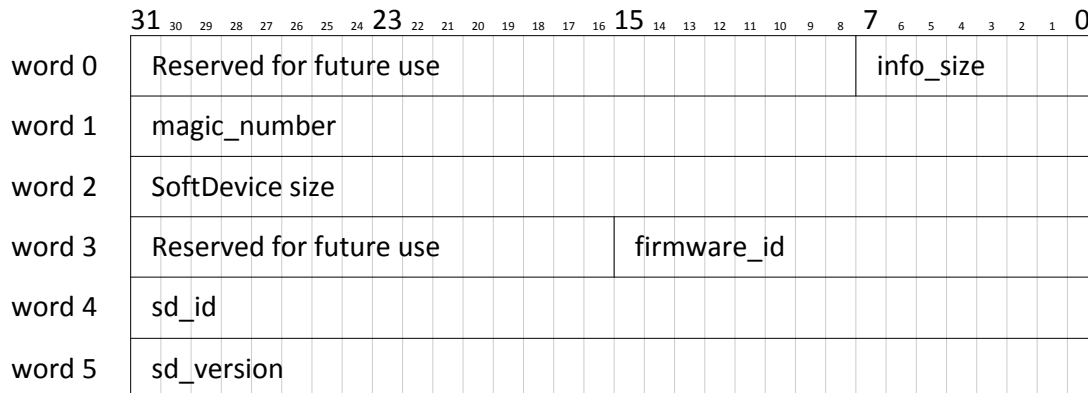
Figure 13-1. SoftDevice information structure

14 SoftDevice memory usage

The SoftDevice shares the available flash memory and RAM on the nRF52 SoC with the application. The application must therefore be aware of the memory resources needed by the SoftDevice and leave the parts of the memory used by the SoftDevice undisturbed for correct SoftDevice operation.The SoftDevice requires a fixed amount of flash memory and RAM, which are detailed in Table 14-1 and Table 14-2.

14.1 Memory resource map and usage

The memory map for program memory and RAM when the SoftDevice is enabled is described in this section. Figure 14-1 illustrates the memory usage of the SoftDevice alongside a user application. The flash memory for the SoftDevice is always reserved and the application program code should be placed above the SoftDevice at APP_CODE_BASE. The SoftDevice uses the first 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 14-2. 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 behaviour: see Table 14-2 for minimum RAM requirements.
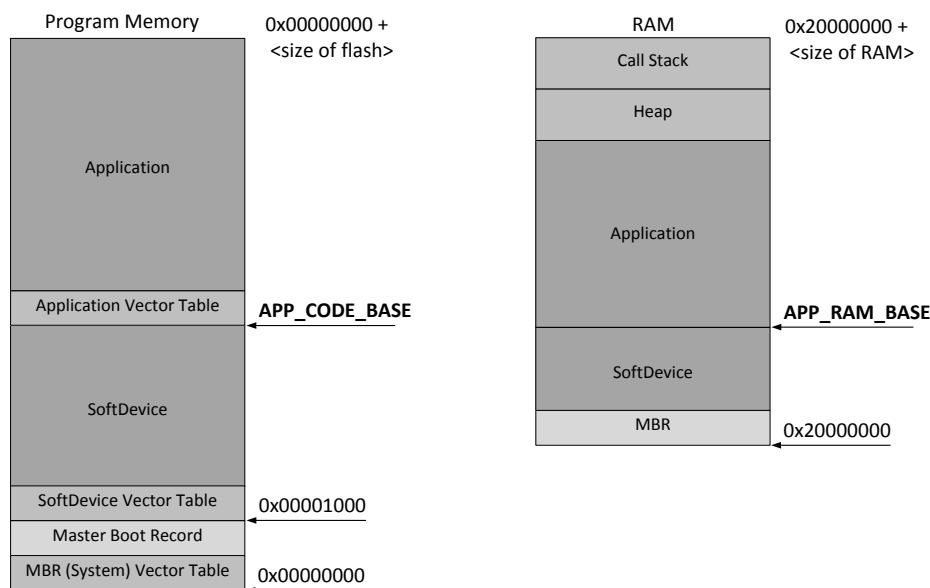
Figure 14-1. Memory Resource Map

14.1.1 Memory resource requirements

The tables below show the memory resource requirements both when the S212 SoftDevice is enabled and disabled.

Flash

Table 14-1. S212 Memory resource requirements for flash

Flash	Value
Required by the SoftDevice	72kB[12]
Required by the MBR	4 kB
APP_CODE_BASE address (absolute value)	0x00012000

RAM

Table 14-2. S212 Memory resource requirements for RAM

RAM	S212 Enabled	S212 Disabled
Required by the SoftDevice (in bytes)	0xA80 + Configurable Resources	8
APP_RAM_BASE address (minimum required value)	0x20000000 + SoftDevice RAM consumption	0x20000008

Call Stack

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

---

[12]1kB = 1024 bytes

The application must reserve sufficient space to satisfy both the application and the nRF52 SoftDevice stack memory requirements. The nRF52 SoC has no designated hardware for detecting stack overflow. The application can use the ARM® Cortex®-M4 MPU to implement a mechanism for stack overflow detection.

The SoftDevice does not use the ARM® Cortex®-M4 Floating-point Unit (FPU) and does not configure any floating-point registers. Table 14-3 gives the maximum call stack size that may be consumed by the SoftDevice when not using the FPU.

Note that the SoftDevice uses multiple interrupt levels, as detailed in section 16. If the FPU is used by the application, the processor will need to reserve memory in the stack frame for stacking the FPU registers, for each interrupt level used by the SoftDevice. This must be accounted for when configuring the total call stack size. For more information on how the use of multiple interrupt levels impacts the stack size when using the FPU, refer to the ARM® Cortex®-M4 processor with FPU documentation, Application Note 298.

Table 14-3. S212 Memory resource requirements for call stack

Call stack	S212 Enabled	S212 Disabled
Maximum usage with FPU disabled	1536bytes (0x600)	0 bytes

Heap

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

14.2   ANT Channel Configuration

This SoftDevice allows for applications to tailor and scale the following ANT stack options using an ANT Stack Enable Configuration API.

- Total number of ANT channels

- Number of encrypted ANT channels

- Transmit burst queue size

- Event queue size

Upon calling `sd_softdevice_enable()`, the ANT stack defaults to supporting one ANT channel (with encryption support), a 64 byte transmit burst buffer, and an event queue depth of 32 events If additional channels are needed and/or more encrypted channels are needed and/or a larger Tx burst buffer size is needed and/or a larger event queue is needed, then sd_ant_enable() must be used to specify the desired configuration. Application RAM memory must be supplied to the SoftDevice in order to increase these stack options beyond the default capabilities.

15   Scheduling

The S212 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: called a timing-event. Examples of timing-activities include: ANT channels, Flash memory API usage, and Radio Timeslot API timeslots.

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

The timing-activities run to completion and cannot be pre-empted by other timing-activities, even if the timing-activity trying to pre-empt has a higher priority. This is the case, for example, when timing-activity A and timing-activity B request a timing-event at an 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.

15.1   SoftDevice timing-activities and priorities

The SoftDevice supports up to fifteen ANT channels as master or slave. In addition to these ANT channels, Flash memory API and Radio Timeslot API can run simultaneously.

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. Table 15-1 summarizes the priorities:

Table 15-1. Scheduling priorities

Priority (Decreasing order)	Role state
First priority	N/A
Second priority	N/A
Third priority	• ANT channels that have been blocked for some time
Fourth priority	• Flash access after it has been blocked consecutively for a few times. • Radio Timeslot with high priority. • Normal ANT traffic
Last priority	• Flash access • Radio Timeslot with normal priority

15.2   Flash API timing

Flash timing-activity is a one-time activity with no periodicity, as opposed to ANT 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 to Table 15-1 for details on the priority of timing-activities, which are 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 waiting long while using the Flash API, flash timing-activity will fail in the case where it cannot get a timing-event before a timeout.

15.3   Timeslot API timing

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

Refer to Table 15-1 for details on the priority of timing-activities, which are 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 cancelled. However, a timing-event that has already started cannot be interrupted or cancelled.

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

16 Interrupt model and processor availability

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

16.1 Exception model

As the SoftDevice, including the 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 section 12.1.

16.1.1 Interrupt forwarding to the application

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

At the lowest level, the MBR receives all interrupts and forwards them to the SoftDevice regardless of whether the SoftDevice is enabled or not. The use of a bootloader introduces some exceptions to this (see section 12).

Some peripherals and their respective interrupt numbers are reserved for use by the SoftDevice (section 7.1). 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 14-1. When the base address of the application code is directly after the top address of the SoftDevice, the code can be developed as a standard ARM® Cortex® -M4 application project with the compiler creating the interrupt vector table.

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

Latency, additional to ARM® Cortex® -M4 hardware architecture latency, is introduced by SoftDevice logic to manage interrupt events.

This latency occurs when an interrupt is forwarded to the application from the SoftDevice and is part of the minimum latency for each application interrupt. This is the latency added by the interrupt forwarding latency alone. The maximum application interrupt latency is dependent on SoftDevice activity, as described in section 16.3.

Table 16-1. Additional latency due to SoftDevice and MBR forwarding interrupts

Interrupt	SoftDevice enabled	SoftDevice disabled
Open peripheral interrupt	< 2 µs	< 1 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	N/A	< 2 µs
Application SVC interrupt	< 2 µs	< 2 µs

16.2   Interrupt priority levels

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

To implement the SoftDevice API as SuperVisor Calls (SVCs, see section 4) 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 pre-empted, 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® -M4 processor has eight configurable interrupt priorities ranging from 0 to 7 (with 0 being highest priority). On reset, all interrupts are configured with the highest priority (0).

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

- Level 0 is used for the SoftDevice's timing critical processing.

- Level 1 is used for handling the memory isolation and run time protection (section 5.4).

- Level 4 is used by higher level deferrable tasks and the API functions executed as SVC interrupts.

- Level 5 is reserved for future use.

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

As seen from Figure 16-1, the application has available priority level 2 and 3, located between the higher and lower priority levels reserved by the SoftDevice. This enables a low-latency application interrupt to support fast sensor interfaces. An application interrupt at priority level 2 or 3 will only experience latency from SoftDevice interrupts at priority levels 0 and 1, while application interrupts at priority levels 6 or 7 can experience latency from all SoftDevice priority levels.

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 4 (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).

Cortex M4
Priorities
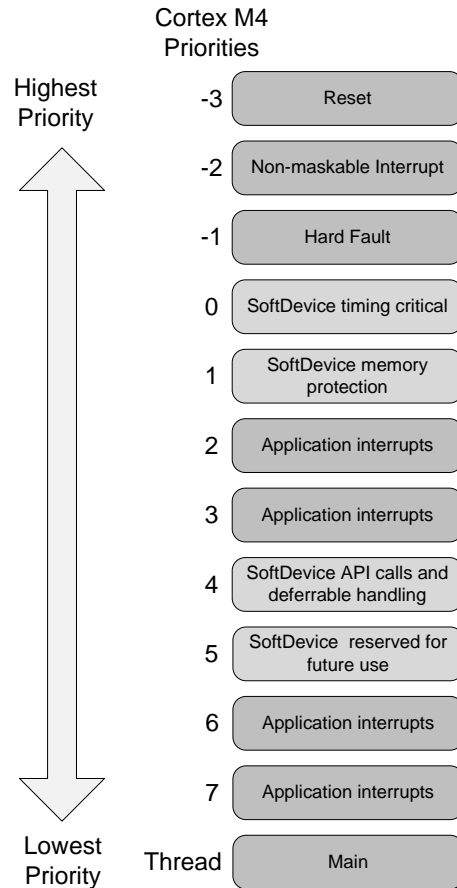
Figure 16-1. Exception model

The figure below shows an example of how interrupts with different priorities may run and pre-empt each other. Note that some priority levels are left out for clarity.

Softdevice - Exception examples
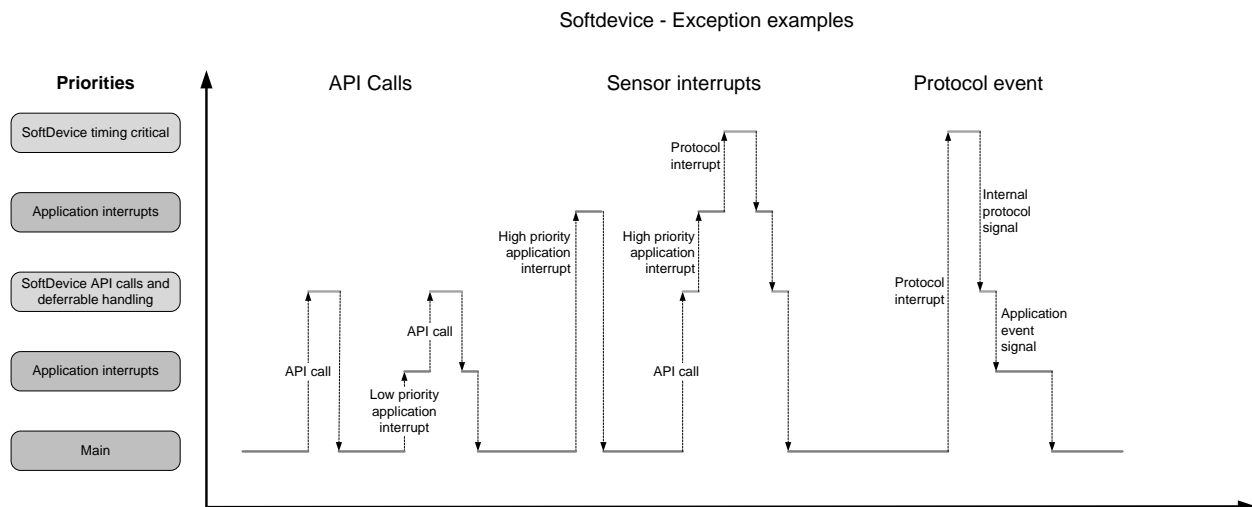
Figure 16-2. SoftDevice Exception examples (some priority levels left out for clarity)

## 16.3   Processor usage patterns and availability

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

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

In the 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.

16.3.1   Flash API processor usage patterns

This section describes the processor availability and interrupt processing time for the SoftDevice when the Flash API is being used.
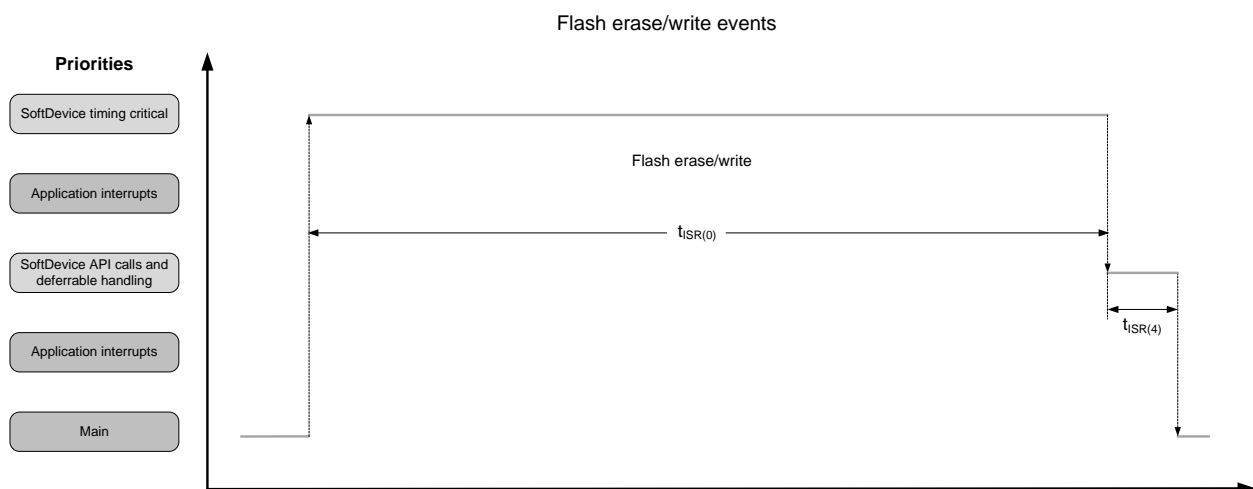
Figure 16-3. Flash API activity (some priority levels left out for clarity)

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 4 that does some clean up after the flash operation.

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

Table 16-2. 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 most of the length of this interrupt.			90ms

Parameter	Description	Min	Typical	Max
$t_{ISR(0),FlashWrite}$	Interrupt processing when writing one or more words to flash. Note that the CPU is halted most of the length of this interrupt. The Max time provided is for writing one word. When writing more than one word, please see the Product Specification to find out how long time is needed for per word to write, and add to the Max time provided in this table.			500 µs
$t_{ISR(4)}$	Priority level 4 interrupt at the end of flash write or erase.		10 µs	

16.3.2　Radio Timeslot API processor usage patterns

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

See section 9.2.6 for more information on the Radio Timeslot API.
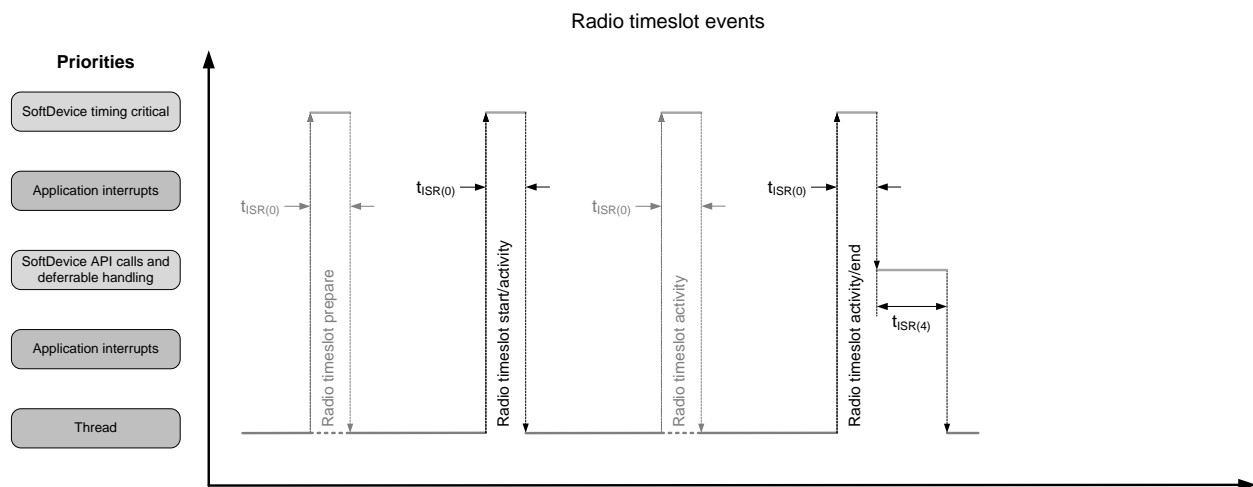
Figure 16-4. Radio Timeslot API activity (some priority levels left out for clarity)

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 4 that does some clean up 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 16-3. 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.			9 µs
$t_{ISR(0),RadioTimeslotActivity}$	The application processing timeslot. The length of this is application dependent.			
$t_{ISR(4)}$	Priority level 4 interrupt at the end of the timeslot.		7 µs	

16.3.3 ANT processor usage patterns

16.3.3.1 ANT processor priorities

The breakdown for a single ANT transmit or receive protocol event is shown in Figure 16-5.
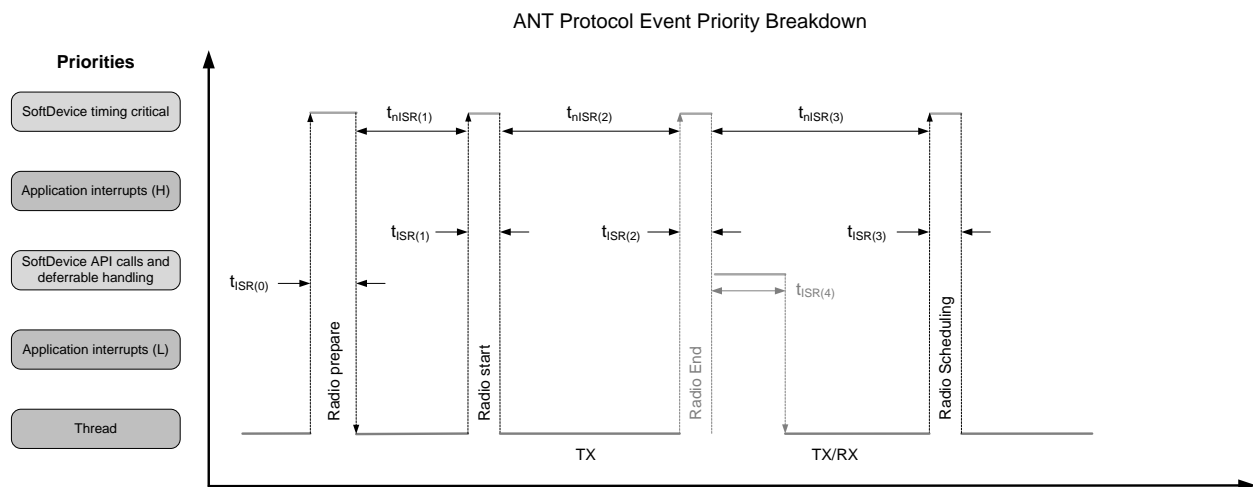
Figure 16-5. ANT Protocol Event

If required, Application interrupts (H) may be used. However, the interrupt should not exceed 100 µs per 500 µs interval or ANT performance will be adversely affected. If high application interrupts are not needed, all application processes should be run at Application interrupts (L) and/or at Thread level.

To improve RF transmit integrity, system power changes are prevented during radio transmissions (Radio start to Radio end). Applications issuing the SoftDevice wait for event API call will not result in the CPU entering idle/sleep mode until the transmission activity has completed.

ANT radio and stack processing times for all supported ANT activity types in typical use cases (without high application priority interruption) are summarized in Table 16-4. High priority application interrupts will directly affect ANT stack processing overhead time ($t_{ISR(4)}$).

Table 16-4. Processor usage latency when connected (ANT)

Parameter	Description	Min	Typical	Max
$t_{ISR(0)}$	Radio prepare time			58us
$t_{ISR(1)}$	Start of radio activity		12us	
$t_{ISR(2)}$	End of radio activity			19us
$t_{ISR(3)}$	Scheduling next radio session		37us	
$t_{ISR(4)}$	ANT stack processing time			168us
$t_{nISR(1)}$	Idle time between prepare and start	117us		
$t_{nISR(2)}$	Idle time during radio on	173us		
$t_{nISR(3)}$	Idle time between radio end and radio schedule	129us		

16.3.3.2   ANT processor availability

This section shows the processor availability for an application with different configurations of ANT traffic. This availability is measured as idle time, i.e. the time that the SoftDevice is not using the CPU. The data collected in Table 16-5 shows the amount of idle time for different types of ANT traffic. There are average and worst cases for the idle times.

The average case covers the time that traffic is occurring and the idle time between traffic. For example, a 4Hz broadcast channel would have a 99% average idle time, this would include the time that the traffic is occurring and the ~250ms idle time between the channels.

The worst case covers only the idle time when traffic is occurring. For example, you can expect the CPU availability to dip to 86% on a 4Hz broadcast master channel for the short duration when ANT traffic is occurring (~4-5ms per traffic window for this type of traffic).

Table 16-5. Processor idle time for ANT traffic

Type of ANT Traffic	Average Idle Time	Worst Case Idle Time
Background Scan/Channel Search	98%	95%
High Duty Search	95%	51%
Continuous Scan	97%	97%
4Hz Broadcast (Master)	99%	85%
4Hz Broadcast (Slave)	99%	92%
4Hz Acknowledged in Both Directions (Master)	95%	78%
4Hz Acknowledged in Both Directions (Slave)	95%	77%
Standard Burst (Master)	90%	78%
Standard Burst (Slave)	90%	78%

Type of ANT Traffic	Average Idle Time	Worst Case Idle Time
Advanced Burst (Master)	90%	78%
Advanced Burst (Slave)	90%	78%
Encrypted Burst (Master)	89%	76%
Encrypted Burst (Slave)	89%	76%

16.3.4 Interrupt latency when using multiple modules and channels

Concurrent use of the Flash API, Radio Timeslot API, and/or ANT channel(s) can affect the interrupt latency.

The same interrupt priority levels are used by all Flash API, Radio Timeslot API, and ANT channels. When using more than one of these concurrently, their respective events can be scheduled back-to-back (section 9.2.4). In those cases, the last interrupt in the activity by one module/channel can be directly followed by the first interrupt of the next activity. Therefore, to find the real worst-case interrupt latency, the application developer must sum the latency of the first and last interrupts for all combinations of processor usage.

Example: If the application uses the Radio Timeslot API while having an ANT channel running, the worst case interrupt latency 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 ANT channel

- The sum of the maximum time of the first interrupt of the Radio Timeslot API and the last interrupt of the ANT channel

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

17 ANT power profiles

This chapter provides power profiles for MCU activity during ANT radio events implemented in the SoftDevice. These profiles give a detailed overview of the stages of a radio event, the approximate timing of stages within the event, and how to calculate the peak current draw at each stage using data from the product specification. The CPU profile during the event is shown separately. Currently only the master channel and slave channel profiles are shown. Similar calculations can be extended to other ANT activity modes.

17.1 Master Channel

A radio transmit and receive event for a single ANT master channel is shown in Figure 17-1.
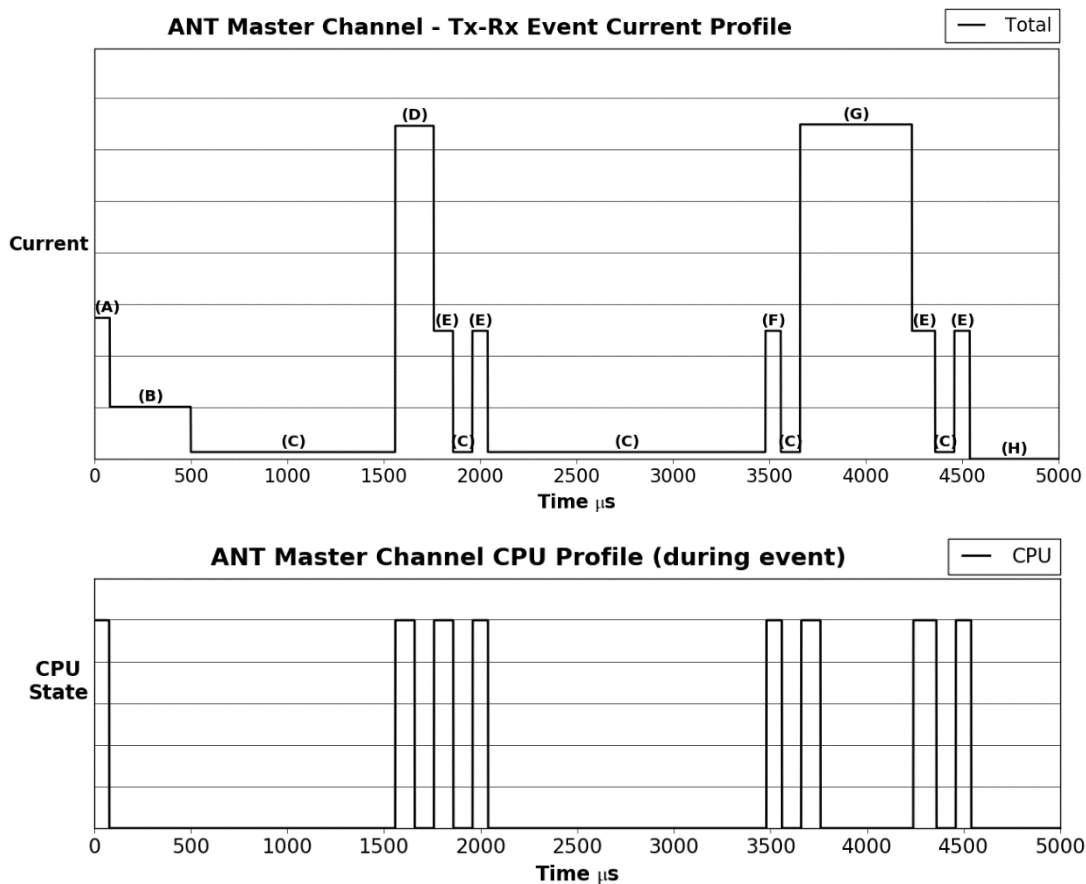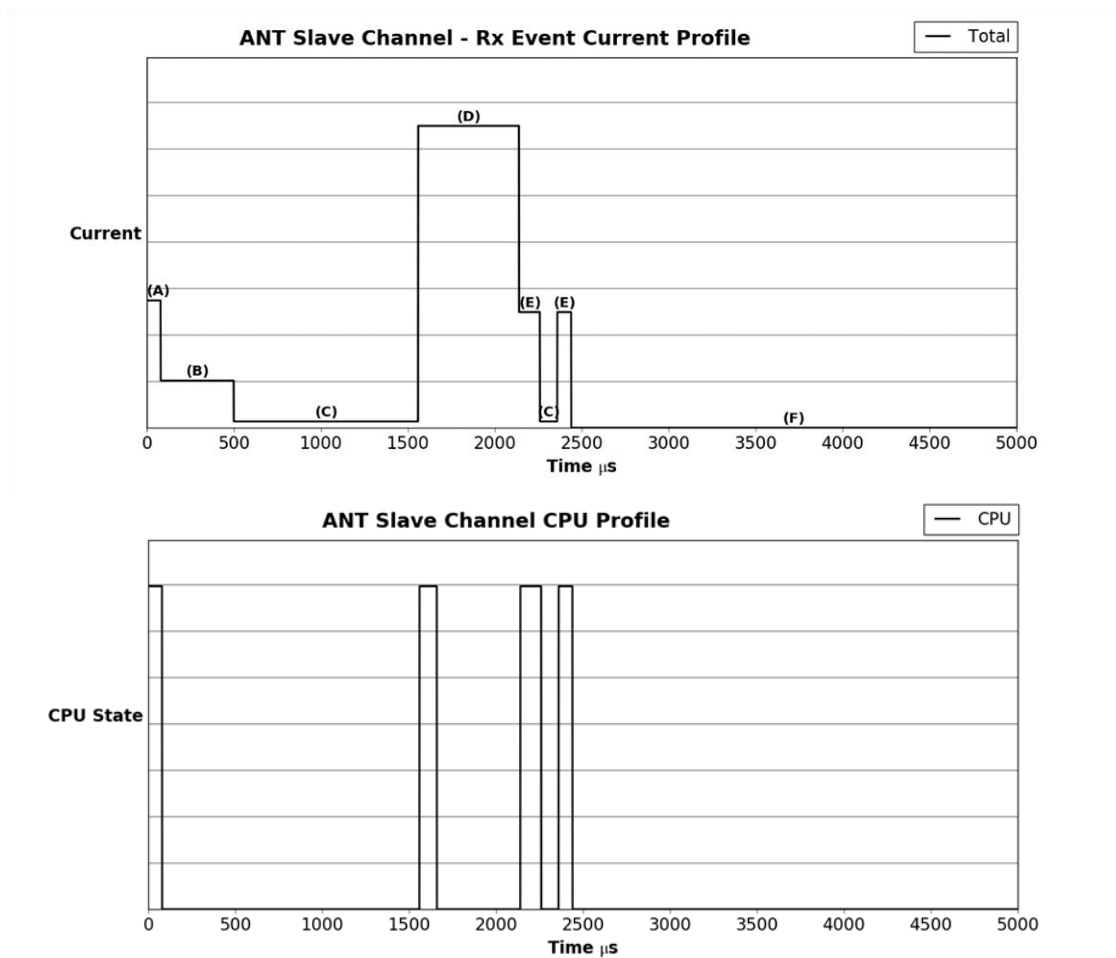
Figure 17-1. Master Channel Power and CPU Profile

Table 17-1. Master Channel power usage breakdown

Stage	Description
(A)	Pre-processing
(B)	Standby + XO ramp
(C)	Standby
(D)	Radio Tx

Stage	Description
(E)	Post-processing
(F)	Pre-processing
(G)	Radio Rx
(H)	Idle

17.2   Slave Channel

A radio receive event for a single ANT slave channel is shown in Figure 17-2.

Figure 17-2. Slave Channel Power and CPU Profile

Table 17-2. Slave Channel power usage breakdown

Stage	Description
(A)	Pre-processing
(B)	Standby + XO ramp
(C)	Standby

Stage	Description
(D)	Radio Rx
(E)	Post-processing
(F)	Idle

18 SoftDevice identification and revision scheme

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

The identification scheme for SoftDevices consists of the following items:

- For revisions of the SoftDevice which are production qualified, the version string consists of major, minor, and revision numbers only, as described in the table below.

- For revisions of the SoftDevice which are not production qualified, a test qualification level (alpha/beta) and a build number are appended to the version string.

- For example: s110_nrf51_1.2.3.alpha4, where major = 1, minor = 2, revision = 3, test qualification level is alpha and build number = 4. Additional examples are given in Table 18-2.

Table 18-1. Revision scheme

Revision	Description
Major increments	Modifications to the API or the function or behaviour 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. 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.
Test qualification level (if present)	Refer to Table 18-3.
Build number increment (if present)	New build of non-production versions.

Table 18-2. SoftDevice revision examples

Sequence number	Description
s212_nrf52_1.2.3.alpha1	Revision 1.2.3, first build qualified at alpha level
s212_nrf52_1.2.3.alpha2	Revision 1.2.3, second build qualified at alpha level
s212_nrf52_1.2.3.beta5	Revision 1.2.3, fifth build qualified at beta level
s212_nrf52_1.2.3	Revision 1.2.3, qualified at production level

Table 18-3. 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.

18.1   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.