SoftDevice Specification

S130 SoftDevice v1.0.0

2015-06-12

Contents

Chapter 1
Revision history

Date	Version	Description
June 2015	1.0	Updated to correspond to SoftDevice S130 version 1.0.0.
July 2014	0.5	Preliminary release.

Chapter 2
S130 SoftDevice

The S130 SoftDevice is a *Bluetooth*® Low Energy (BLE) central and peripheral protocol stack solution. It supports up to three connections as a central, one connection as a peripheral, an observer, and a broadcaster all running concurrently. The S130 SoftDevice integrates a BLE Controller and Host, and provides a full and flexible API for building *Bluetooth* Smart System on Chip (SoC) solutions.

Key features	Applications

• *Bluetooth* 4.2 compliant low energy single-mode protocol stack suitable for *Bluetooth* Smart products
  • Concurrent Central, Observer, Peripheral, and Broadcaster roles with up to:
  • Three connections as a central
  • One connection as a peripheral
  • Observer
  • Broadcaster
  • Link layer
  • L2CAP, ATT, and SM protocols
  • GATT and GAP APIs
  • GATT Client and Server
• Complementary nRF51 SDK including *Bluetooth* profiles and example applications
• Master Boot Record for over-the-air device firmware update
• Memory isolation between application and protocol stack for robustness and security
• Thread-safe supervisor-call based API
• Asynchronous, event-driven behavior
• No RTOS dependency
  • Any RTOS can be used
• No link-time dependencies
  • Standard ARM® Cortex™M0 project configuration for application development
• Support for concurrent and non-concurrent multiprotocol operation
  • Concurrent with the *Bluetooth* stack using concurrent multiprotocol timeslot API
  • Alternate protocol stack in application space
	• Sports and fitness devices
	  • Sports watches
	  • Bike computers
	• Personal Area Networks
	  • Health and fitness sensor and monitoring devices
	  • Medical devices
	  • Key fobs and wrist watches
	• Home automation
	• Rezence wireless charging
	• Remote control toys
	• Computer peripherals and I/O devices
	  • Mice
	  • Keyboards
	  • Multi-touch trackpads
	• Interactive entertainment devices
	  • Remote controls
	  • Gaming controllers

Chapter 3
Documentation

Required reading for a comprehensive understanding of the SoftDevice includes the SoftDevice architecture, product specification, product anomaly notifications, compatibility matrix, and Bluetooth core specification.

Below is a list of the core documentation for the SoftDevice.

Table 1: S130 SoftDevice core documentation

Documentation	Description
Appendix A: SoftDevice architecture *on page 73*	Essential reading for understanding the resource usage and performance related to chapters of this document.
nRF51822 Product Specification	Contains a description of the hardware, modules, and electrical specifications specific to the nRF51822 IC.
nRF51822 PAN	Contains information on anomalies related to the nRF51822 IC.
nRF51 Series Compatibility Matrix	Contains information on the compatibility between nRF51 Integrated Circuit (IC) revisions, SoftDevices and SoftDevice Specifications, SDKs, development kits, documentation, and Qualified Design Identifications (QDIDs).
Bluetooth Core Specification	The *Bluetooth* Core Specification version 4.2, Volumes 1, 3, 4, and 6, describes *Bluetooth* terminology which is used throughout the SoftDevice Specification.

Chapter 4
Product overview

The SoftDevice is a precompiled and linked binary image implementing a *Bluetooth* 4.2 low energy protocol stack for the nRF51 series of ICs.

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

The Application Programming Interface (API) is a set of standard C language functions and data types that give the application complete compiler and linker independence from the SoftDevice implementation.

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

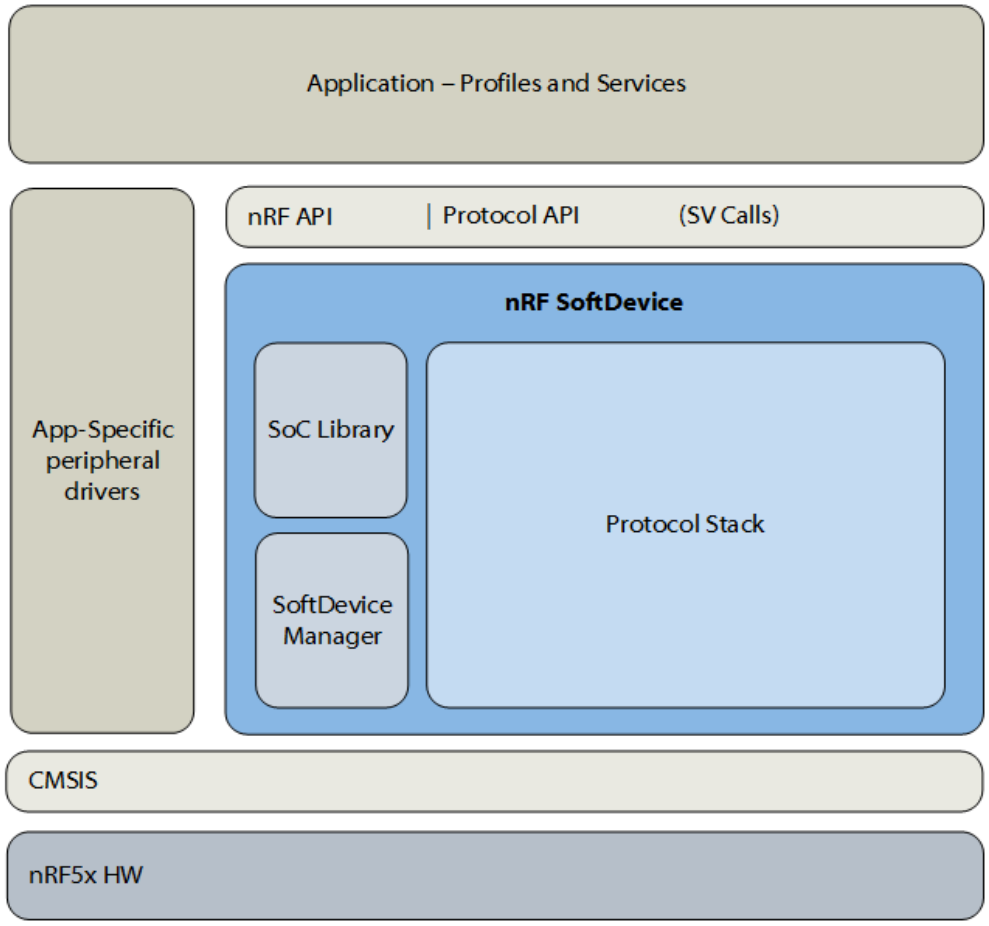


Figure 1: System on Chip application with the SoftDevice

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

4.1 Multiprotocol support

The SoftDevice supports both non-concurrent and fully concurrent multiprotocol implementations.

For non-concurrent operation, a proprietary 2.4 GHz protocol can be implemented in the application program area and can access all hardware resources when the SoftDevice is disabled.

For concurrent multiprotocol operation, with a proprietary protocol running concurrently with the SoftDevice protocol(s), see Concurrent Multiprotocol Timeslot API on page 29.

Chapter 5
Bluetooth low energy protocol stack

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

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

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

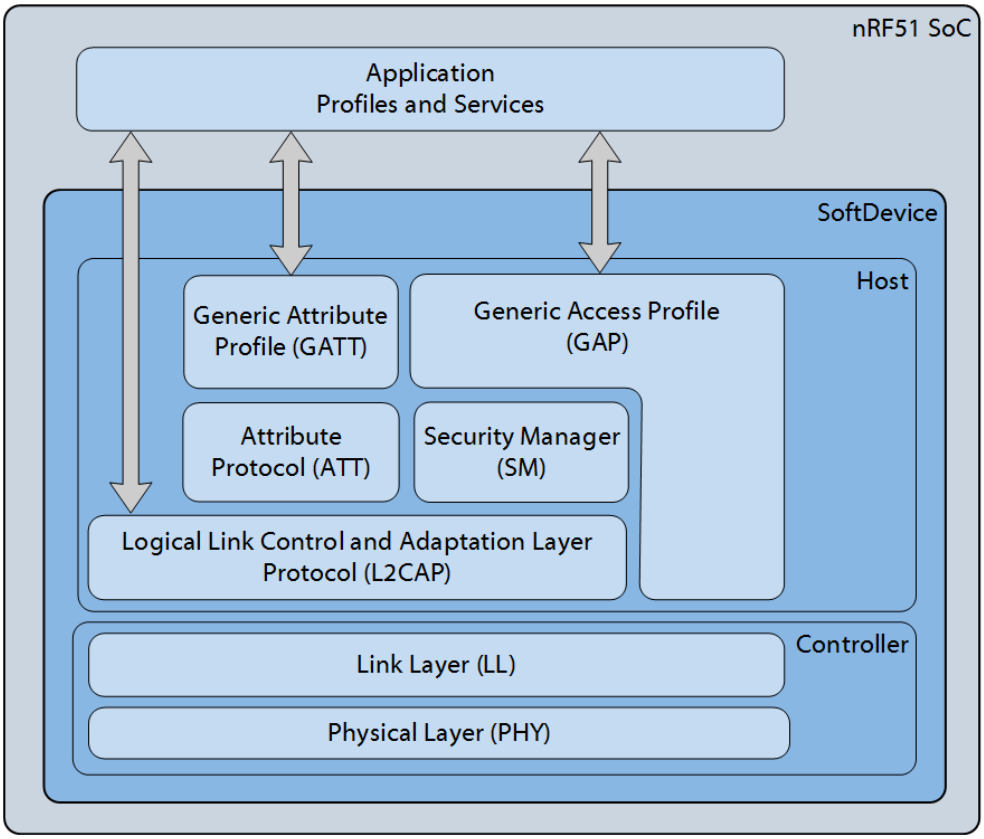


Figure 2: SoftDevice stack architecture

5.1 Profile and service support

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

The SoftDevice supports all of these as well as additional proprietary profiles.

Table 2: Supported profiles and services

Adopted profile	Adopted services
HID over GATT	HID

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

Adopted profile	Adopted services
	Device Information
Environmental Sensing	Environmental Sensing

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

5.2 *Bluetooth* low energy features

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

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

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

Table 3: API features in the BLE stack

API features	Description
Interface to: GATT / GAP	Consistency between APIs including shared data formats.
Attribute table sizing, population and access	Full flexibility to size the attribute table at application compile time and to populate it at run time. Attribute removal is not supported.
Asynchronous and event driven	Thread-safe function and event model enforced by the architecture.
Vendor-specific (128-bit) UUIDs for proprietary profiles	Compact, fast and memory efficient management of 128-bit UUIDs.
Packet flow control	Full application control over data buffers to ensure maximum throughput.

Table 4: GAP features in the BLE stack

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

Table 5: GATT features in the BLE stack

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

Table 6: Security Manager (SM) features in the BLE stack

Security Manager features	Description
Flexible key generation and storage for reduced memory requirements	Keys are stored directly in application memory to avoid unnecessary copies and memory constraints.
Authenticated MITM (Man in the middle) protection	Allows for per-link elevation of the encryption security level.

Security Manager features	Description
Pairing methods: Just works, Passkey Entry and Out of Band	API provides the application full control of the pairing sequences.

Table 7: Attribute Protocol (ATT) features in the BLE stack

ATT features	Description
Server protocol	Fast and memory efficient implementation of the ATT server role.
Client protocol	Fast and memory efficient implementation of the ATT client role.
Max MTU size 23 bytes	Up to 20 bytes of user data available per packet.

Table 8: Controller, Link Layer (LL) features in the BLE stack

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

Table 9: Proprietary features in the BLE stack

Proprietary features	Description
TX Power control	Access for the application to change TX power settings anytime.
Enhanced Privacy 1.1 support	Synchronous and low power solution for Bluetooth low energy enhanced privacy with hardware-accelerated address resolution for whitelisting.

Proprietary features	Description
Master Boot Record (MBR) for Device Firmware Update (DFU)	Enables over-the-air SoftDevice replacement, giving full SoftDevice update capability.

5.3 Limitations on procedure concurrency

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

The Host instantiates both GATT and GAP instances for each connection, while the Security Manager (SM) Initiator is only instantiated once for all connections. The Link Layer also has concurrent procedure limitations that are handled inside the SoftDevice without requiring management from the application.

Table 10: Limitations on procedure concurrency

Protocol procedures	Limitation with multiple connections
GATT	None. All procedures can be executed in parallel.
GAP	None. All procedures can be executed in parallel. Note that some GAP procedures require LL procedures (connection parameter update and encryption). In this case, the GAP module will queue the LL procedures and execute them in sequence.
SM	SM procedures cannot be executed in parallel for connections as a central. That is, each SM procedure must run to completion before the next procedure begins across all connections as a central. For example `sd_ble_gap_authenticate()`.
LL	The LL Disconnect procedure has no limitations and can be executed on any, or all, links simultaneously. The LL connection parameter update and encryption establishment procedure on a master link can only be executed on one master link at a time. Accepting connection parameter update and encryption establishment on a slave link is always allowed irrespective of any control procedure running on master links.

Chapter 6
System on Chip library

The coexistence of Application and SoftDevice with safe sharing of common System on Chip (SoC) resources is ensured with a number of features.

Table 11: 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	Gives the application access to all NVIC features without corrupting SoftDevice configurations.
Rand	Provides random numbers from the hardware random number generator.

Power	Access to POWER block configuration while the SoftDevice is enabled:
	• Access to RESETREAS register
	• Set power modes
	• Configure power fail comparator
	• Control RAM block power
	• Use general purpose retention register
	• Configure DC/DC converter state:
	  • 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 event.
PPI	Configuration interface for PPI channels and groups reserved for an application.
Concurrent Multiprotocol Timeslot API	Schedule other radio protocol activity, or periods of radio inactivity. Fo more information, see Concurrent Multiprotocol Timeslot API on page 29.
Radio Notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See Radio Notification on page 22.
Block Encrypt (ECB)	Safe use of 128 bit AES encrypt HW accelerator.

Feature	Description
Event API	Fetch asynchronous events generated by the SoC library.
Flash memory API	Application access to flash write, erase, and protect. Can be safely used during all protocol stack states. See Flash memory API on page 20.
Temperature	Application access to the temperature sensor.

Chapter 7
SoftDevice Manager

The SoftDevice control API enables the Application to manage the SoftDevice on a top level.

Table 12: Features enabling the Application to manage the SoftDevice on a top level.

Feature	Description
SoftDevice control API	Control of SoftDevice state through enable and disable. On enable, the low frequency clock source can be selected between the following options: • RC oscillator • Crystal oscillator

Chapter 8
SoftDevice information structure

The SoftDevice binary file contains an information structure.

The structure is illustrated in Figure 3: SoftDevice information structure on page 19. The location of the structure, the SoftDevice size, and the firmware_id can be obtained at run time by the application using macros defined in the `nrf_sdm.h` header file. Accessing this structure requires that the SoftDevice is not read back protected. The information structure can also be accessed by parsing the binary SoftDevice file.

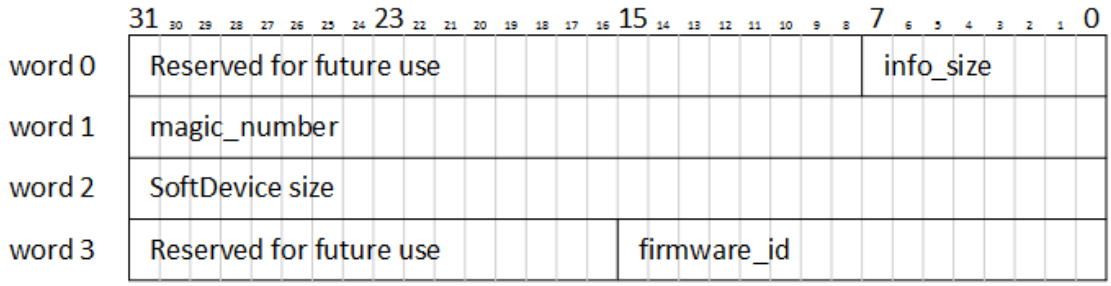


Figure 3: SoftDevice information structure

Chapter 9
Flash memory API

Asynchronous flash memory operations are performed using the SoC library API and provide the application with flash write, flash erase, and flash protect support through the SoftDevice. This interface can safely be used during active BLE connections.

The flash memory access is scheduled in between the protocol radio events. For short connection, advertisement or scan intervals, the time required for the flash memory access may be larger than the interval. In this case, protocol radio events may be skipped. The flash memory access may also be delayed to minimize the disturbance of the BLE radio protocol.

If the protocol radio events are in a certain critical state, flash memory access may get delayed for a long period resulting in the time-out event `NRF_EVT_FLASH_OPERATION_ERROR`. If this happens, retry the flash memory operation. Examples of typical critical phases of protocol radio events include: connection setup, connection update, disconnection, and just before supervision time-out.

The probability of successfully accessing the flash memory is higher when there is little BLE activity. For example, with long connection intervals there will be a higher probability of accessing flash memory successfully. Use the guidelines in Table 13: Behavior with BLE traffic and concurrent flash write/erase on page 20 to improve the probability of flash operation success.

Important:

Flash page erase takes approximately 22 ms and a 256 byte flash write takes approximately 13 ms.

Table 13: Behavior with BLE traffic and concurrent flash write/erase

BLE activity	Flash write/erase
High Duty cycle directed advertising	Does not allow flash operation while advertising is active (maximum 1.28 seconds). In this case, retrying flash operation will only succeed after the advertising activity has finished.
3 connections as a central, 1 connection as a peripheral, LE advertiser, and a scanner, all running concurrently. All active connections fulfill the following criteria: Supervision time-out > 14 x connection interval	Low to medium probability of flash operation success Probability of success increases with increase in connection interval and advertiser interval and decreases with increase in scan window.
3 connections as a central Scanner All active connections fulfill the following criteria: Supervision time-out > 6 x connection interval	Low to medium probability of flash operation success. Probability of success increases with increase in connection interval and advertiser interval and decreases with increase in scan window.
3 connections as a central 1 connection as a peripheral All active connections fulfill the following criteria: Supervision time-out > 6 x connection interval Connection interval ≥ 50 ms	High probability of flash write success. Low to medium probability of flash erase success. (high probability if the connection interval is > 60 ms)

BLE activity	Flash write/erase
Advertisement interval ≥ 50 ms All central connections have an equal connection interval	
3 connections as a central All active connections fulfill the following criteria: Supervision time-out > 6 x connection interval Connection interval ≥ 40 ms All connections have an equal connection interval	High probability of flash operation success.
1 connection as a peripheral All active connections fulfill the following criteria: Supervision time-out > 6 x connection interval	High probability of flash operation success.
Connectable Undirected Advertising Nonconnectable Advertising Scannable Advertising Connectable Low Duty Cycle Directed Advertising	High probability of flash operation success.
No BLE activity	Flash operation will always succeed.

Chapter 10
Radio Notification

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

The signal is sent using software interrupt, as specified in Table 27: Allocation of software interrupt vectors to SoftDevice signals on page 44.

In order to make sure that the Radio Notification signals behave in a consistent way, Radio Notification shall always be configured when the SoftDevice is in an idle state with no protocol stack or other SoftDevice activity in progress. It is therefore recommended to configure the Radio Notification signals directly after the SoftDevice has been enabled.

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

Because both ACTIVE and nACTIVE use the same software interrupt, it is up to the application to manage them. If both ACTIVE and nACTIVE are configured ON by the application, there will always be an ACTIVE signal before an nACTIVE signal.

For an explanation of the notation used in this section, see Table 14: Radio Notification figure labels on page 23.

When there is sufficient time between radio events, each radio event will have both an ACTIVE signal and an nACTIVE signal. This is the case when $t_{GAP} > t_{ndist}$. Figure 4: Two radio events with ACTIVE and nACTIVE signals on page 22 shows an example of this for two radio events. The figure also shows where the ACTIVE and nACTIVE signals will be placed.

If $t_{GAP} < t_{ndist}$, there is no sufficient time to have Radio Notification signals between the radio events, and the signals will be skipped. There will still be an ACTIVE signal before the first event, and an nACTIVE signal after the last event. This is shown in Figure 5: Two radio events where tGAP is too small and the notification signal will not be available between the events on page 23.

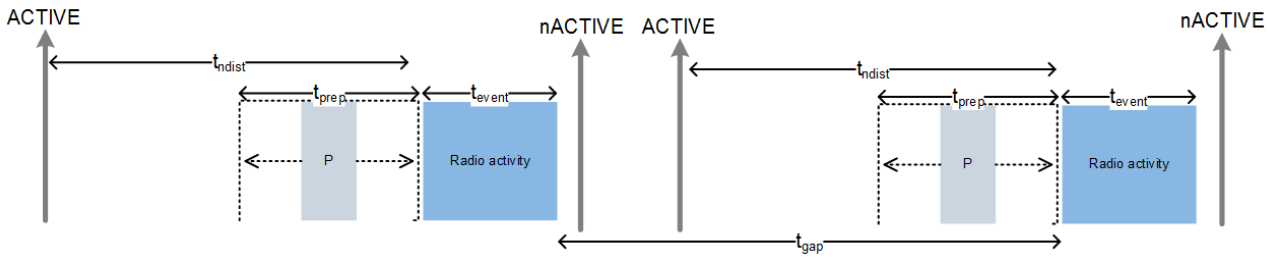


Figure 4: Two radio events with ACTIVE and nACTIVE signals

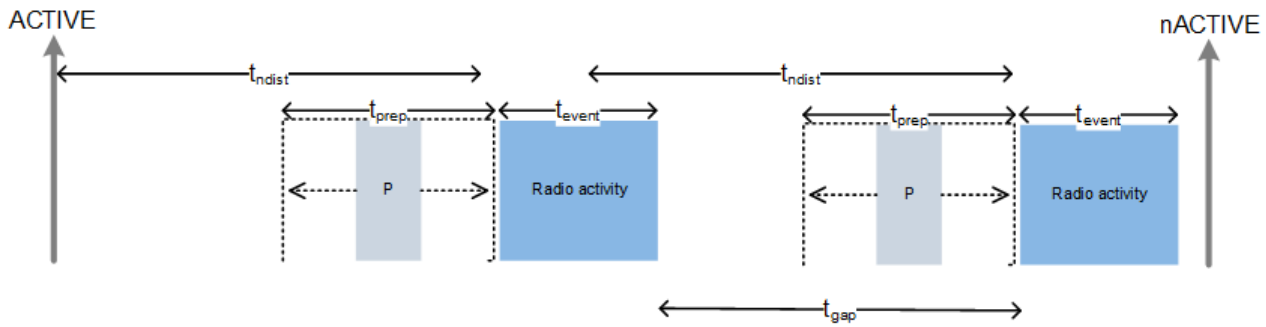


Figure 5: Two radio events where t_{GAP} is too small and the notification signal will not be available between the events

Table 14: Radio Notification figure labels

Label	Description	Notes
ACTIVE	The ACTIVE signal prior to a Radio Event.	
nACTIVE	The nACTIVE signal after a Radio Event.	Because both ACTIVE and nACTIVE use the same software interrupt, it is up to the application to manage them. If both ACTIVE and nACTIVE are configured ON by the application, there will always be an ACTIVE signal before an nACTIVE signal.
P	CPU processing in the lower stack interrupt between ACTIVE and RX.	The CPU processing may occur anytime, up to tprep before RX.
RX	Reception of packet.	
TX	Transmission of packet.	
t_{event}	The time used in a Radio Event.	
t_{gap}	The time between the end of one Radio Event and the start of another.	
t_{ndist}	The notification distance - the time between ACTIVE and first RX/TX in a Radio Event.	This time is configurable by the application developer.
t_{prep}	The time before first RX/TX to prepare and configure the radio.	The application will be interrupted by the LowerStack during t_{prep}. **Important:** All packet data to send in an event should be sent to the stack t_{prep} before the Radio starts.
t_P	Time used for preprocessing before the Radio Event.	
$t_{interval}$	Time between Radio Events as per the protocol.	

Label	Description	Notes
t_{EEO}	Time between central role Radio Events (Event-to-Event Offset).	The time between the start of adjacent connections, and between the last connection and the scanner. Some or all connections and/or the scanner may be idle.

Table 15: BLE Radio Notification timing ranges

Value	Range (µs)
t_{ndist}	800, 1740, 2680, 3620, 4560, 5500 (Configured by the application)
t_{event}	2750 to 5500 - Undirected and scannable advertising, 0 to 31 byte payload, 3 channels 2150 to 2950 - Non-connectable advertising, 0 to 31 byte payload, 3 channels 1.28 seconds - Directed advertising, 3 channels 1000 to 3500 Slave - 1 to 3 packets RX and TX unencrypted data when connected 1000 to 3700 Slave - 1 to 3 packets RX and TX encrypted data when connected 2500 to 10.24 seconds – Scanner running. Depends on scan window. 1200 to 1690 Master - 1 packets RX and TX unencrypted data when connected 1200 to 1760 Master - 1 packets RX and TX encrypted data when connected
t_{prep}	165 to 1550
t_P	≥ 150
t_{EEO}	2000 to 2250

Using the numbers from Table 15: BLE Radio Notification timing ranges on page 24, the amount of CPU time available between the ACTIVE signal and a Radio Event is:

$t_{ndist} - t_P$

The following equation shows the amount of time before stack prepare interrupt after ACTIVE signal. Data packets must be transferred to the stack using the API within this time from the ACTIVE signal if they are to be sent in the next Radio Event.

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

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

10.1 Radio Notification on connection events as a central

This section illustrates the radio Notification signal in relation to different combinations of active links and scanning events.

See Table 14: Radio Notification figure labels on page 23 for a description of the notations used in text and figures and Multilink scheduling on page 46 to understand the scheduling of roles.

To ensure the notification signal is available to the application at the configured time when a single link is established as a central, the following rule must be followed:

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

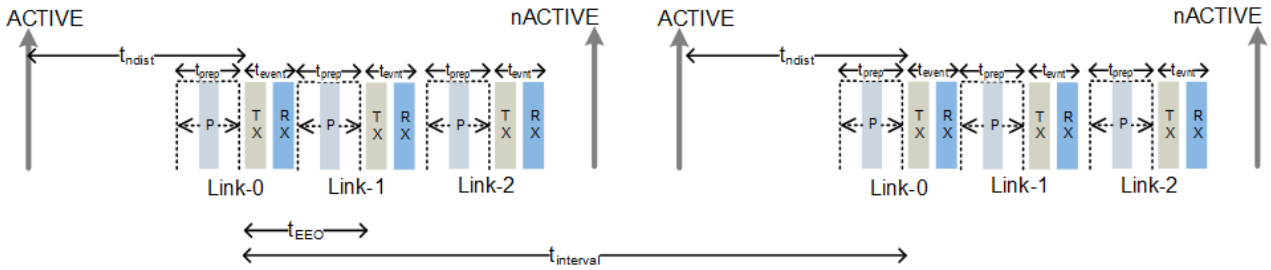
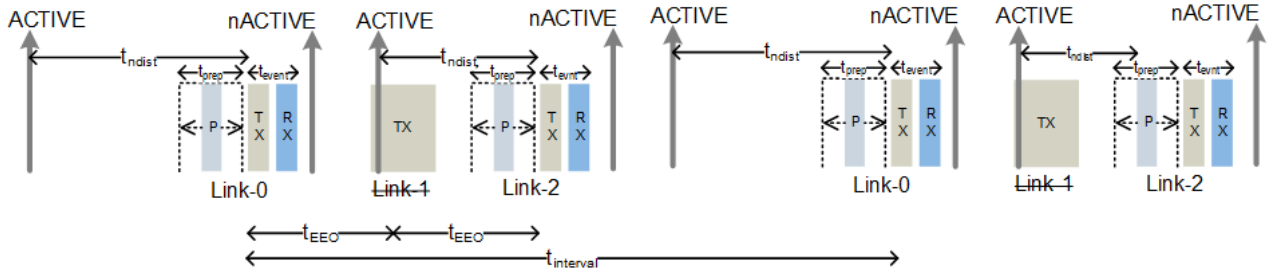


Figure 6: BLE Radio Notification signal in relation to 3 active links

To ensure the notification signal is available to the application at the configured time when 3 links are established as a central, the following rule must be followed:

$t_{ndist} + 3 \times t_{EEO} < t_{interval}$

Figure 7: BLE Radio Notification signal when the number of active links as a central is 2



To ensure the notification signal is available to the application in the gap left by inactive links as a central, the gap should be greater than t_{ndist}. This can be expressed as (where $n_{inactive}$ is the number of consecutive inactive links as a central):

$n_{inactive} \times t_{EEO} > t_{ndist}$

For example, the case shown in Figure 5 where link-1 is not connected, a gap of t_{EEO} exists between two links as a central, so active signal will come if:

$t_{EEO} > t_{ndist}$

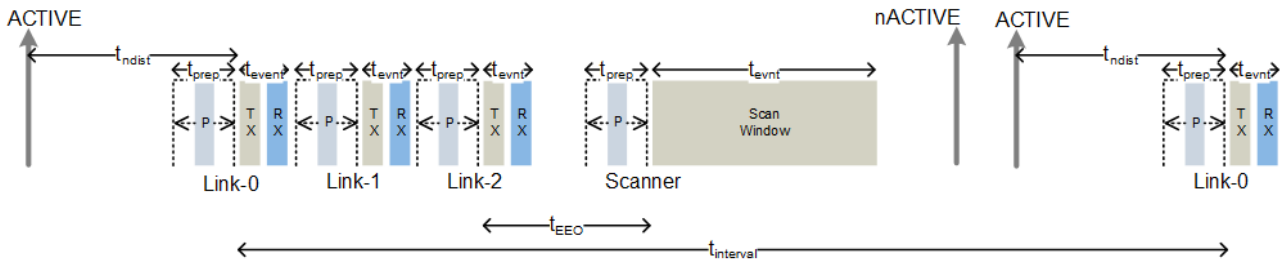


Figure 8: BLE Radio Notification signal in relation to 3 active connections as a central and running scanner

To ensure the notification signal is available to the application at the configured time with 3 links as a central established and a scanner started, the following rule must be followed:

$t_{ndist} + 4x\ t_{EEO} + Scan\_window < t_{interval}$

10.2 Radio Notification on peripheral events

For the peripheral role, many packets can be sent and received in one Radio Event.

Radio Notification events will be as shown in Figure 9: BLE Radio Notification, multiple packet transfers on page 26.

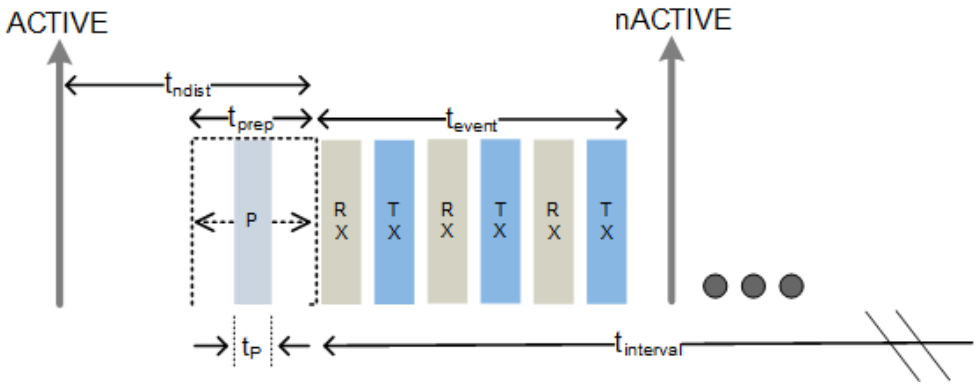


Figure 9: BLE Radio Notification, multiple packet transfers

To ensure the notification signal is available to the application at the configured time when a single slave link is established, the SoftDevice enforces the following rule (with one exception, see Table 16: Maximum peripheral packet transfer per BLE Radio Event for given combinations of tndist and tinterval on page 27):

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

The stack will limit the length of a Radio Event (tevent), thereby reducing the maximum packets exchanged, to accommodate the selected t_{ndist}. Figure 10: Consecutive Radio Events with BLE Radio Notification on page 27 shows consecutive Radio Events with Radio Notification and illustrates the limitation in t_{event} which may be required to ensure t_{ndist} is preserved.

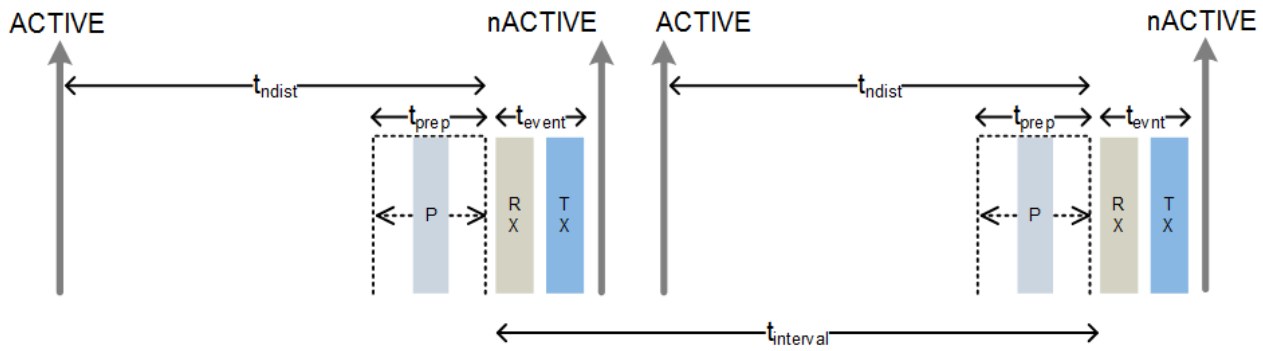


Figure 10: Consecutive Radio Events with BLE Radio Notification

Table 16: Maximum peripheral packet transfer per BLE Radio Event for given combinations of tndist and tinterval on page 27 shows the limitation on the maximum number of full length packets which can be transferred per Radio Event given a t_{ndist} and $t_{interval}$ combination.

Table 16: Maximum peripheral packet transfer per BLE Radio Event for given combinations of t_{ndist} and $t_{interval}$

t_{ndist}	$t_{interval}$		
	7.5 ms	10 ms	≥ 15 ms
800	3	3	3
1740	3	3	3
2680	3	3	3
3620	3	3	3
4560	2	3	3
5500	0[1]	3	3

10.3 Radio notification with concurrent peripheral and central events

A link as a peripheral can be placed at any time relative to links as a central. Depending on how close the link as a peripheral is to the links as a central, the notification signal might not be available to the application.

Figure 11: Example: the gap between the links as a central and the peripheral is too small to trigger the notification signal on page 28 shows an example where the gap between the links as a central and the peripheral is too small to trigger the notification signal.

[1] Radio notifications may be suppressed with the longest t_{ndist} combined with a 7.5 ms connection interval.

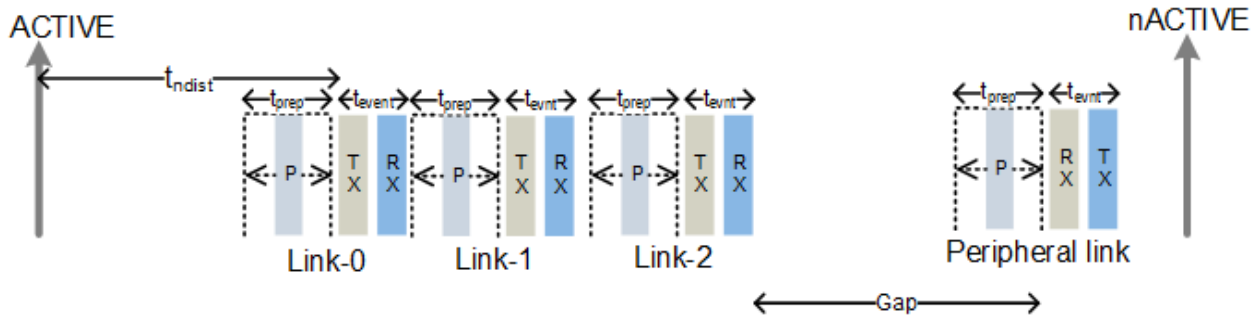


Figure 11: Example: the gap between the links as a central and the peripheral is too small to trigger the notification signal

If the following condition is met

$t_{GAP} > t_{ndist}$

the notification signal will come, as illustrated below.

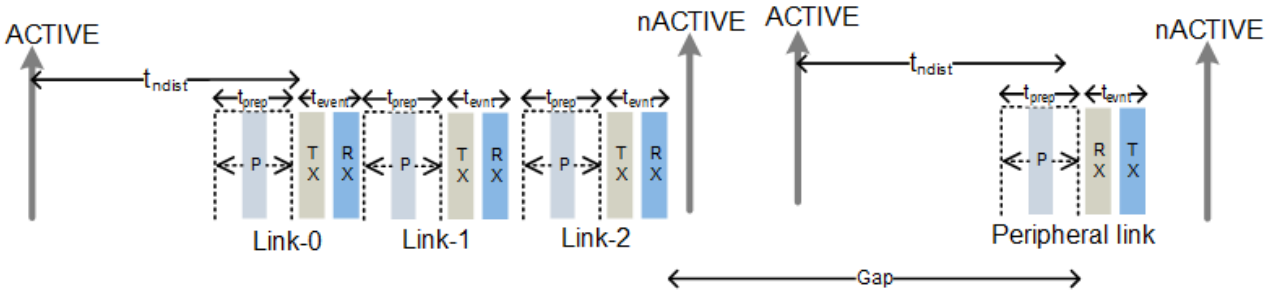


Figure 12: Example: the gap between the links as a central and the peripheral is sufficient to trigger the notification signal

Chapter 11
Concurrent Multiprotocol Timeslot API

The Multiprotocol Timeslot API allows an application developer to safely schedule 2.4 GHz proprietary radio usage while the SoftDevice protocol stack is in use by the device.

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

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

The Timeslot feature is part of the SoC library. The feature works by having the SoftDevice time-multiplex access to peripherals between the application and itself. Through the SoC API, the application can open a Timeslot session and request timeslots. When a timeslot is granted, the application has exclusive and real-time access to the normally blocked RADIO, TIMER0, CCM, AAR, and PPI (channels 14 – 15) peripherals and can use these freely for the length of the timeslot, see Table 25: Hardware access type definitions on page 42 and Table 26: Peripheral protection and usage by SoftDevice on page 42.

11.1 Request types

There are two types of 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. In this case, the application specifies in the request when the timeslot should start and the time is measured from the start of the previous timeslot.

The application may also request to extend an ongoing timeslot. Extension requests may be repeated, prolonging the timeslot even further.

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

11.2 Request priorities

Timeslots can be requested at either high or normal priority, indicating how important it is for the application to access the specified peripherals.

Using normal priority should be considered best practice to minimize the influence of the use of the Multiprotocol Timeslot API on other activities. The high priority should only be used when required, such as for running a radio protocol with certain timing requirements that are not met using normal priority.

11.3 Timeslot length

A timeslot is requested at a given length, but may be extended.

The length of the timeslot is specified by the application in the request and ranges from 100 µs to 100 ms. Longer continuous timeslots can be achieved by requesting to extend the current timeslot. Successive extensions will give a timeslot as long as possible within the limits set by other SoftDevice activities, up to a maximum of 128 s.

11.4 Scheduling

Timeslots requested by the application are scheduled within the SoftDevice along with the SoftDevice protocol and the Flash API activities.

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 time the timeslot is wanted, the priority of the request, and the length of the requested timeslot. If the requested timeslot does not collide with other activities, the request will be granted and the timeslot scheduled. If the requested timeslot collides with an already scheduled activity with equal or higher priority, the request will be blocked. If a later arriving activity of higher priority causes a collision, the request will be canceled and the scheduled timeslot revoked.

However, a timeslot that has already started cannot be interrupted or canceled. Timeslots requested at high priority will cancel other activities scheduled at lower priorities in case of a collision. 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 Multiprotocol Timeslot interface as for SoftDevice internal activities. See section Radio Notification on page 22 for more information. If Radio Notifications are enabled, Multiprotocol Timeslots will be notified.

11.5 Performance considerations

Since the Multiprotocol Timeslot API shares core peripherals with the SoftDevice, and are scheduled along with other SoftDevice activities, use of the Timeslot feature may influence SoftDevice performance.

Therefore the application configuration of the SoftDevice protocol should be considered when using the Multiprotocol Timeslot API.

In general, all timeslot requests should use the lowest priority to ensure that interruptions to other activity is minimized. In addition, timeslots should be kept as short as possible in order to minimize the impact on the overall performance of the device. Similarly, requesting a shorter timeslot and then extending it gives more flexibility to schedule other activities than requesting a longer timeslot.

11.6 Multiprotocol timeslot API

This section describes the calls, events, signals, and return actions of the Multiprotocol timeslot API.

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

> **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.

11.6.1 API calls

These are the API calls defined for the S130 SoftDevice:

Table 17: API calls

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

11.6.2 Timeslot events

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

Events are received in the application event handler callback function, which will typically be run in App(L) priority, see BLE peripheral performance on page 56. The following events are defined:

Table 18: Timeslot events

Event	Description
NRF_EVT_RADIO_SESSION_IDLE	Session status: The current timeslot session has no remaining scheduled timeslots.
NRF_EVT_RADIO_SESSION_CLOSED	Session status: The timeslot session is closed and all acquired resources are released.
NRF_EVT_RADIO_BLOCKED	Timeslot status: The last requested timeslot could not be scheduled, due to a collision with already scheduled activity or for other reasons.
NRF_EVT_RADIO_CANCELED	Timeslot status: The scheduled timeslot was preempted by higher priority activity.
RF_EVT_RADIO_SIGNAL_ CALLBACK_INVALID_RETURN	Signal handler: The last signal hander return value contained invalid parameters.

11.6.3 Timeslot signals

Signals come from the peripherals and arrive within a timeslot.

Signals are received in a signal handler callback function that the application must provide. The signal handler runs in LowerStack priority, which is the highest priority in the system, see section Processor availability on page 55.

Table 19: Timeslot signals

Signal	Description
NRF_RADIO_CALLBACK_SIGNAL_ TYPE_START	Start of the timeslot. The application now has exclusive access to the peripherals for the full length of the timeslot.
NRF_RADIO_CALLBACK_SIGNAL_ TYPE_RADIO	Radio interrupt, for more information, see chapter 2.4 GHz radio (RADIO) in the nRF51 Reference Manual.
NRF_RADIO_CALLBACK_SIGNAL_ TYPE_TIMER0	Timer interrupt, for more information, see chapter Timer/counter (TIMER) in the nRF51 Reference Manual.

Signal	Description
NRF_RADIO_CALLBACK_SIGNAL_ TYPE_EXTEND_SUCCEEDED	The latest extend action succeeded.
NRF_RADIO_CALLBACK_SIGNAL_ TYPE_EXTEND_FAILED	The latest extend action failed.

11.6.4 Signal handler return actions

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

Table 20: 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.

11.6.5 Ending a timeslot in time

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

For these purposes, the application is granted access to the TIMER0 peripheral for the length of the timeslot. This timer is started from zero by the SoftDevice at the start of the timeslot, and is configured to run at 1 MHz. The recommended practice is to set up a timer interrupt that expires before the timeslot expires, with enough time left of the timeslot to do any clean-up actions before the timeslot ends. Such a timer interrupt can also be used to request an extension of the timeslot, but there must still be enough time to clean up if the extension is not granted.

11.6.6 The signal handler runs at LowerStack priority

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

As for the App(H) interrupt, SVC calls are not available in the signal handler. It is a requirement that processing in the signal handler does not exceed the granted time of the timeslot. If it does, the behavior of the SoftDevice is undefined and the SoftDevice may malfunction.

The signal handler may be called several times during a timeslot. It is recommended to use the signal handler only for the real time signal handling. When a signal has been handled, exit the signal handler to wait for the next signal. Processing other than signal handling should be run at lower priorities, outside of the signal handler.

11.7 Timeslot usage examples

Several timeslot usage examples are provided with descriptions of the sequence of events within them.

11.7.1 Complete session example

This section describes a complete timeslot session.

Figure 13: Complete session example on page 33 shows a complete timeslot session. In this case, only timeslot requests from the application are being scheduled, there is no SoftDevice activity.

At start, the application calls the API to open a session and to request a first timeslot (which must be of type earliest). The SoftDevice schedules the timeslot. At the start of the timeslot, the SoftDevice calls the application signal hander with the START signal. After this, the application is in control and has access to the peripherals. The application will then typically set up TIMER0 to expire before the end of the timeslot, to get a signal that the timeslot is about to end. In the last signal in the timeslot, the application uses the signal handler return action to request a new timeslot 100 ms after the first.

The following timeslots (the middle timeslot in the figure below) are all similar. The signal handler is called with the START signal at the start of the timeslot. The application then has control, but must arrange for a signal to come towards the end of the timeslot. As the return value for the last signal in the timeslot, the signal handler requests a new timeslot using the REQUEST_AND_END action.

Eventually, the application does not require the radio any more. So, at the last signal in the last timeslot, the application returns END from the signal handler. The SoftDevice then sends an IDLE event to the application event handler. The application calls session_close, and the SoftDevice sends the CLOSED event. The session has now ended.

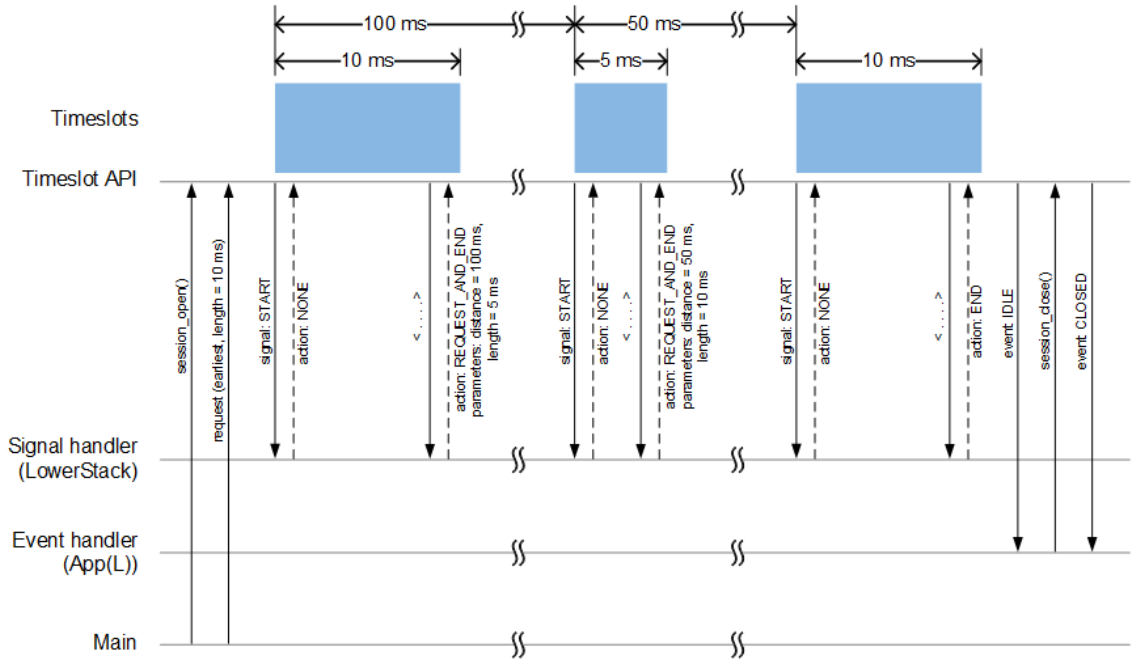


Figure 13: Complete session example

11.7.2 Blocked timeslot example

Situations may occur where a new timeslot cannot be scheduled as requested because of a collision with an already scheduled SoftDevice activity.

Figure 14: Blocked timeslot example on page 34 shows a situation in the middle of a session where a requested timeslot cannot be scheduled. At the end of the first timeslot illustrated here, the application signal handler returns a `REQUEST_AND_END` action to request a new timeslot. The new timeslot cannot be scheduled as requested, because of a collision with an already scheduled SoftDevice activity. The application is notified about this by a `BLOCKED` event to the application event handler. The application then makes a new request further out in time. This request succeeds (it does not collide with anything), and a new timeslot is scheduled.

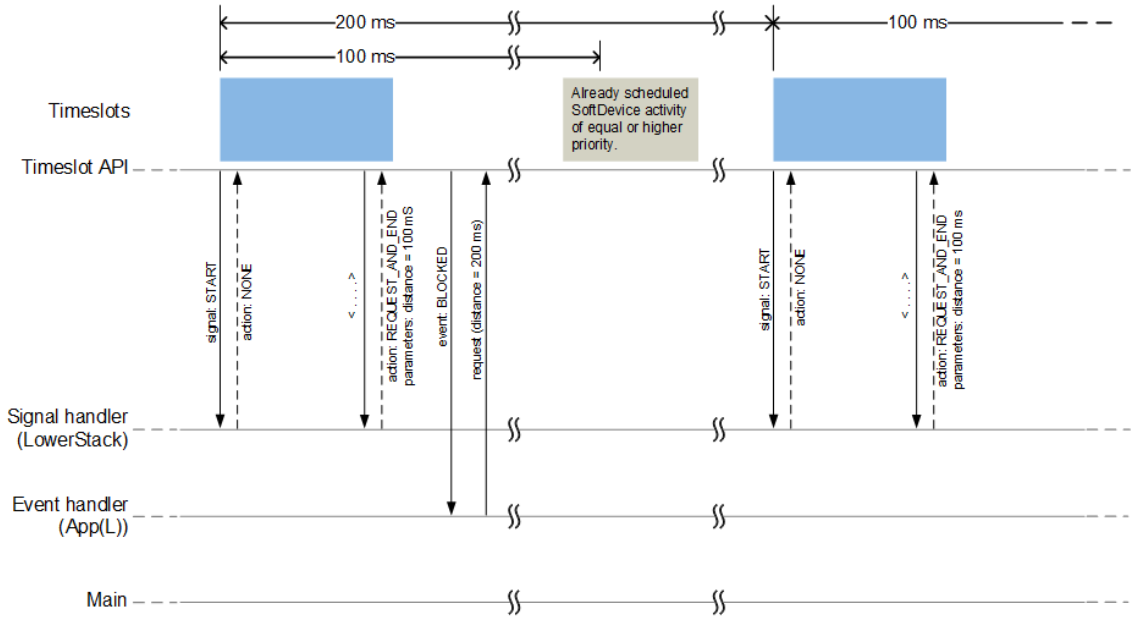


Figure 14: Blocked timeslot example

11.7.3 Canceled timeslot example

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

Figure 15: Canceled timeslot example on page 35 shows a situation in the middle of a session where a requested and scheduled application timeslot is being revoked. The upper part of the figure shows that the application has ended a timeslot by returning the `REQUEST_AND_END` action, and the new timeslot has been scheduled. The new scheduled timeslot has not been started yet, it is still some time into the future. The lower part of the figure shows the situation some time later.

In the meantime, time for a SoftDevice activity of higher priority has been requested internally in the SoftDevice, at a time which collides with the scheduled application timeslot. To accommodate the higher priority request, the application timeslot has been removed from the schedule, and the higher priority SoftDevice activity scheduled instead. The application is notified about this by a `CANCELED` event to the application event handler. The application then makes a new request further out in time. This request succeeds (it does not collide with anything), and a new timeslot is scheduled.

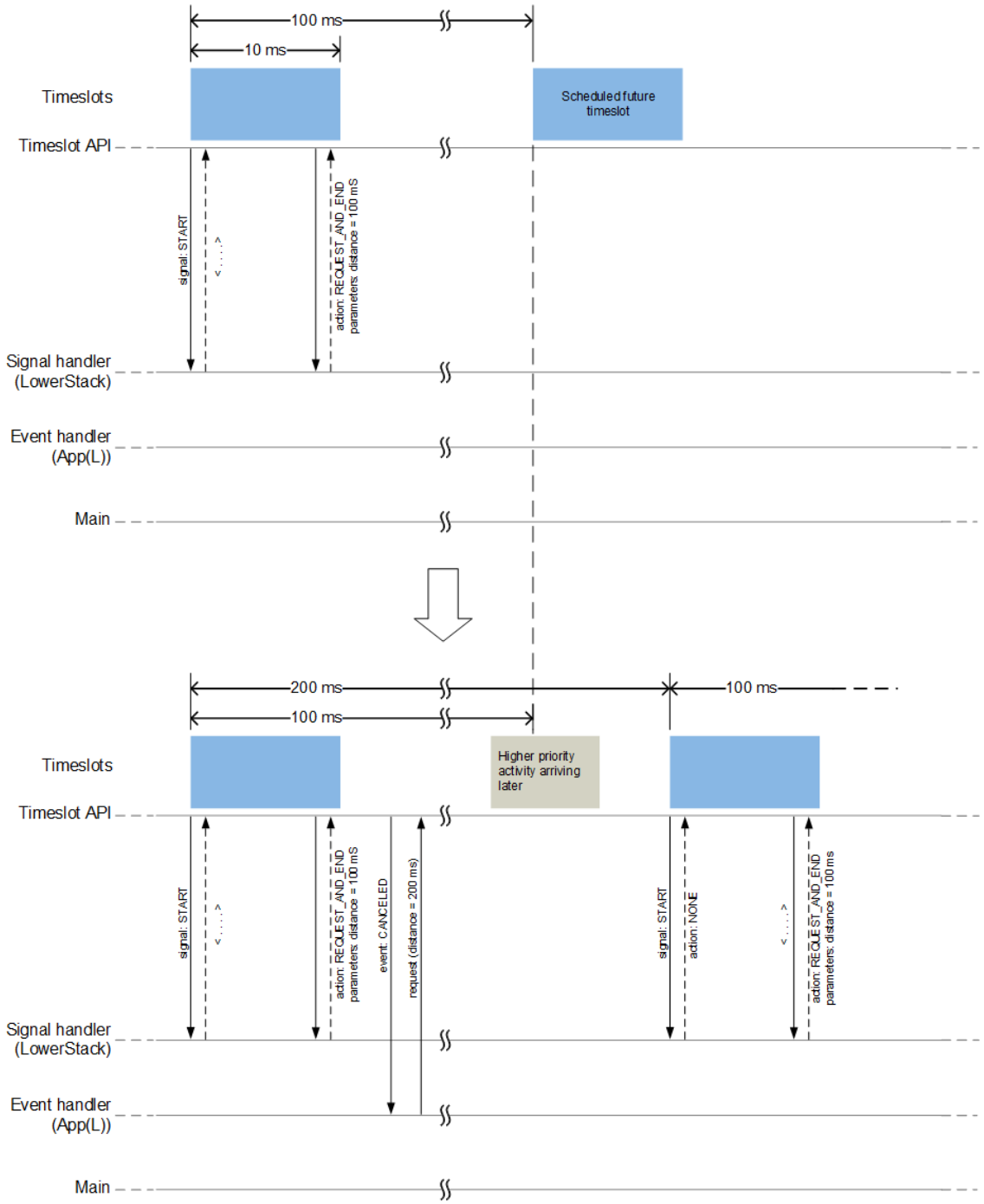


Figure 15: Canceled timeslot example

11.7.4 Timeslot extension example

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

In the first slot in Figure 16: Timeslot extension example on page 36, the application uses the signal handler return action to request an extension of the timeslot. The extension is granted, and the timeslot is seamlessly prolonged. The second attempt at extending the timeslot fails, as a further extension would cause a collision with a SoftDevice activity that has been scheduled. Therefore the application does a new request, of type earliest. This results in a new radio timeslot being scheduled immediately after the SoftDevice activity. This new timeslot can be extended a number of times.

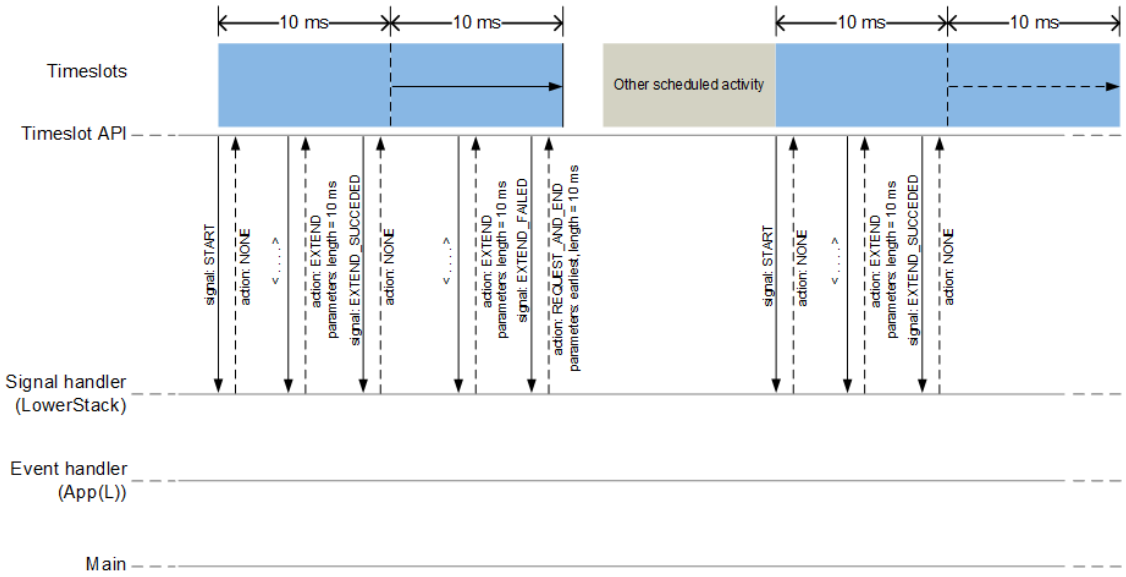


Figure 16: Timeslot extension example

Chapter 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 iC.

The SoftDevice also contains a Master Boot Record (MBR). 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 Master Boot Record (MBR) module occupies a defined region in flash memory where the System Vector table resides.

All exceptions (reset, hard fault, interrupts, SVC) are processed first by the MBR and then forwarded to appropriate handlers (for example bootloader or SoftDevice). The main feature of the MBR is to provide an interface to allow in-system updates of the SoftDevice and bootloader firmware.

The MBR is not updated between versions of the SoftDevice, meaning that during an update process, the MBR is never erased. The MBR ensures safe restart of any ongoing update process if an unexpected reset occurs.

12.2 Bootloader

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

The bootloader has access to the full SoftDevice API and can be implemented just as any application that uses a SoftDevice. In particular, the bootloader can make use of the SoftDevice API to enable protocol stack interaction.

The bootloader is supported in the SoftDevice architecture by using a configurable base address for the bootloader in application code space. 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 SoftDevice. If an unexpected reset occurs during an update of the SoftDevice, it is the responsibility of the bootloader to detect this and recover.

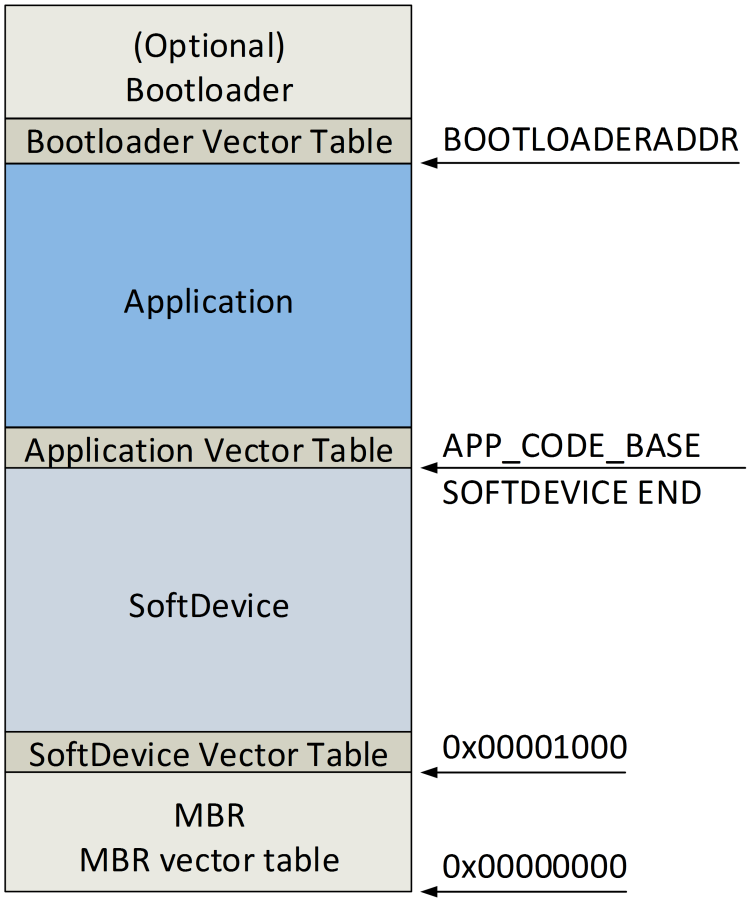


Figure 17: MBR, SoftDevice and bootloader architecture

12.3 Master Boot Record (MBR) and SoftDevice reset behavior

Upon system reset, the MBR Reset Handler is run as specified by the System Vector table.

The MBR and SoftDevice reset behavior is as follows:

- If an in-system bootloader update procedure is in progress:
  - Then in-system update procedure is run to completion.
  - System is reset.
- Else if `SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET` has been called previously:
  - Forward interrupts to the parameter given.
  - Run from Reset Handler (defined in vector table at parameter given).
- 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 SoftDevice.
  - Run SoftDevice Reset Handler (defined in SoftDevice vector table at 0x00001000).
  - In this case, `APP_CODE_BASE` is hardcoded inside the SoftDevice.
  - SoftDevice run Application Reset Handler (defined in application vector table at `APP_CODE_BASE`).
- Else system startup error:
  - Sleep forever.

12.4 Master Boot Record (MBR) and SoftDevice initialization

The SoftDevice can be enabled by the bootloader.

The SoftDevice can be enabled by the bootloader in the following in this order:

1. Issue a command for MBR to forward interrupts to the SoftDevice using `sd_mbr_command()` with `SD_MBR_COMMAND_INIT_SD`.
2. Issue 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. Enable the SoftDevice using `sd_softdevice_enable()`.

For a bootloader to transfer execution from itself to the application, you can do the following:

1. If interrupts have not been forwarded to SoftDevice, issue a command for MBR to forward interrupts to SoftDevice using `sd_mbr_command()` with `SD_MBR_COMMAND_INIT_SD`.
2. Ensure that the SoftDevice is disabled using `sd_softdevice_disable()`.
3. Issue 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. Branch to the application's reset handler after reading the handler from the Application Vector Table.

Chapter 13
System on Chip resource requirements

This section describes how the MBR and SoftDevice use resources. The SoftDevice requirements are shown both when enabled and disabled.

The SoftDevice and MBR are designed to be installed on a System on Chip (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 IC including RAM and hardware peripherals like the radio.

13.1 Attribute Table size

The size of the Attribute Table can be configured through the SoftDevice API when initializing the Bluetooth low energy stack.

The amount of RAM reserved by the SoftDevice, and thereby the amount of RAM available for the application, is dependent upon this configuration.

The Attribute Table size (ATTR_TAB_SIZE) has a default value of 0x600 bytes.

Applications that require an Attribute Table smaller or bigger than the default one can choose to either reduce or increase the Attribute table size. The amount of RAM reserved by the SoftDevice, and the start address for the application RAM (APP_RAM_BASE) will then change accordingly. The application linker configuration must be adapted to reflect the changed SoftDevice RAM requirement.

For more information on how to configure the Attribute Table size, refer to the *SoftDevice API*.

13.2 Memory resource map and usage

The memory map for program memory and RAM at run time with the SoftDevice enabled is illustrated in this section.

Memory resource requirements, both when the SoftDevice is enabled and disabled, are shown in section Memory resource requirements on page 41.

> **Important:** The definitions of Region 0 (R0) and Region 1 (R1) are valid only when the CLENR0 and RLENR0 registers are optionally programmed to enable memory protection. See the MPU chapter in the nRF51*Reference Manual* for more details.

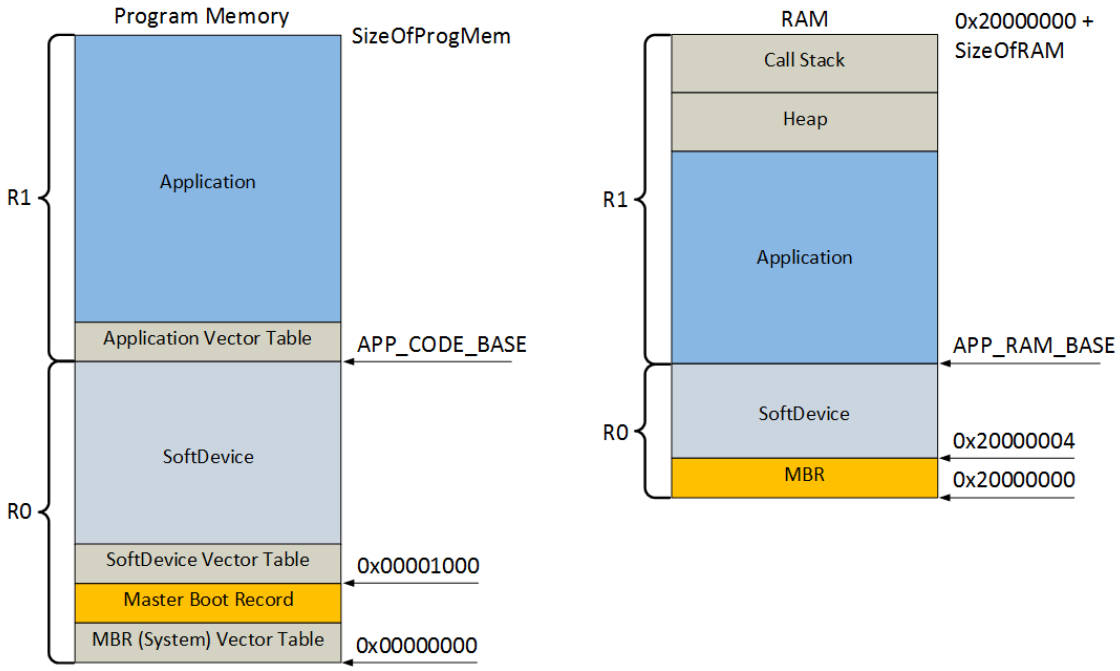


Figure 18: Memory resource map

13.2.1 Memory resource requirements

Listed below are the memory resource requirements both when the S130 SoftDevice is enabled and disabled.

Table 21: S130 Memory resource requirements for flash

Flash	S130 Enabled	S130 Disabled
SoftDevice	108 kB[2]	108 kB
MBR	4 kB	4 kB
APP_CODE_BASE	0x0001C000	0x0001C000

Table 22: S130 Memory resource requirements for RAM

RAM	S130 Enabled	S130 Disabled
SoftDevice	0x2200 - 4 + ATTR_TAB_SIZE Default: 10236 (0x2200 - 4 + 0x600) Minimum: 8916 (0x2200 - 4 + 216)	4 bytes
MBR	4 bytes	4 bytes
APP_RAM_BASE	0x20002200 + ATTR_TAB_SIZE[3] Default: 0x20002800 (0x20002200 + 0x600) Minimum: 0x200022D8 (0x20002200 + 0xD8)	0x20000008

[2] 1 kB = 1024 bytes
[3] See section Attribute Table size on page 40.

Table 23: S130 Memory resource requirements for call stack

Call stack[4]	S130 Enabled	S130 Disabled
Maximum usage	1536 bytes (0x600)	0 bytes

Table 24: S130 Memory resource requirements for heap

Heap	S130 Enabled	S130 Disabled
Maximum allocated bytes	0 bytes	0 bytes

13.3 Hardware blocks and interrupt vectors

SoftDevice access types are used to indicate the availability of hardware blocks to the application. The access the application varies per hardware block, both when the SoftDevice is enabled and disabled.

Table 25: Hardware access type definitions

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

Table 26: Peripheral protection and usage by SoftDevice

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
0	0x40000000	MPU	Restricted	Open
0	0x40000000	POWER	Restricted	Open
0	0x40000000	CLOCK	Restricted	Open
1	0x40001000	RADIO	Blocked	Open
2	0x40002000	UART0	Open	Open
3	0x40003000	SPI0 / TWI0	Open	Open
4	0x40004000	SPI1/TW1/SPIS1	Open	Open
...				
6	0x40006000	GPIOTE	Open	Open

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
7	0x40007000	ADC	Open	Open
8	0x40008000	TIMER0	Blocked[5]	Open
9	0x40009000	TIMER1	Open	Open
10	0x4000A000	TIMER2	Open	Open
11	0x4000B000	RTC0	Blocked	Open
12	0x4000C000	TEMP	Restricted	Open
13	0x4000D000	RNG	Restricted	Open
14	0x4000E000	ECB	Restricted	Open
15	0x4000F000	CCM	Blocked[6]	Open
15	0x4000F000	AAR	Blocked[7]	Open
16	0x40010000	WDT	Open	Open
17	0x40011000	RTC1	Open	Open
18	0x40012000	QDEC	Open	Open
19	0x40013000	LPCOMP	Open	Open
20	0x40014000	Software interrupt	Open	Open
21	0x40015000	Radio Notification	Restricted[8]	Open
22	0x40016000	SoC Events	Blocked	Open
23	0x40017000	Software interrupt	Blocked	Open
24	0x40018000	Software interrupt	Blocked	Open
25	0x40019000	Software interrupt	Blocked	Open
...				
30	0x4001E000	NVMC	Restricted	Open
31	0x4001F000	PPI	Open[9]	Open
NA	0x50000000	GPIO P0	Open	Open

5 Available to the application in Multiprotocol Timeslot API timeslots, see Concurrent Multiprotocol Timeslot API on page 29.

6 Available to the application in Multiprotocol Timeslot API timeslots, see Concurrent Multiprotocol Timeslot API on page 29.

7 Available to the application in Multiprotocol Timeslot API timeslots, see Concurrent Multiprotocol Timeslot API on page 29.

8 Blocked only when radio notification signal is enabled. See Application signals – software interrupts (SWI) on page 44 for software interrupt allocation.

9 See section Programmable Peripheral Interconnect (PPI) on page 44 for limitations on the use of PPI when the SoftDevice is enabled.

ID	Base address	Instance	Access SoftDevice enabled	Access SoftDevice disabled
NA	0xE000E100	NVIC	Restricted[10]	Open

13.4 Application signals – software interrupts (SWI)

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

Table 27: 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	LowerStack processing - not user configurable.
5	25	UpperStack signaling - not user configurable.

13.5 Programmable Peripheral Interconnect (PPI)

PPI may be configured using the PPI API in the SoC library.

This API is available both when the SoftDevice is disabled and when it is enabled. It is also possible to configure the PPI using the Cortex Microcontroller Software Interface Standard (CMSIS) directly when the SoftDevice is disabled.

When the SoftDevice is disabled, all PPI channels and groups are available to the application. When the SoftDevice is enabled, some PPI channels and groups, as described in the table below, are in use by the SoftDevice.

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

Table 28: Assigning PPI channels between the application and SoftDevice

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 - 13	Channels 0 - 15
SoftDevice	Channels 14 - 15[11]	-

[10] 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.

[11] Available to the application in Multiprotocol Timeslot API timeslots, see Concurrent Multiprotocol Timeslot API on page 29.

Table 29: Assigning preprogrammed channels between the application and SoftDevice

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

Table 30: Assigning PPI groups between the application and SoftDevice

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

13.6 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 31: SVC number allocation

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

13.7 External requirements

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

The external clock crystal and other related components must be chosen accordingly. Data for the device XOSC input can be found in the product specification for the device.

Chapter 14
Multilink scheduling

The S130stack supports up to three connections as a central, up to one connection as a peripheral, an advertiser or broadcaster and an Observer or Scanner simultaneously.

An Initiator can only be started if there are less than three connections established as a central. Similarly, a connectable advertiser can only be started if there is no connection as a peripheral established.

The link scheduling system in the SoftDevice uses a slot based mechanism for Central role events. Advertiser and broadcaster events are scheduled as early as possible. Connection events as a peripheral follow the timings dictated by the connected peer. Peripheral role events and central role events are scheduled independently and so may occur at the same time and collide.

If roles/activities collide, their scheduling is determined by a priority system. If role A needs the radio at a time that overlaps with role B, and role A has higher priority, role A will get the event. Role B will be blocked from the event and its event will be rescheduled for a later time. If both role A and role B have same priority, the role which requested the event first will get the event.

The different roles have different priorities at different times, dependent upon their state. Table 32: Scheduling priorities on page 46 summarizes the priorities:

Table 32: *Scheduling priorities*

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

Priority (Decreasing order)	Role state
Last priority	• Flash access • Concurrent Multiprotocol Timeslot with normal priority

As an example, if a connection as a peripheral is close to its supervision time-out it will block all other roles and get the events it requests. In this case all other roles will be blocked if they overlap with the connection event, and they will lose their events.

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

14.1 Connection timing as a central

Link events as a central are added relative to the first connected link as a central.

Figure 19: Multilink scheduling - one or more connections as a central, factored intervals on page 47 shows a scenario where there are two links as a central established. C0 events correspond to the first connection as a central made and C1 events correspond to the second connection made. C1 events are initially offset from C0 events by t_{EEO} milliseconds. C1 events, in this example, have exactly double the connection interval of C0 events (the connection intervals have a common factor which is “connectionInterval 0”), so the events remain forever offset by t_{EEO} ms.

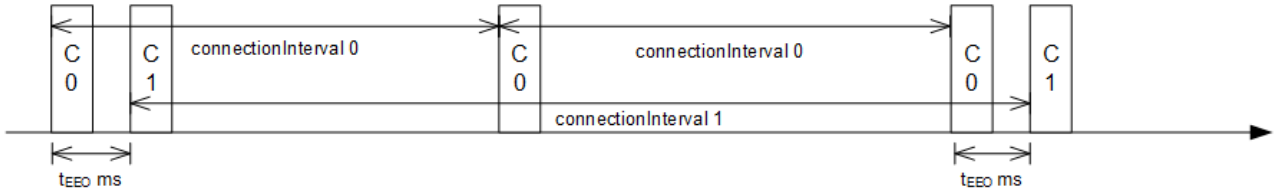


Figure 19: Multilink scheduling - one or more connections as a central, factored intervals

In Figure 20: Multilink scheduling - one or more connections as a central, unfactored intervals on page 47 the connection intervals do not have a common factor. This connection parameter configuration is possible, though this will result in dropped packets when events overlap. In this scenario, the second event shown for C1 is dropped because it collides with the C0 event.

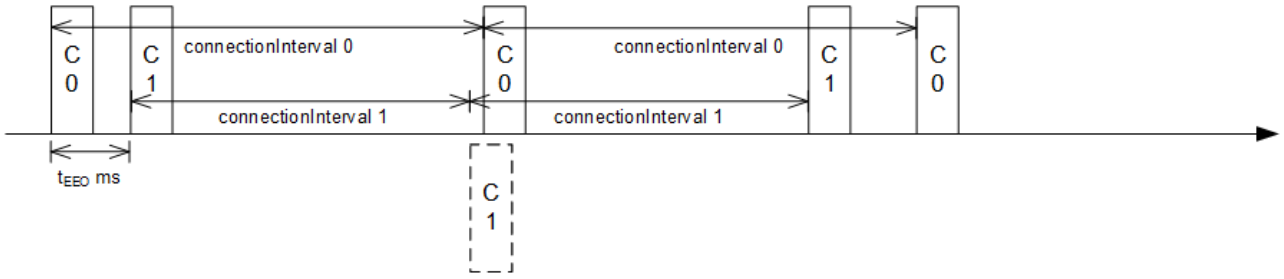


Figure 20: Multilink scheduling - one or more connections as a central, unfactored intervals

Table 33: *Multilink central role timing ranges*

Value	Description	Range (µs)
t_{EEO}	Refer to Table 14: Radio Notification figure labels on page 23.	Refer to Table 15: BLE Radio Notification timing ranges on page 24.
$t_{ScanReserved}$	Reserved time needed by the SoftDevice for each ScanWindow	1000

Figure 21: Multilink scheduling with maximum connections as a central and minimum interval on page 48 shows the maximum number of links as a central possible at a time (3) with the minimum connection interval possible without having event collisions and dropped packets (7.5 ms). In this case, all available event time is used for the links as a central.

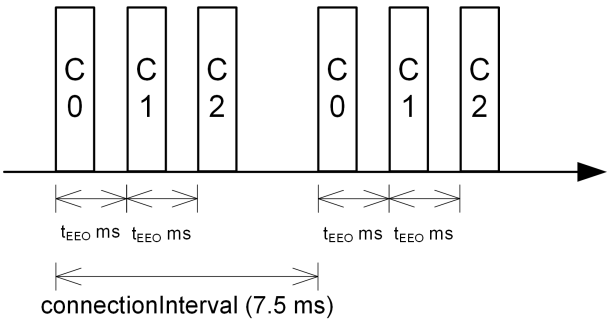


Figure 21: Multilink scheduling with maximum connections as a central and minimum interval

Figure 22: Multilink scheduling with maximum connections as a central and interval > min on page 48 shows a scenario where the connInterval is longer than the minimum, and Central 1 has been disconnected or does not have an event in this time period. It shows idle event time for each connection interval, and the remaining connections as a central maintain their timing offsets without the other links.

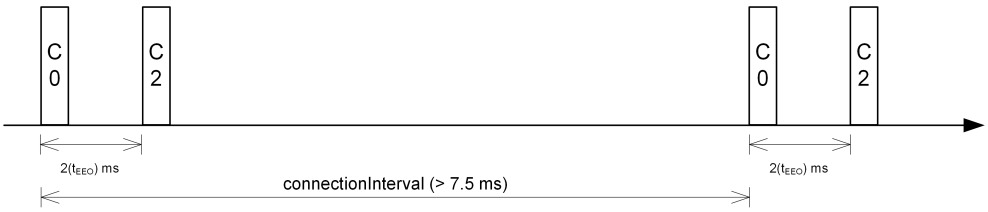


Figure 22: Multilink scheduling with maximum connections as a central and interval > min

14.2 Scanner timing

This section describes scanner timing with different connections.

Figure 23: Scanner timing - no active connections on page 48 shows that when scanning for advertisers with no active connections, the scan interval and window can be any value within the Bluetooth Core Specification.

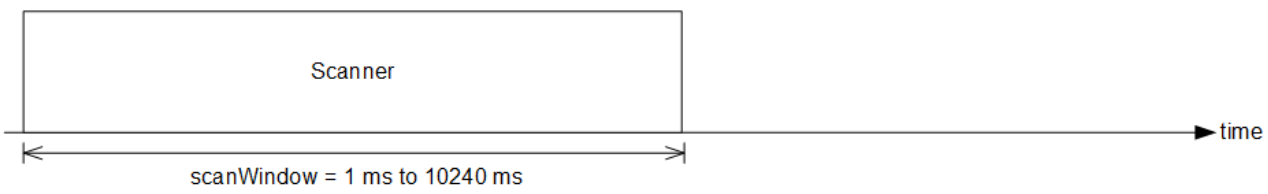


Figure 23: Scanner timing - no active connections

Figure 24: Scanner timing - one connection as a central on page 49 shows that when there is an active connection, the scanner or observer role will be started synchronously with the first connected link as a central at a distance of 3(t_{EEO})ms. With scanInterval equal to the connectionInterval and a scanWindow ≤ connectionInterval - (3*t_{EEO}+ tScanReserved)ms, scanning will proceed without packet loss.

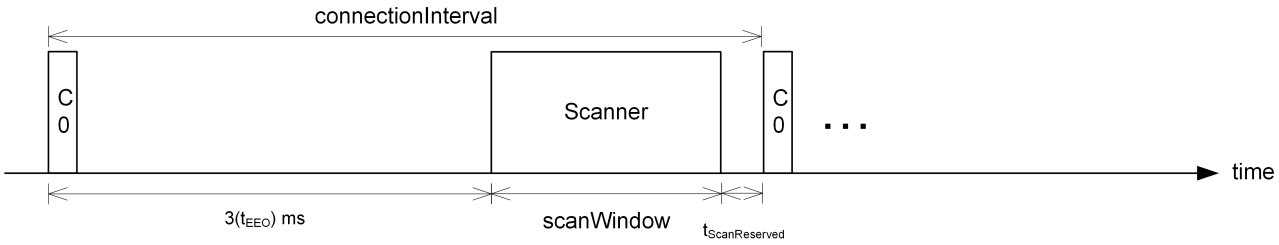


Figure 24: Scanner timing - one connection as a central

Figure 25: Scanner timing - one connection, long window on page 49 shows a scanner with a long scanWindow which will cause some connection events to be dropped.

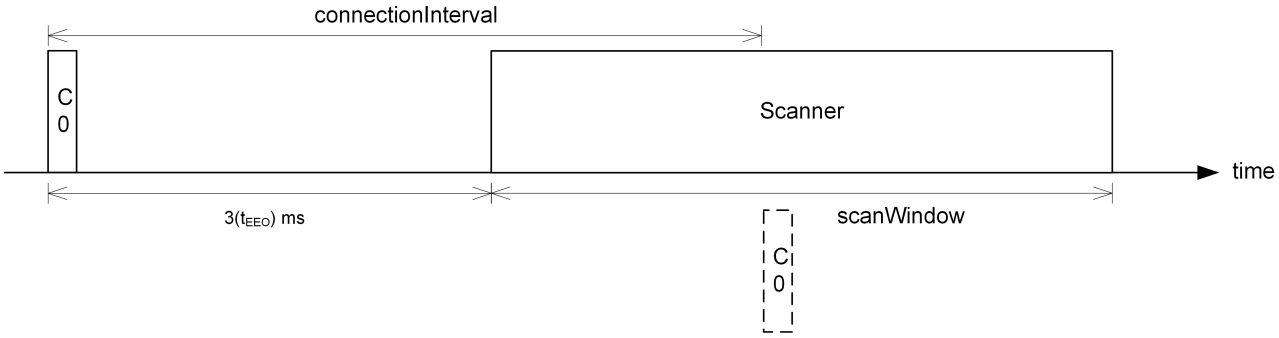


Figure 25: Scanner timing - one connection, long window

If all links as a central have a short connection interval (7.5 ms) and the scanner is started, the scanner events will collide with link events as a central causing packets on connections to be dropped as shown in Figure 26: Scanner timing - minimum connection interval on page 49.

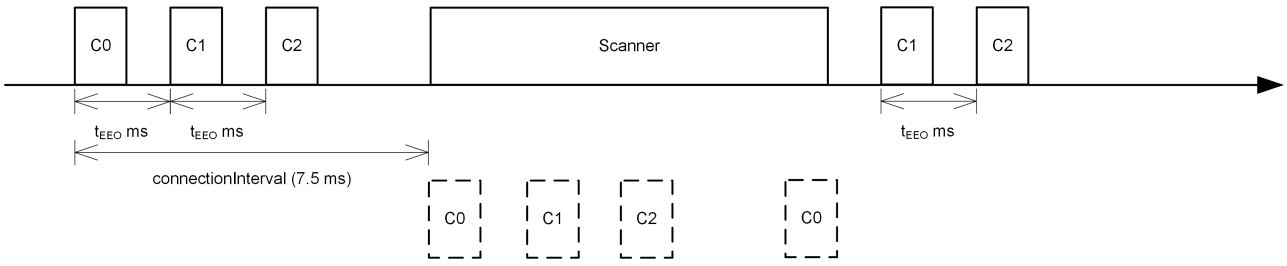


Figure 26: Scanner timing - minimum connection interval

14.3 Initiator timing

This section introduces the different situations what happens with the initiator when establishing a connection.

When establishing a connection with no other connections active, the initiator will establish the connection in the minimum time and allocate the first Central link connection event 1.25 ms after the connect request was sent, as shown in Figure 27: Initiator - first connection on page 50.

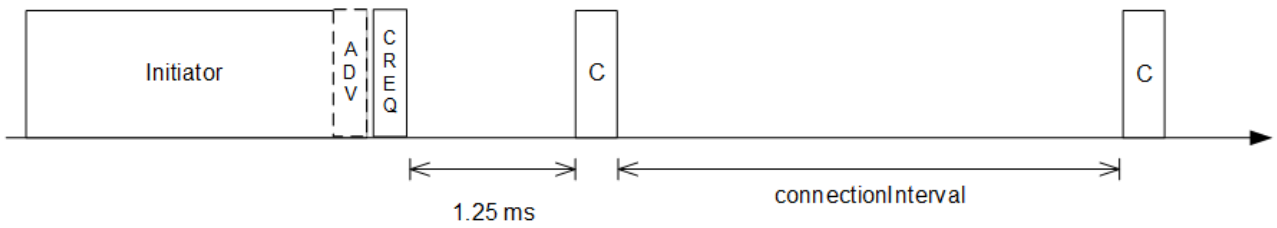


Figure 27: Initiator - first connection

When establishing a new connection with other connections already made as a central, the initiator will start asynchronously to the connected link events and position the new Central connection's first event in a free slot between existing central events. Figure 28: Initiator - one or more connections as a central on page 50 illustrates this when all existing central connections have the same connection interval and the initiator starts around the same time as the 1st Central connection (C0) event in the schedule. The new connection, C1, is positioned in the available slot between C0 and C2.

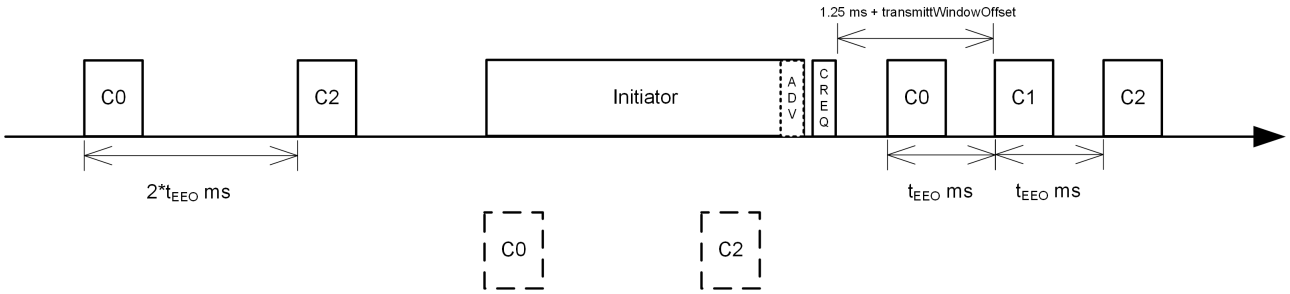


Figure 28: Initiator - one or more connections as a central

When establishing connections to newly discovered devices, the scanner may be used for discovery followed by the initiator. In Figure 29: Initiator - fast connection on page 50, the initiator is started directly after discovering a new device to connect as fast as possible to that device. The result is some connection events being dropped while the initiator runs. Events scheduled in the transmit window offset will not be dropped (C2). In this case the 2nd peer connection schedule is available (C1), and is allocated for the new connection.

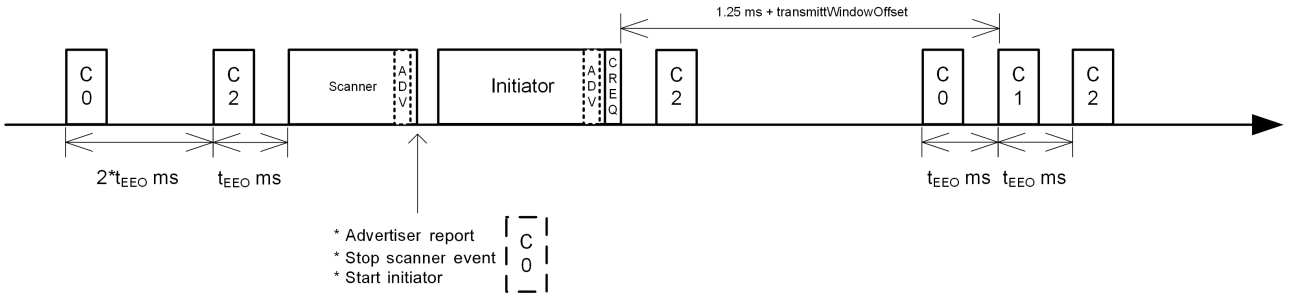


Figure 29: Initiator - fast connection

Important:

The inititator is scheduled asynchronously to any other role and assigned higher priority to ensure faster connection setup.

14.4 Advertiser (connectable and non-connectable) timing

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

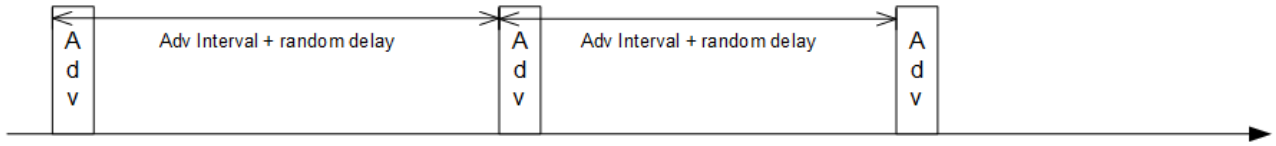


Figure 30: Advertiser

When other role events are running in addition, the advertiser role event may collide with those.

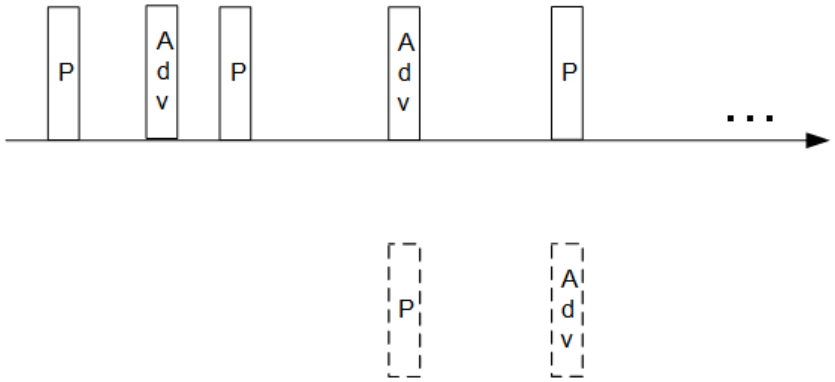


Figure 31: Advertiser collide

Directed advertiser is different compared to other advertiser types because it is not periodic. The scheduling of the single event required by directed advertiser is done in the same way as other advertiser type events. Directed advertiser is also started as early as possible, and its priority (refer to Table 32: Scheduling priorities on page 46) is raised if it is blocked by other roles multiple times.

14.5 Peripheral connection setup and connection timing

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

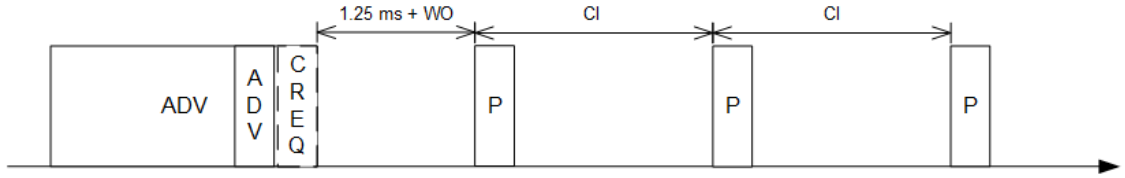


Figure 32: Peripheral connection setup and connection

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

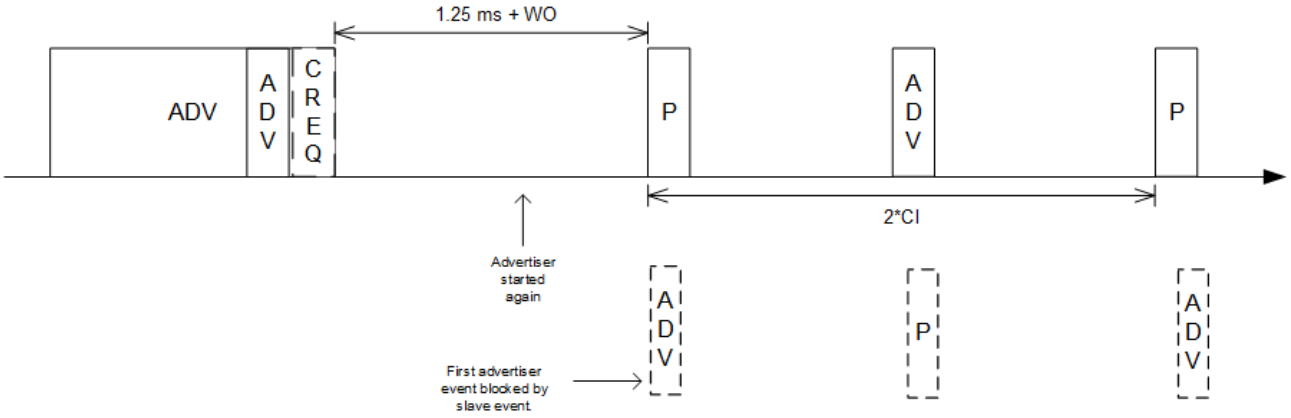


Figure 33: Peripheral connection setup and connection with collision

Table 34: Peripheral role timing ranges

Value	Description	Value (µs)
$t_{SlaveNominalWindow}$	Listening window on slave to receive first packet in a connection event.	1000 (assuming 250 ppm sleep clock accuracy on both slave and master with 1 second connection interval)
$t_{SlaveEventNominal}$	Nominal event length for slave link.	t_{prep}(max) + $t_{SlaveNominalWindow}$ + $t_{event\ (max\ for\ slave\ role)}$ Refer to Table 14: Radio Notification figure labels on page 23 and Table 15: BLE Radio Notification timing ranges on page 24.
$t_{SlaveEventMax}$	Maximum event length for slave link.	$t_{SlaveEventNominal}$ + 7 ms Where 7 ms is added for the maximum listening window for 500 ppm sleep clock accuracy and 4-second connection interval.
$t_{AdvEventMax}$	Maximum event length for advertiser (all types except directed advertiser) role.	t_{prep}(max) + $t_{event\ (max\ for\ adv\ role\ except\ directed\ adv)}$ Refer to Table 14: Radio Notification figure labels on page 23 and Table 15: BLE Radio Notification timing ranges on page 24.

14.6 Suggested intervals and windows

The distance between each connection as a central needed to send and receive one full length BLE packet before another event starts, is t_{EEO}.

Therefore three link events can complete in maximum 3 * t_{EEO},which is around 7.5 ms (see Table 33: Multilink central role timing ranges on page 48.

The minimum connection interval recommended for three connections is 7.5 ms. Note that this does not leave sufficient time in the schedule for scanning or initiating new connections (when the number of connections already established is less than three). Scanner, Observer, and Initiator events can therefore cause connection packets to be dropped as in Figure 26: Scanner timing - minimum connection interval on page 49.

It is recommended that all connections have intervals that have a common factor. This common factor should be 7.5 ms or more. In the case of using 7.5 ms as the common factor, all connections would have an interval of 7.5 ms or a multiple of 7.5 ms like 15 ms, 22.5 ms, 30 ms, etc.

If short connection intervals are not essential to the application and there is a need to have a scanner and/or initiator running at the same time as connections (an initiator will have to be started to establish new connections), then it is possible to avoid dropping packets on any connection connection as a central by having a connection interval larger than 7.5 ms plus the scanWindow plus $t_{ScanReserved}$. In this case, three connections and a scanner/initiator window can complete within each connection interval.

As an example, setting the connection interval to 40 ms will allow three connection events and a scanner/ initiator window of 31.0 ms, which is sufficient to ensure advertising packets from a 20 ms (nominal) advertiser hitting and being responded to within the window.

To summarize, a recommended configuration for operation without dropped packets for cases of only central roles running is:

1. The minimum Central connection intervals should be ≥ 3 * t_{EEO}+ scanWindow + $t_{ScanReserved}$.
2. All connections as a central should have connection intervals that have a common factor. This common factor should be 7.5 ms or more. For example [15 ms, 22.5 ms, 30 ms, 200 ms…] or [75 ms, 150 ms, 225 ms], etc.
3. Scanner, Observer, and Initiator roles should have intervals which can be factored by the smallest connection interval and the window should be ≤ connectionInterval – 3 * t_{EEO}– $t_{ScanReserved}$.

Peripheral roles use the same time space as all other roles (including any other peripheral and central roles), hence a collision free schedule cannot be guaranteed if a peripheral role is running along with any other role. The probability of collision can be reduced (though not eliminated) if the central role link parameters are set as suggested in this section, and the following rules are applied for all roles:

- The minimum interval of any roles is > = 3*t_{EEO} + ($t_{ScanReserved}$ + ScanWindow) + $t_{SlaveEventNominal}$ + $t_{AdvEventMax}$

  Important:

  $t_{SlaveEventNominal}$ can be used in above equation in most cases, but should be replaced by $t_{SlaveEventMax}$ for cases where links as a peripheral can have worst sleep clock accuracy and longer connection interval.
- Connections as a peripheral also follow the constraint of connection interval having common factor of 7.5 ms or more, similar to connections as a central.
- Broadcaster and Advertiser roles also follow the constraint of interval which can be factored by the smallest connection interval.

  Important:

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

If the above conditions are met, the worst case collision scenario will be broadcaster, connection as a peripheral, initiator and a connection as a central colliding in time. This will result in collision resolution via priority mechanism. The worst case collision will be reduced if any of the above roles are not running. For example, in the case of only connections as a central and slave connection is running, in the worst case each role will get a timeslot half the time because they both run with the same priority (Refer to Table 32: Scheduling priorities on page 46). Figure 34: Maximum links running as a central and peripheral on page 53 shows this case of collision.

Important:

These are worst case collision numbers, and an average case might be better.

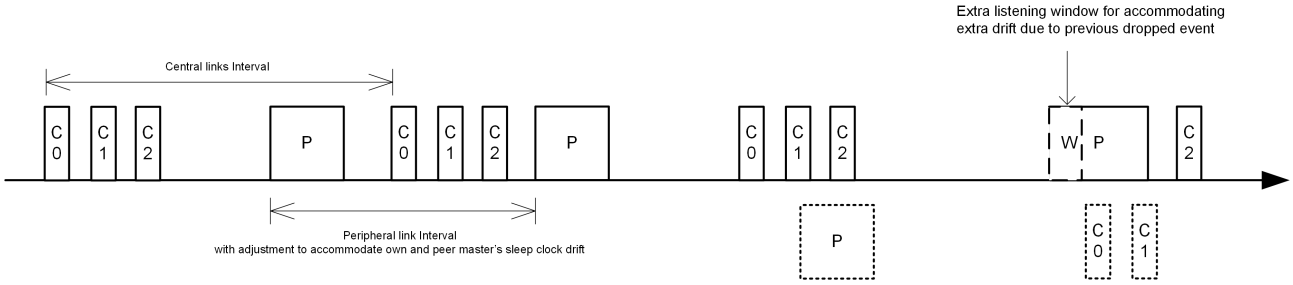


Figure 34: Maximum links running as a central and peripheral

Packet drops might happen due to collision between different roles, as is explained above. Application should tolerate dropped packets by setting the supervision time-out for connections long enough to avoid loss of

connection when packets are dropped. For example, in the case of only connections as a central and a slave connection is running, in the worst case each role will get a timeslot half the time. To accommodate this packet drop, the application should set the supervision time-out to twice the size it would have set if only either central or peripheral role was running.

Chapter 15
Processor availability and interrupt latency

This section documents key SoftDevice performance parameters for processor availability and interrupt latency.

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

This section describes latency introduced by the SoftDevice when managing interrupt events.

Latency, additional to ARM® Cortex™M0 hardware architecture latency, is introduced by SoftDevice logic to manage interrupt events. This latency occurs when an interrupt is forwarded to the application from the SoftDevice and is part of the minimum latency for each application interrupt. Additional latency is incurred due to interrupt processing and forwarding performed by the Master Boot Record (MBR). The maximum application interrupt latency is dependent on protocol stack activity as described in section Processor availability on page 55.

Table 35: Additional latency due to SoftDevice and MBR processing

Interrupt	CPU cycles	
	SoftDevice enabled	SoftDevice disabled
Open peripheral interrupt	54	30
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	N/A	37
Application SVC interrupt	59	59

15.2 Processor availability

This section gives an overview of interrupt levels and interrupt usage by the SoftDevice during protocol events.

Appendix A: SoftDevice architecture on page 73 describes interrupt management in SoftDevices and is required knowledge for understanding this section.

The SoftDevice protocol stack runs in the LowerStack and UpperStack interrupts. These protocol stack interrupts determine the processor availability and latencies for the interrupts/priorities available to the application - App(H), App(L), and main.

LowerStack processing will determine the processor availability and interrupt latency for App(H) (and all lower priorities), while LowerStack, App(H), and UpperStack processing together will determine the processor availability for App(L) and main context. The figure below illustrates UpperStack activity (API calls) and LowerStack activity (Protocol events) and the time reserved/not reserved for those interrupts.

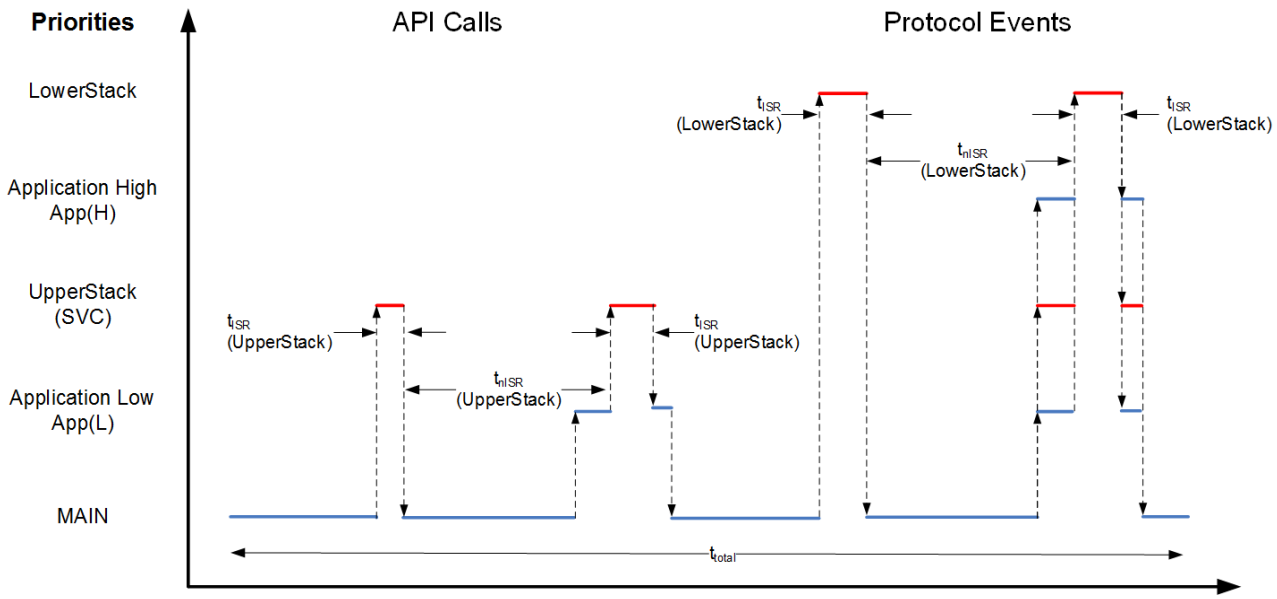


Figure 35: UpperStack and LowerStack activity

15.2.1 SoftDevice interrupt latency definitions

This section describes the parameters used for interrupt latency timings.

Table 36: SoftDevice interrupt latency definitions

Parameter	Description
t_{ISR} (LowerStack)	Interrupt processing time in LowerStack. This is the minimum interrupt latency for App(H) (and lower priorities).
t_{nISR} (LowerStack)	Time between LowerStack interrupts. This is the time available to run for App(H) (and lower priorities).
t_{ISR} (UpperStack)	Interrupt processing time in UpperStack. This is the minimum interrupt latency for App(L) and processing latency for main context.
t_{nISR} (UpperStack)	Time between UpperStack interrupts. This is the time available to run for App(L) and main context.

15.3 BLE peripheral performance

This section describes the processor availability and interrupt latency for the BLE peripheral stack.

The interrupt latency and processor availability interrupt latencies are dependent upon whether the SoftDevice uses CPU Suspend[12] during radio activity or not.

[12] CPU Suspend: During BLE protocol events, LowerStack interrupts are extended by a CPU Suspend state during radio activity to improve link integrity. This means that LowerStack interrupts will block application and UpperStack processing during a Radio Activity for a time proportional to the number of packets transferred during the Radio activity period.

For all S130 SoftDevice versions CPU Suspend is by default not enabled, but may optionally be enabled for compatibility with older versions of nRF51 ICs. This document describes interrupt latency and CPU availability when CPU Suspend is not used.

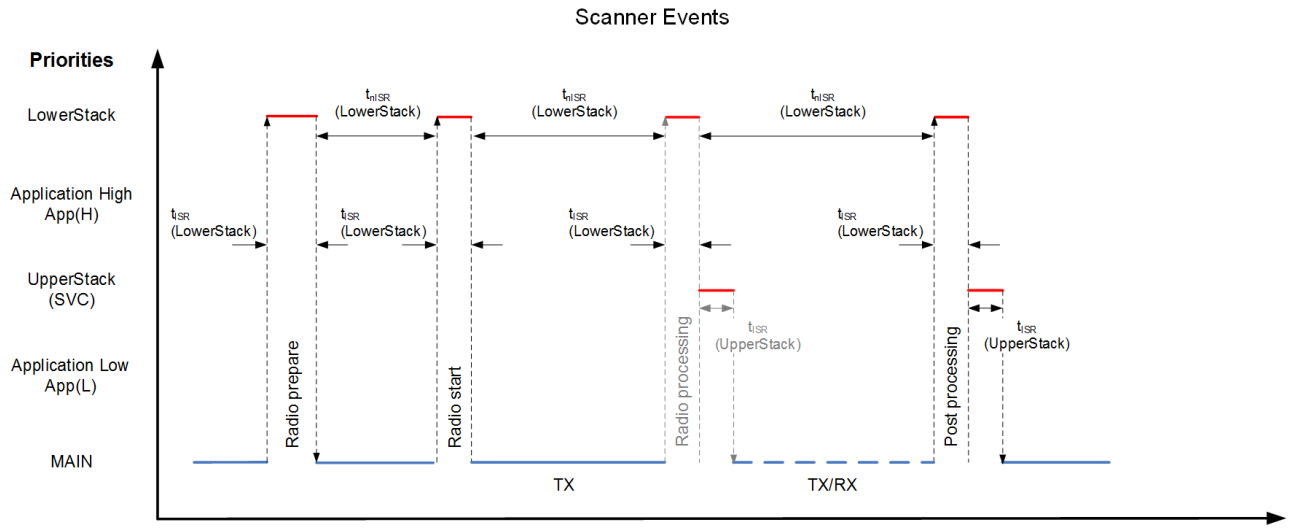


Figure 36: Advertising

For advertising, the pattern of LowerStack activity is as follows:

First, there is Radio prepare, which is followed by Radio start that starts the actual advertising. Depending upon the type of advertising, this may be followed by one or more instances of Radio processing (including UpperStack processing) and further reception/transmission. Finally, advertising ends with post processing and some UpperStack activity.

Table 37: Interrupt latency for advertising

Parameter	Description	Min	Typical	Max
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for advertising.			80 µs
$t_{ISR(LowerStack),RadioStart}$	Interrupt latency starting the advertising.			40 µs
$t_{ISR(LowerStack),RadioProcessing}$	Processing after sending/receiving a packet.			60 µs
$t_{ISR(LowerStack),PostProcessing}$	Interrupt latency at the end of an advertising event.		250 µs	640 µs
$t_{nISR(LowerStack)}$	Distance between interrupts during advertising.	40 µs	250 µs	
$t_{ISR(UpperStack)}$	UpperStack interrupt at the end of an advertising event.		300 µs	1.5 ms

15.3.1 BLE peripheral connection

In a connection event, the LowerStack activity is typically as follows: First there is Radio prepare and then Radio start, which starts the actual connection event (reception).

When the reception is finished, there is a Radio processing including a switch to transmission. When the transmission is finished, there is either a Radio processing including a switch back to reception and possibly a new transmission after that, or the event ends with Post processing.

After the LowerStack Post processing, the UpperStack processes any received packets with data, executes GATT, ATT or SMP operations and generates events to the application as required. The UpperStack interrupt is therefore highly variable based on the stack operations executed.

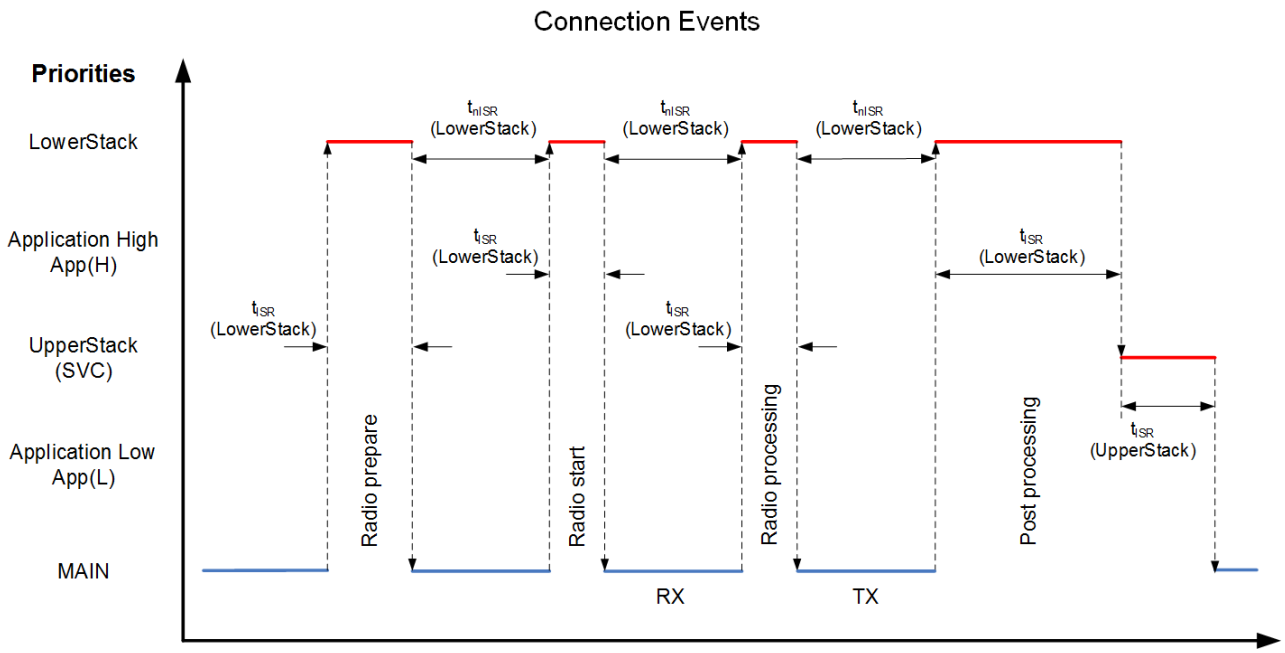


Figure 37: Peripheral connection events

The data in the table below is for a connection under good conditions. Continued packet loss, clock drift, and other effects may force longer Radio activity and longer LowerStack processing. This may affect the CPU availability and interrupt latency for lower priorities.

Table 38: Interrupt latency when connected

Parameter	Description	Min	Typical	Max
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for a connection event.			130 µs
$t_{ISR(LowerStack),RadioStart}$	Interrupt latency starting the connection event.			40 µs
$t_{ISR(LowerStack),RadioProcessing}$	Interrupt latency after sending or receiving a packet.			100 µs
$t_{ISR(LowerStack),PostProcessing}$	Interrupt latency at the end of a connection event.		300 µs	720 µs
$t_{nISR(LowerStack)}$	Distance between interrupts during a connection event.	30 µs	150 µs	
$t_{ISR(UpperStack)}$	UpperStack interrupt processing.		500 µs	1.5 ms

15.4 BLE central performance

This section describes the processor availability and interrupt latency for the BLE central stack.

The interrupt latency and processor availability interrupt latencies are dependent upon whether the SoftDevice uses CPU Suspend[13] during radio activity or not. See section BLE peripheral performance on page 56 for more information on when CPU Suspend is enabled.

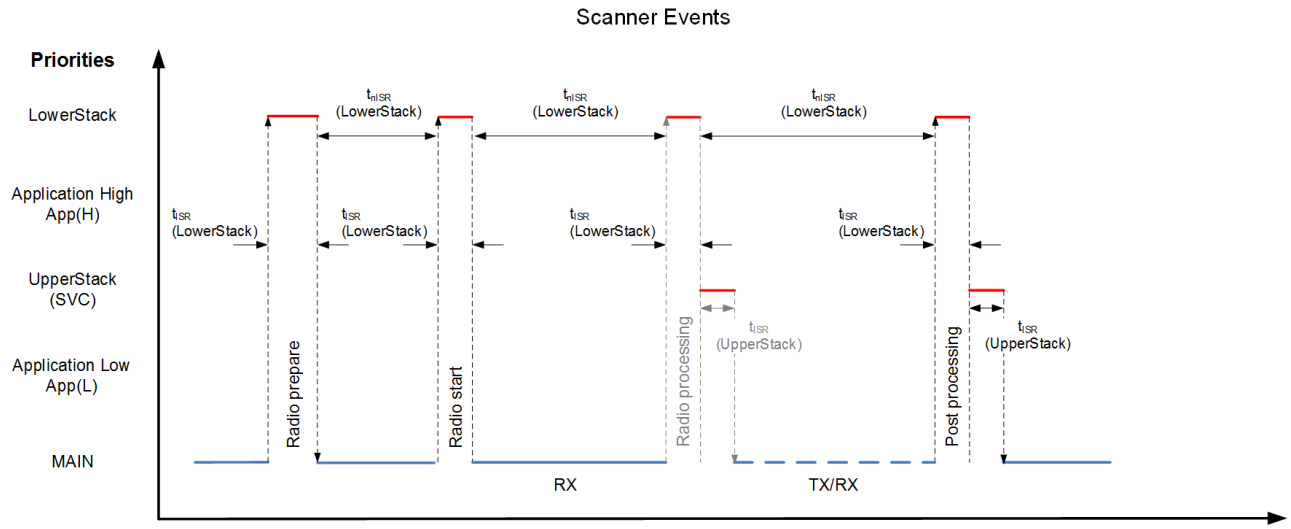


Figure 38: Scanning or initiating

For scanning and initiating, the pattern of LowerStack activity is as follows: First, there is Radio prepare, which is followed by Radio start that starts the actual scanning or initiating.

During scanning, there will be zero or more instances of Radio processing, depending upon whether the scanning is passive or active, whether advertising packets are received or not and upon the type of the received advertising packets. Such Radio processing may be followed by UpperStack processing.

During initiating a connectable advertising packet may be received. Packet reception will cause radio processing, which may result in sending a connect request before ending initiating.

Scanning and initiating ends with Post processing and some UpperStack activity.

Table 39: Interrupt latency for passive scanning or initiating

Parameter	Description	Min	Typical	Max
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for scanning or initiating.			140 µs
$t_{ISR(LowerStack),RadioStart}$	Interrupt latency starting the scan or initiation.			100 µs
$t_{ISR(LowerStack),RadioProcessing}$	Processing after sending/receiving packet.			130 µs
$t_{ISR(LowerStack),PostProcessing}$	Interrupt latency at the end of a scanner or initiator event.		250 µs	650 µs
$t_{nISR(LowerStack)}$	Distance between interrupts during scanning.	30 µs		2 ms
$t_{ISR(UpperStack)}$	UpperStack interrupt at the end of a scanner or initiator event.		300 µs	1.5 ms

[13] CPU Suspend: During BLE protocol events, LowerStack interrupts are extended by a CPU Suspend state during radio activity to improve link integrity. This means that LowerStack interrupts will block application and UpperStack processing during a Radio Activity for a time proportional to the number of packets transferred during the Radio activity period.

15.4.1 Central connection event interrupt latency

In a connection event, the LowerStack activity is as follows: First there is Radio prepare and then Radio start, which starts the actual connection event (transmission).

When the transmission is finished, there is Radio processing including a switch to reception. When the reception is finished, the event ends with Post processing.

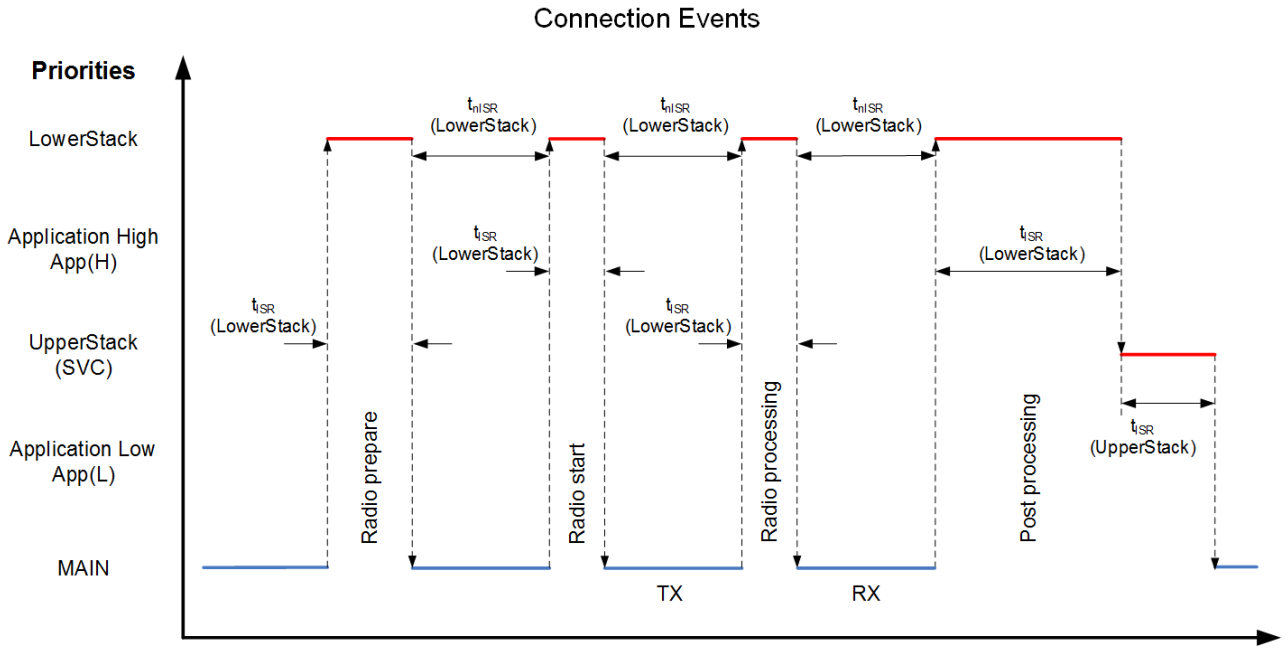


Figure 39: Central connection events

After the LowerStack Post processing, the UpperStack processes any received packets with data, executes GATT, ATT or SMP operations and generates events to the application as required. The UpperStack interrupt is therefore highly variable based on the stack operations executed. Interrupt latency during connections is outlined in the table below.

Table 40: Interrupt latency when connected

Parameter	Description	Min	Typical	Max
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for a connection event.			130 µs
$t_{ISR(LowerStack),RadioStart}$	Interrupt latency starting the connection event.			100 µs
$t_{ISR(LowerStack),RadioProcessing}$	Interrupt latency after sending a packet.			100 µs
$t_{ISR(LowerStack),PostProcessing}$	Interrupt latency at the end of a connection event.		350 µs	780 µs
$t_{nISR(LowerStack)}$	Distance between connection event interrupts.	30 µs	150 µs	

Parameter	Description	Min	Typical	Max
$t_{ISR(UpperStack)}$	UpperStack interrupt processing.		500 µs	1.5 ms

15.5 BLE CPU utilization

This section shows the CPU capacity available for an application with different configurations of peers connected.

The available CPU capacity depends on the number of peers connected, connection interval, and data throughput.

For the data collected in Table 41: CPU capacity available for the application, with the SoftDevice managing connections and receiving data from peers simultaneously on page 61, each peer is sending 1 packet per connection interval with 5 bytes of data. For the idle test the links are being maintained but no data sent.

The last row in the table shows the SoftDevice is receiving write requests from 3 peripherals and 1 central. It processes those requests and sends responses to all 4 peers on a connection interval of 150 ms. The SoftDevice uses 4% of the processor time, and leaves 96% for the customer's application.

When receiving notifications from 4 peers (without responses), the CPU is free 97% of the time.

Table 41: CPU capacity available for the application, with the SoftDevice managing connections and receiving data from peers simultaneously

Peers connected	Connection interval	RX write requests with response CPU free	RX notifications CPU free	Idle CPU free
1 peripheral	7.5 ms	88%	86%	91%
1 peripheral	20.0 ms	95%	94%	96%
1 peripheral	100.0 ms	98%	98%	98%
1 peripheral	150.0 ms	98%	98%	98%
1 central	7.5 ms	88%	85%	91%
1 central	20.0 ms	95%	94%	96%
1 central	100.0 ms	98%	98%	98%
1 central	150.0 ms	98%	98%	98%
3 peripherals	7.5 ms	62%	55%	73%
3 peripherals	20.0 ms	86%	83%	89%
3 peripherals	100.0 ms	96%	96%	97%
3 peripherals	150.0 ms	97%	97%	98%
3 peripherals	7.5 ms	69%	63%	77%
3 peripherals	20.0 ms	79%	86%	85%
3 peripherals	100.0 ms	95%	94%	96%
3 peripherals	150.0 ms	96%	96%	97%

15.6 Performance with Flash memory API, Concurrent Multiprotocol Timeslot API and multiple roles

Use of the Flash memory and the Concurrent Multiprotocol Timeslot API may affect CPU availability.

The LowerStack interrupt is also used by the Flash memory API processing and by the Concurrent Multiprotocol Timeslot API processing. Therefore the use of these APIs may affect CPU availability and interrupt latencies for all lower priorities. The effects of this are dependent upon the application and the use case.

Chapter 16
BLE data throughput

To achieve maximum data throughput, the application must exchange data at a rate that matches on-air packet transmissions and use the maximum data payload per packet.

The maximum data throughput limits in Table 42: GATT maximum data throughput as a peripheral with a connection interval of 7.5 ms on page 63 and Table 43: GATT maximum data throughput as a central for each connection on page 63 apply to packet transfers under the following common conditions:

- Encrypted link
- CPU Suspend (Radio CPU mutex) not enabled

Table 42: GATT maximum data throughput as a *peripheral* with a connection interval of 7.5 ms

Protocol	Role	Method	Maximum data throughput
GATT	Client	Receive Notification Send Write command Send Write request Simultaneous receive Notification and send Write command	63.4 kbps 42.6 kbps 10.6 kbps 50.8 kbps (each direction)
GATT	Server	Send Notification Receive Write command Receive Write request Simultaneous send Notification and receive Write command	42.6 kbps 63.4 kbps 10.6 kbps 50.8 kbps (each direction)

Table 43: GATT maximum data throughput as a *central* for each connection

Protocol	Role	Method	Number of connected peripherals	Connection interval (ms)	Maximum data throughput
GATT	Client	Receive Notification	1 - 3 1 - 3	20 50	8 kbps 3.2 kbps
		Send Write command	1 - 3 1 - 3	20 50	8 kbps 3.2 kbps
		Send Write request	1 - 3 1 - 3	20 50	4 kbps 1.6 kbps

Protocol	Role	Method	Number of connected peripherals	Connection interval (ms)	Maximum data throughput
		Simultaneous receive Notification and send Write command	1 1 - 3 1 - 3	7.5 20 50	21.3 kbps (each direction) 8 kbps (each direction) 3.2 kbps (each direction)
GATT	Server	Send Notification	1 - 3 1 - 3	20 50	8 kbps 3.2 kbps
		Receive Write command	1 - 3 1 - 3	20 50	8 kbps 3.2 kbps
		Receive Write request	1 - 3 1 - 3	20 50	4 kbps 1.6 kbps
		Simultaneous send Notification and receive Write command	1 1 - 3 1 - 3	7.5 20 50	21.3 kbps (each direction) 8 kbps (each direction) 3.2 kbps (each direction)

Important:

1. 1 kbps = 1000 bits per second
2. Values are rounded to the closest 0.1 kbps

Chapter 17
BLE power profiles

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

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

The LowerStack CPU profile during the event is shown separately. These profiles are based on typical events with empty packets.

17.1 Advertising event

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

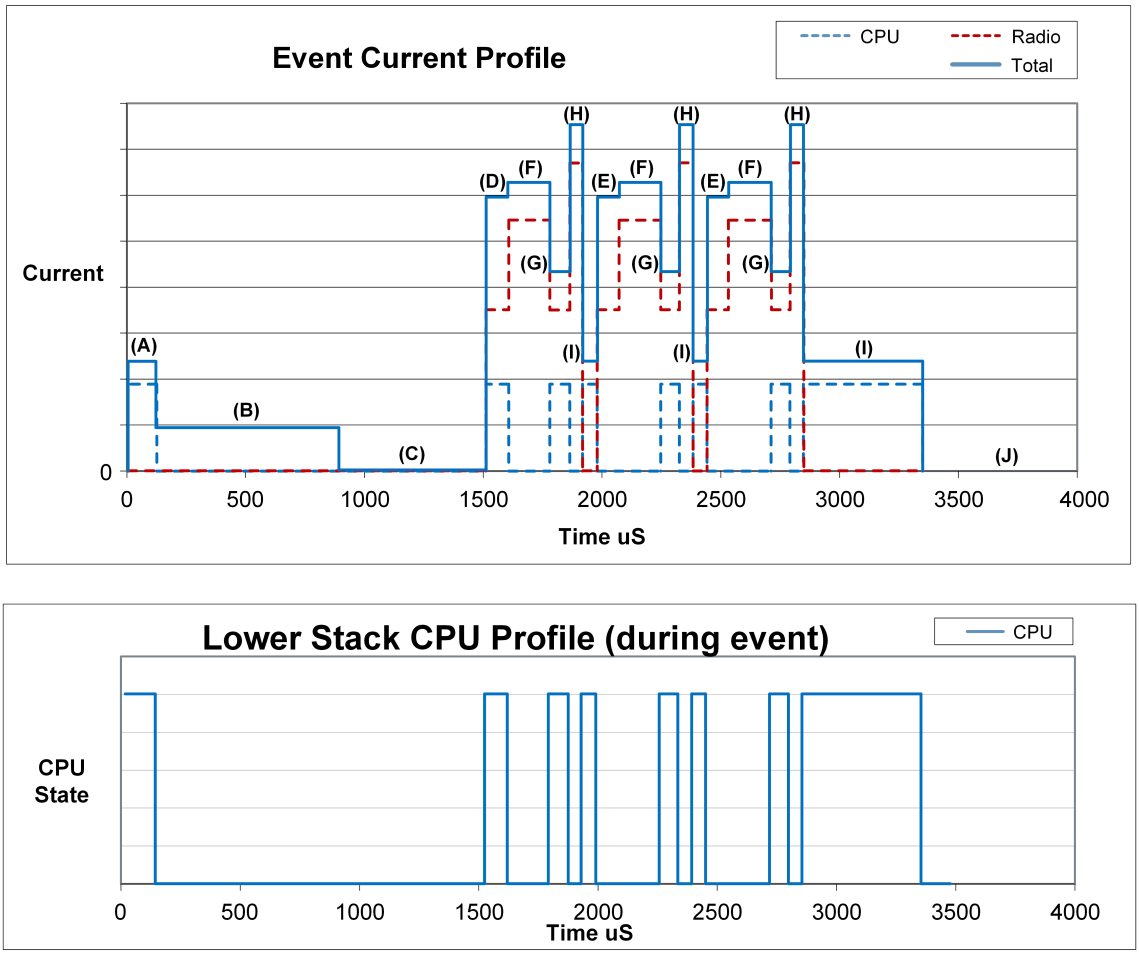


Figure 40: Advertising event

Table 44: Advertising event

Stage	Description	Current calculation[14]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash} + I_{START,X16M}$

Stage	Description	Current calculation[14]
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start/switch	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#I(_{START,TX}) + I_{CPU,Flash}$
(E)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX})$
(F)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM}$
(G)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(H)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(I)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(J)	Idle	$I_{ON} + I_{RTC} + I_{X32k}$

Important:

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

17.2 Peripheral connection event

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

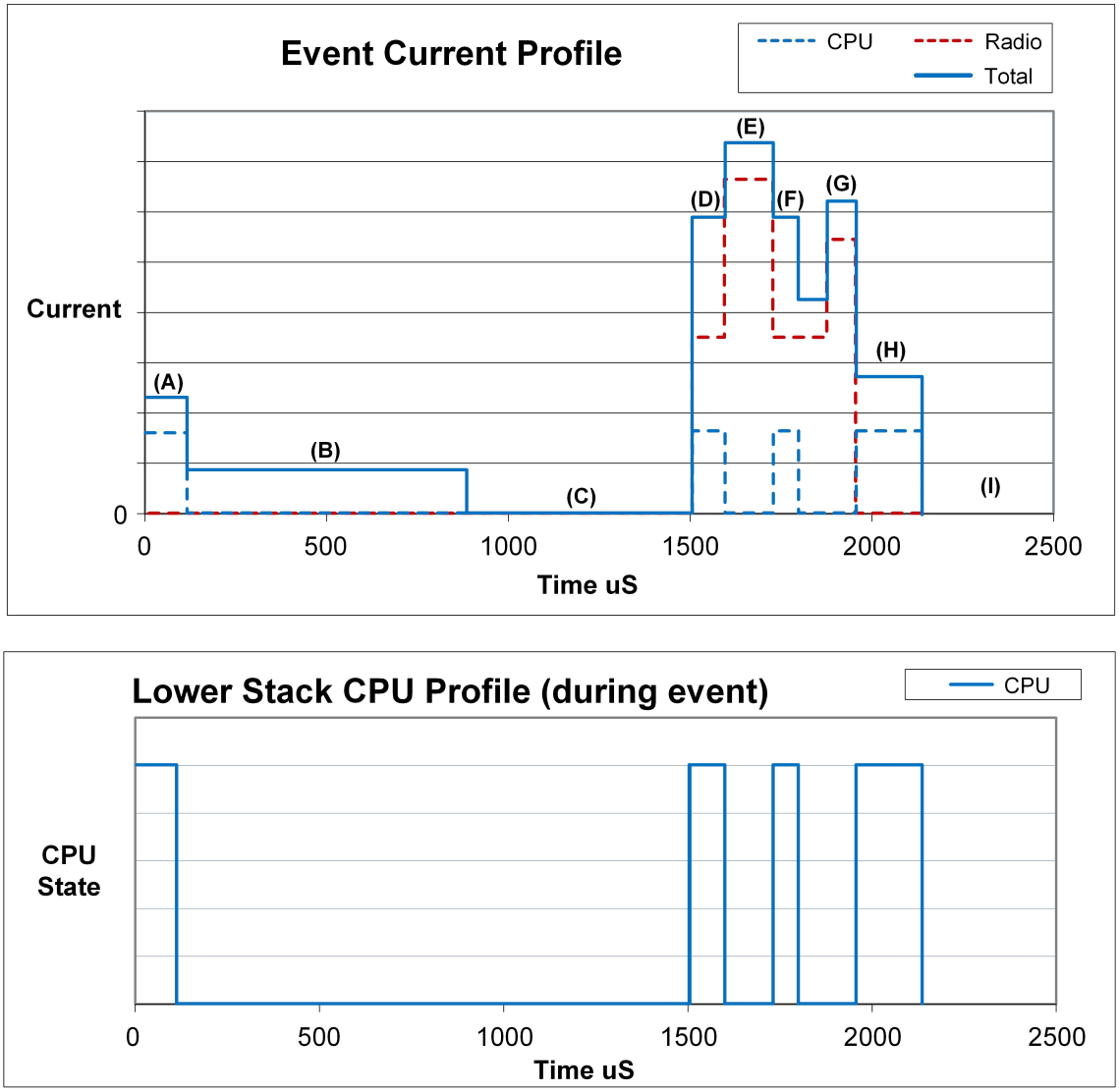


Figure 41: Peripheral connection event

Table 45: Peripheral connection event

Stage	Description	Current Calculation[15]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash} + I_{START,X16M}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(E)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX} + I_{CRYPTO}$
(F)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX}) + I_{CPU,Flash}$
(G)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM} + I_{CRYPTO}$
(H)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

Important:

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

17.3 Scanning event

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

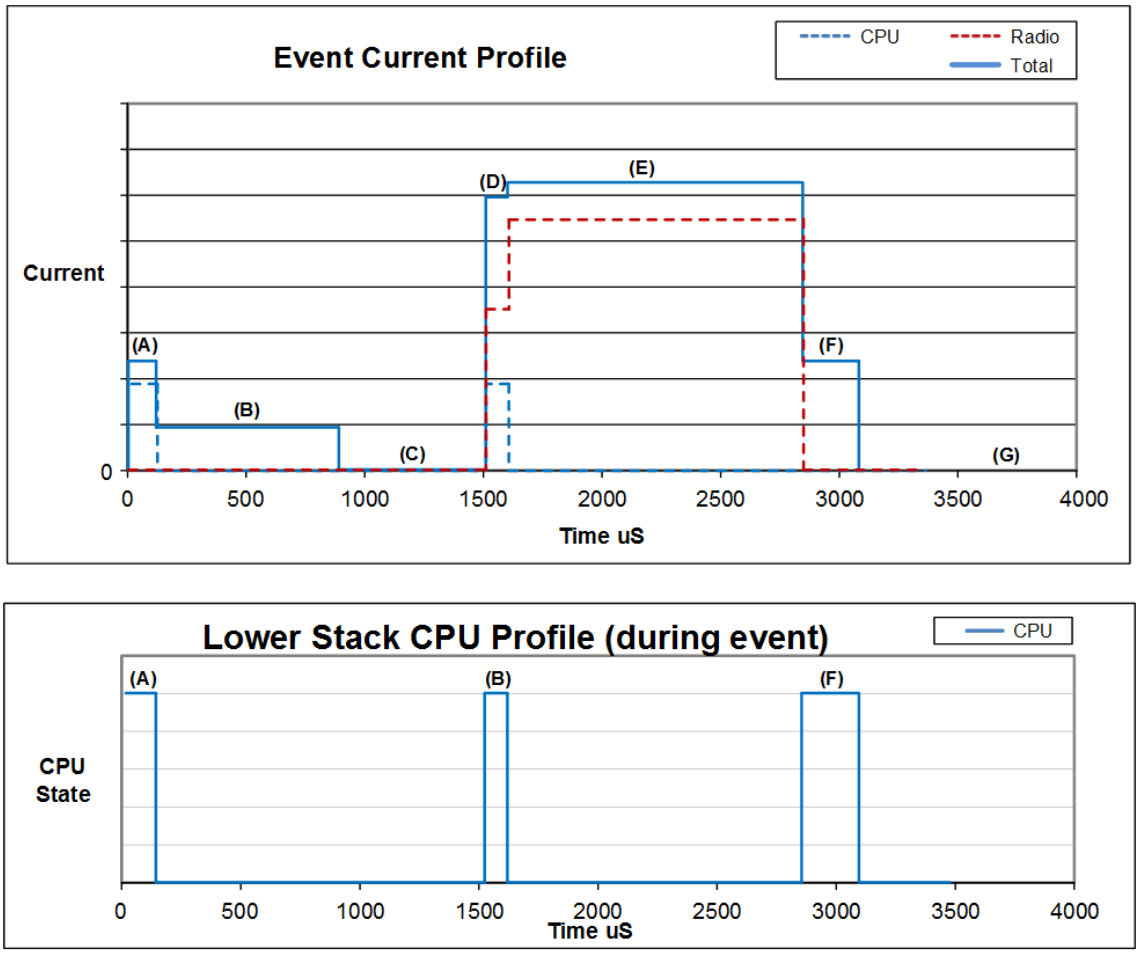


Figure 42: Scanning event

Table 46: Scanning event

Stage	Description	Current Calculation[16]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$
(E)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(F)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(G)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

Important:

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

17.4 Central connection event

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

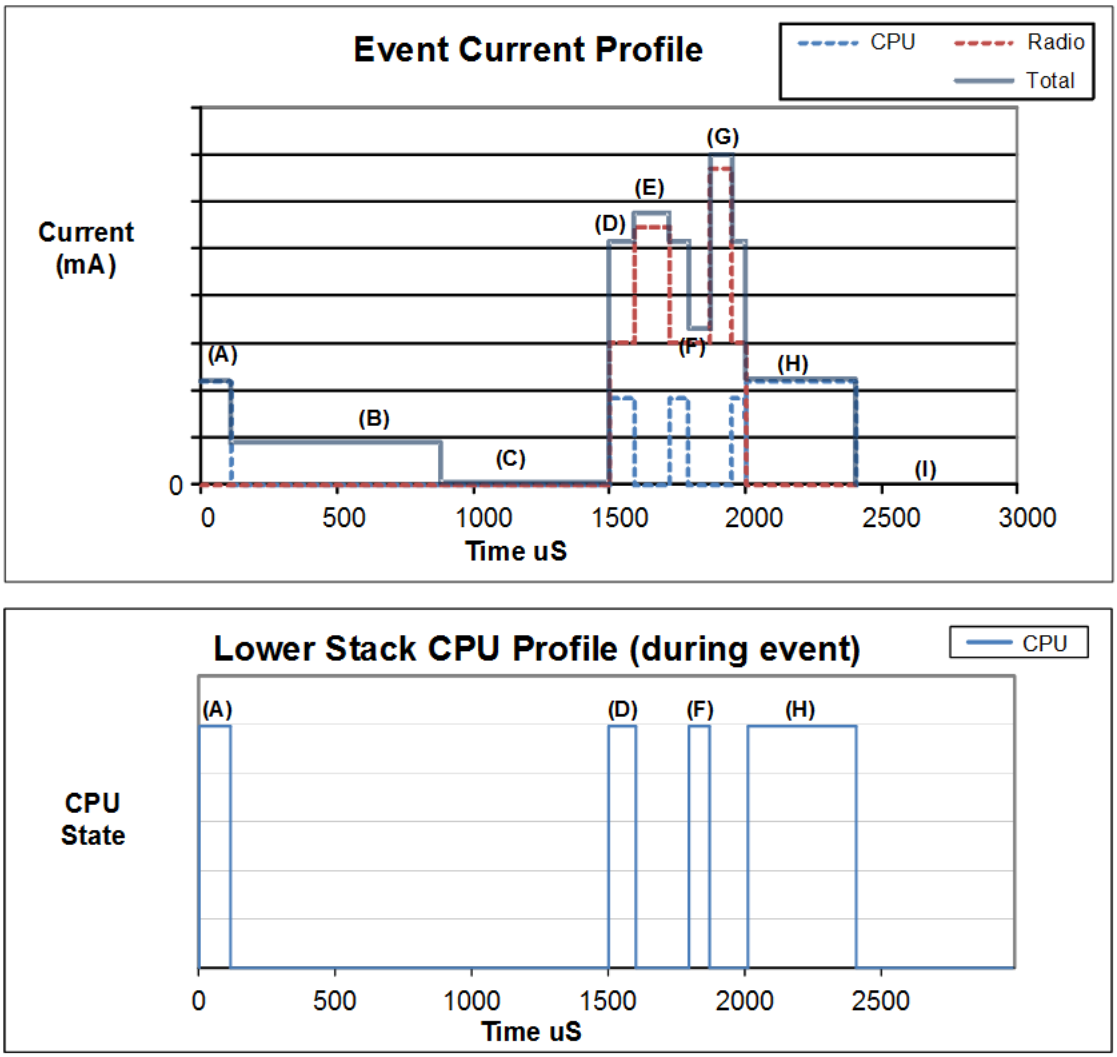


Figure 43: Central connection event

Table 47: Central connection event

Stage	Description	Current Calculation[17]
(A)	Pre-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$I_{ON} + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio start	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,TX}) + I_{CPU,Flash}$
(E)	Radio TX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM}$
(F)	Radio turn-around	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + \#(I_{START,RX}) + I_{CPU,Flash}$

Stage	Description	Current Calculation[17]
(G)	Radio RX	$I_{ON} + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(H)	Post-processing	$I_{ON} + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$I_{ON} + I_{RTC} + I_{X32k}$

Important:

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

Chapter 18
SoftDevice identification and revision scheme

The SoftDevices are identified by the SoftDevice part code, a qualified IC partcode (for example, nRF51822), and a version string.

The identification scheme for SoftDevices consists of the following items:

- For revisions of the SoftDevice which are production qualified, the version string consists of major, minor, and revision numbers only, as described in the table below.
- or revisions of the SoftDevice which are not production qualified, a build number and a test qualification level (alpha/beta) are appended to the version string.
- For example: s110_nrf51_1.2.3-4.alpha, where major = 1, minor = 2, revision = 3, build number = 4 and test qualification level is alpha. Additional examples are given in table Table 49: SoftDevice revision examples on page 71.

Table 48: Revision scheme

Revision	Description
Major increments	Modifications to the API or the function or behavior of the implementation or part of it have changed. Changes as per minor increment may have been made. Application code will not be compatible without some modification.
Minor increments	Additional features and/or API calls are available. Changes as per minor increment may have been made. Application code may have to be modified to take advantage of new features.
Revision increments	Issues have been resolved or improvements to performance implemented. Existing application code will not require any modification.
Build number increment (if present)	New build of non-production versions.

Table 49: SoftDevice revision examples

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

Sequence number	Description
s110_nrf51_1.2.3-5.beta	Revision 1.2.3, fifth build, qualified at beta level
s110_nrf51_1.2.3	Revision 1.2.3, qualified at production level

Table 50: 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.

Chapter 19
Appendix A: SoftDevice architecture

A SoftDevice is precompiled and linked binary software implementing a wireless protocol.

Figure 44: Software architecture block diagram on page 73 is a block diagram of the nRF51 series software architecture including the standard ARM® CMSIS interface for nRF51 hardware, profile and application code, application specific peripheral drivers, and a firmware module identified as a SoftDevice.

While the SoftDevice is software, application developers have minimal compile-time dependence on its features. The unique hardware and software supported framework in which it executes, provides run time isolation and determinism in its behavior. These characteristics make the interface comparable to a hardware peripheral abstraction with a functional, programmatic interface.

The SoftDevice Application Program Interface (API) is available to applications as a high-level programming language interface.

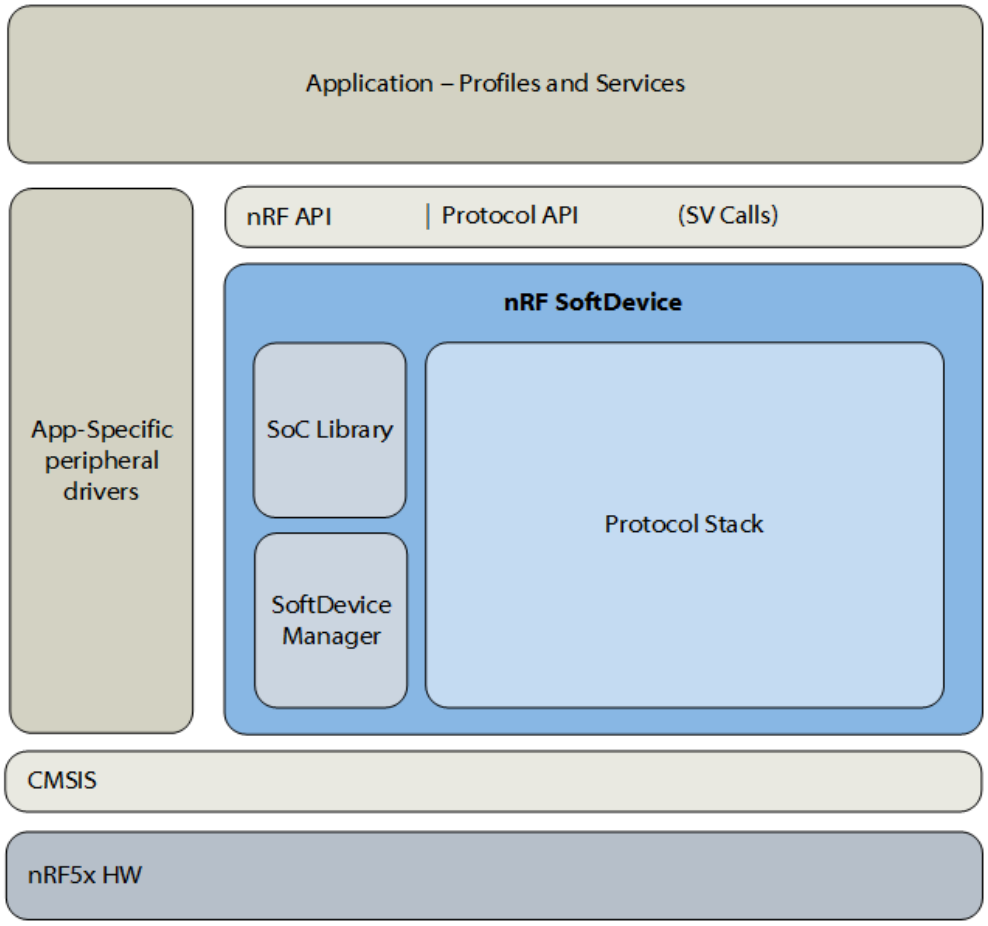


Figure 44: Software architecture block diagram

A SoftDevice consists of three main components:

- SoC Library - Implementation and nRF API for shared hardware resource management (application coexistence).

- SoftDevice Manager - Implementation and nRF API for SoftDevice management (enabling/disabling the SoftDevice, etc.).
- Protocol stack - Implementation of protocol stack and API.

When the SoftDevice is disabled, only the SoftDevice Manager API is available for the application. For more information about enabling/disabling the SoftDevice, see SoftDevice enable and disable on page 81.

19.1 System on Chip (SoC) library

The System on Chip (SoC) library provides functions for accessing shared hardware resources.

The features of this library will vary between implementations of SoftDevices so detailed descriptions of the SoC library API are made available with the Software Development Kits (SDK) specific to each SoftDevice.

Table 51: Common components in the SoC library

Component	Description
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.
MUTEX	Disabling interrupts shall not be done while the SoftDevice is enabled. Mutex functions have been implemented to provide safe regions.
RAND	Random number generator - hardware sharing between SoftDevice and application.
POWER	Power management - Functions for power management.
CLOCK	Clock management – Functions for managing clock sources.
PPI	Safe PPI access to dedicated Application PPI channels.
PWR_MNG	Power management support (not a full implementation) for the application.

19.2 SoftDevice Manager

The SoftDevice Manager (SDM) API implements functions for controlling the state of the SoftDevice enabled/disabled.

When enabled, the SDM configures low frequency clock (LFCLK) source, interrupt management and the embedded protocol stack.

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

19.3 Protocol stack

The major component in each SoftDevice is a wireless protocol stack providing abstract control of the RF transceiver features for wireless applications.

For example, fully qualified Bluetooth low energy and ANT™ protocols layers may be implemented in a SoftDevice to provide application developers with an out-of-the-box solution for applications using standard 2.4 GHz protocols.

19.4 Application Program Interface (API)

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

The nRF API is consistent across SoftDevices in the nRF51 range of ANT™ and Bluetooth products for code compatibility.

All SoftDevice APIs are implemented using thread-safe Supervisor Calls (SVC). All application interaction with the stack and libraries is asynchronous and event driven. From the application's point of view these look like regular functions, but no linking against a library is required. All SVC interface functions are provided through header files for the SDM, SoC Library, and protocol(s).

SV calls are conceptually software triggered interrupts with a procedure call standard for parameter passing and return values. Each API call generates an interrupt allowing single-thread API context and SoftDevice function locations to be independent from the application perspective at compile-time. SoftDevice API functions can only be called from a lower interrupt priority than the SVC priority. For more information, see Exception (interrupt) management with a SoftDevice on page 78.

19.5 Memory isolation and run time protection

SoftDevice program memory, data memory and peripherals can be sandboxed to prevent SoftDevice program corruption by the application ensuring robust and predictable performance.

Sandboxing[18] is enabled by writing the start address of the application program memory to UICR.CLENR0.

Program memory and RAM are divided into two regions using registers. Region 0 is occupied by the SoftDevice while Region 1 is available to the application.

Code regions are defined when programming a SoftDevice by setting a register defining program code length. RAM regions are defined at run-time when the SoftDevice is enabled. The figure below presents an overview of the regions.

[18] A sandbox is a set of access rules for memory imposed on the user.

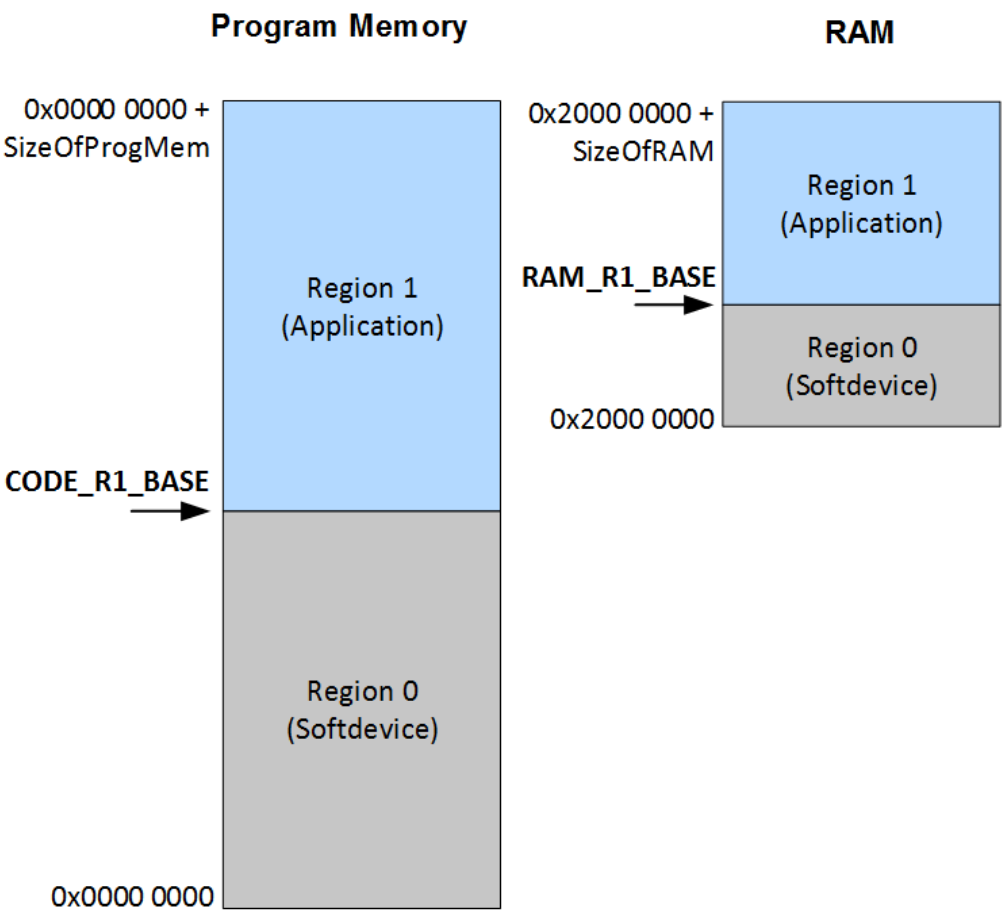


Figure 45: Memory region designation

The SoftDevice uses a fixed amount of flash (program) memory. The amount of RAM used is dependent upon whether the SoftDevice is enabled or not. The flash and RAM usage is specified by size (kilobytes or bytes) and the CODE_R1_BASE and RAM_R1_BASE addresses which are the usable base addresses of Application code and RAM respectively. Application code must be located between CODE_R1_BASE and SizeOfProgMem while the Application RAM must be allocated between RAM_R1_BASE and the top of RAM, excluding the allocation for the call stack and heap.

Example Application program code address range:

```
CODE_R1_BASE ≤ Program ≤ SizeOfProgMem
```

Example Application RAM address range assuming call stack and heap location as shown in:

```
RAM_R1_BASE ≤ RAM ≤ (0x2000 0000 + SizeOfRAM) - (Call Stack + Heap)
```

Sandboxing protects region 0 memory. Region 0 program memory cannot be written or erased at run time[19]. Region 0 RAM cannot be written to by an application at run time. Violation of these rules, for example an attempt to write to the protected Region 0 memory, will result in a system Hard Fault as defined by the ARM® architecture. There are debugging restrictions applied to these regions which are outlined in the “Memory Protection Unit (MPU)” chapter in the nRF51 Reference Manual that do not affect execution.

When the SoftDevice is disabled, the whole of the RAM, with the exception of a few bytes, is available to the application. In the context of an enabled SoftDevice however, lower address space of RAM will be "consumed" by the SoftDevice and be marked as write protected.

It is important to note that when the SoftDevice is disabled, the RAM previously used by the application will not be restored. In practice, the application will in many cases want to specify its RAM region from the

[19] An exception is replacing the SoftDevice using MBR API functions.

protected memory length until the end of RAM. This is to make application development easier without having to think about what data to place where.

Notice:

- The call stack is conventionally located by the initial value of Main Stack Pointer (MSP) at the top address of RAM.
- By default RAM1 block is OFF in System ON-mode. If the MSP initial value defined in the application vector table is in the RAM1 block, the RAM block will be enabled before the application reset vector is executed.
- Do not change the value of MSP dynamically (i.e. never set the MSP register directly).
- RAM located in the SoftDevice's region will be modified once the SoftDevice is enabled.
- The RAM modified by the SoftDevice will not be restored on SoftDevice disable.

19.6 Call stack

The call stack is defined by the application.

The main stack pointer (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 used 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.

In configurations without an OS, the main stack grows down and is shared with the nRF51 SoftDevice. The Cortex™M0 has no hardware for detecting stack overflow, and the application is responsible for leaving enough space both for the application itself and the nRF51 SoftDevice stack requirements.

It is customary, but not required, to let the stack run downwards from the upper limit of RAM Region 1.

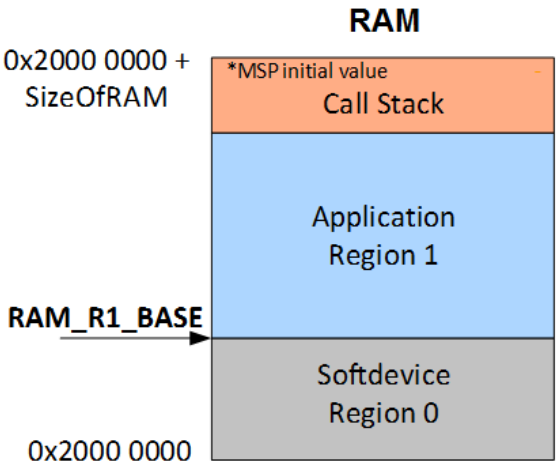


Figure 46: Call stack location example

With each release of a nRF51 SoftDevice its maximum (worst case) call stack requirement is specified, see the SoftDevice specification for more information. The SoftDevice uses the call stack when LowerStack or UpperStack events occur. These events are asynchronous to the application so the application programmer must reserve call stack for the application in addition to the call stack requirement for the SoftDevice.

19.7 Heap

At this time there is no heap required by nRF51 SoftDevices.

The application is free to allocate and use a heap without disrupting the function of a SoftDevice.

19.8 Peripheral run time protection

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

Protected peripheral registers are readable by the application. As with program and data memory protection, an attempt to perform a write to a protected peripheral will result in a Hard Fault. Note that peripherals are only protected while the SoftDevice is enabled, otherwise they are available to the application. See Table 26: Peripheral protection and usage by SoftDevice on page 42 for an overview of the peripherals that are restricted by the SoftDevice.

19.9 Exception (interrupt) management with a SoftDevice

To implement Service Call (SVC) APIs and ensure that embedded protocol real-time requirements are met independent of application processing, the SoftDevice implements an exception model for execution.

Care must be taken when selecting the correct interrupt priority for application events according to the guidelines that follow. The NVIC API to the SoC Library supports safe configuration of interrupt priority from the application.

The Cortex™M0 processor has four configurable interrupt priorities ranging from 0 to 3 (with 0 being highest priority). On reset, all interrupts are configured with the highest priority (0).

The highest priority (LowerStack) is reserved by the SoftDevice to service real-time protocol timing requirements and thus must remain unused by the application programmer. The SoftDevice also reserves priority 2 (UpperStack (SVC) priority). This priority is used by higher level, deferrable, SoftDevice tasks and the API functions executed as SVC interrupts (see Application Program Interface (API) on page 75).

The application provides two configurable priorities, App(H) and App(L), in addition to the background level - main or thread context.

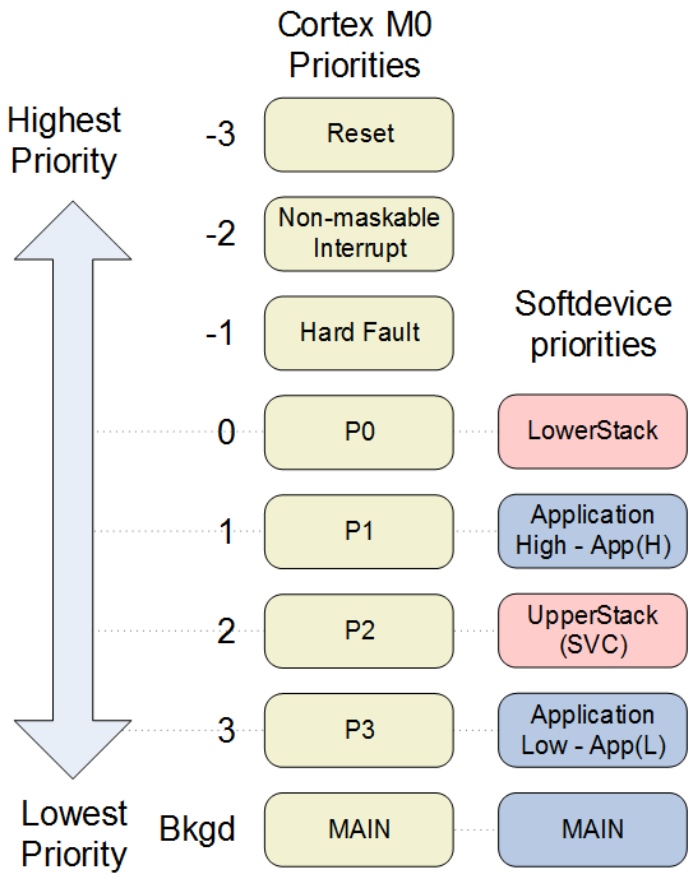


Figure 47: Exception model

As seen from the figure, App(H) is located between the two priorities reserved by the SoftDevice. This enables a low-latency application interrupt to support fast sensor interfaces. The App(H) priority will only experience latency from interrupts in the LowerStack priority, while App(L) can experience latency from LowerStack, App(H) and UpperStack context interrupts.

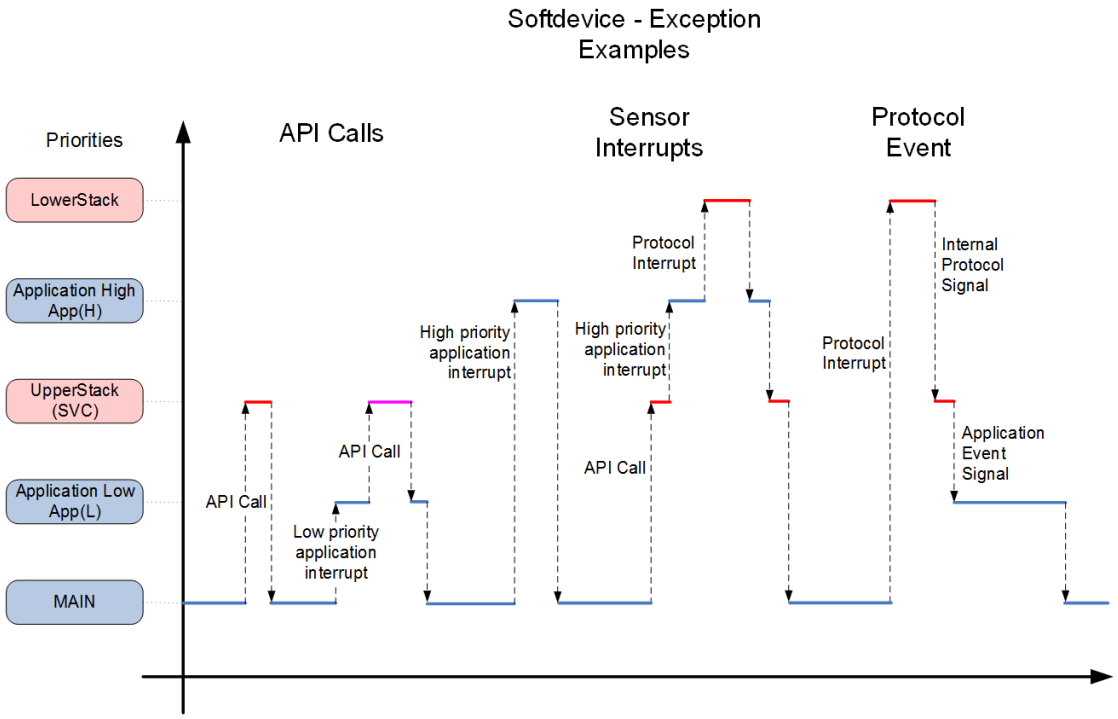


Figure 48: SoftDevice exception examples

19.10 Interrupt forwarding to the application

The forwarding of interrupts to the application is dependent upon SoftDevice state.

At the lowest level, the SoftDevice Manager receives all interrupts regardless of whether the SoftDevice is enabled or not. When the SoftDevice is enabled, some interrupt numbers are reserved for use by the protocol stack implemented in the SoftDevice and any handler defined by the application will not receive these interrupts. The reserved interrupts directly correspond to the hardware resource usage of the SoftDevice which can be found in the corresponding SoftDevice Specification. For example, if a SoftDevice (or embedded protocol stack) requires the exclusive use of a peripheral "TIMER0", that peripheral's interrupt handler can be implemented in the application, but will not be executed while the SoftDevice is enabled.

All interrupts corresponding to hardware peripherals not used by the SoftDevice are forwarded directly to the application defined interrupt handler. For the SoftDevice Manager to locate the application interrupt vectors, the application must define its interrupt vector table at the bottom of code Region 1 illustrated in the figure below. The use of a bootloader introduces some exceptions to this, see Master Boot Record and bootloader on page 37. In a majority of toolchains, the base address of the application code is positioned after the top address of the SoftDevice. Then, the code can be developed as a standard ARM® Cortex™M0 application project with the compiler creating the interrupt vector table as normal.

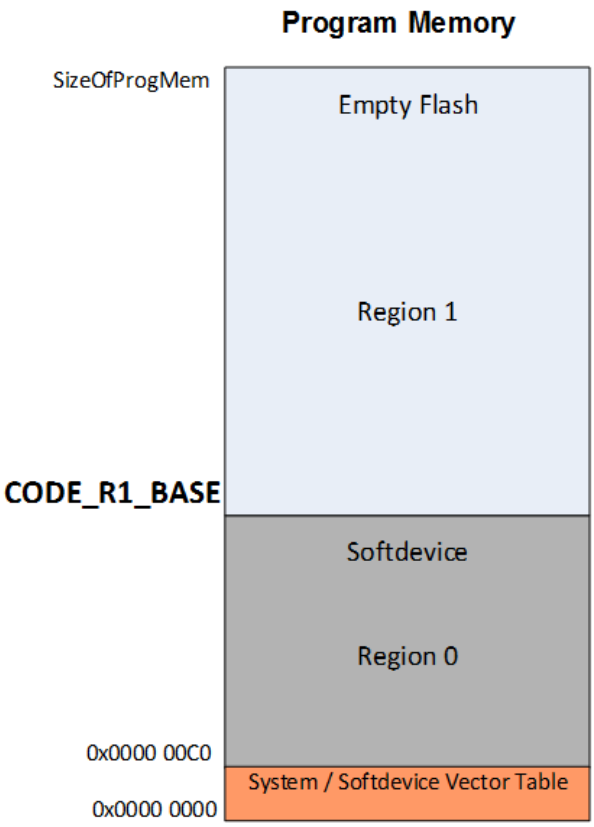


Figure 49: Call stack location example

The SVC interrupt is handled by SoftDevice Manager and the SVC number is inspected. If equal or greater than 0x10, the interrupt is processed by the SoftDevice. Values below 0x10 cause the SVC to be forwarded to the application. This allows the application to make use of a range of SVC numbers for its own purpose, for example, for an RTOS.

> **Important:** While the Cortex™M0 allows each interrupt to be assigned to an IRQ level 0 to 3, the priorities of the interrupts reserved by the SoftDevice cannot be changed. This includes the SVC interrupt. Handlers running at Application High level have neither access to SoftDevice functions nor to application specific SVCs or RTOS functions running at Application Low level.

If the SoftDevice is not enabled, all interrupts are immediately forwarded to the application specified handler. The exception to this is that SVC interrupts with an SVC number above or equal to 0x10 are not forwarded.

19.11 Events - SoftDevice to application

Software triggered interrupts in reserved IRQ slots are used to signal events from SoftDevice to application.

For details on this technique and how to implement handling of these events, refer to the Software Development Kit (SDK) for your device.

19.12 SoftDevice enable and disable

Before enabling the SoftDevice, you cannot use any capabilities of the SoftDevice. This extends to the use of the SoC library and protocol stack functions.

All of the IC's resources are freely available to the application, with some exceptions:

- SVC numbers 0x10 to 0xFF are reserved.
- SoftDevice program (flash) memory is reserved.

- A few bytes of RAM are reserved.

Once the SoftDevice has been enabled, more restrictions apply:

- Some RAM will be reserved.
- Some peripherals will be reserved.
- Some of the peripherals that are reserved will have a SoC library interface.
- Interrupts will not arrive in the application for reserved peripherals.
- 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.
- Maximum interrupt latency will be determined by the SoftDevice.

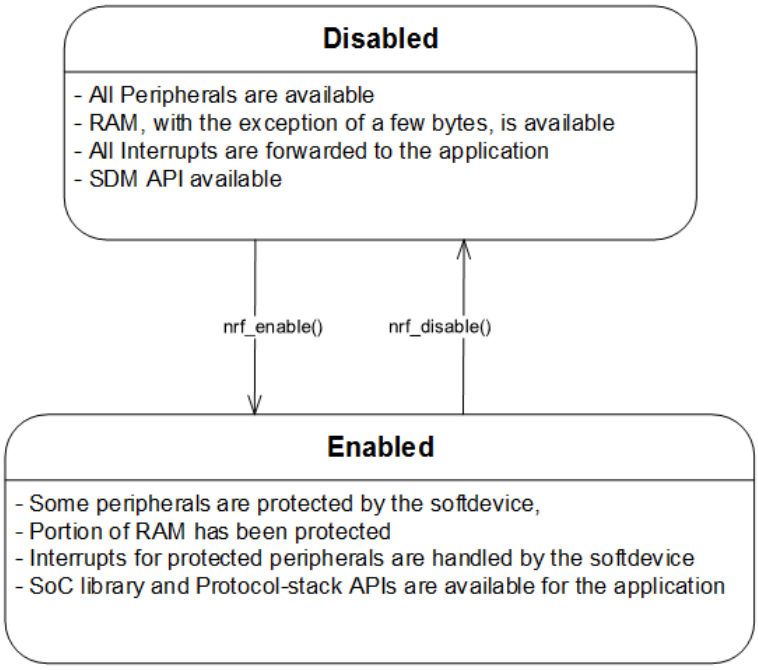


Figure 50: Call stack location example

19.13 Power management

While the SoftDevice is disabled, the application must implement low-level power management by itself by accessing the corresponding hardware register directly.

After a SoftDevice is enabled, the POWER peripheral will be protected. This means that all interactions with the POWER peripheral must happen through the SoC Library Power API.

This API provides an interface for turning on/off peripherals and checking the power status of peripherals that are not protected by the SoftDevice. The application will also have the ability to set the other registers in the peripheral and put the iC in System OFF.

19.14 Error handling

All SoftDevice API functions return an error code on success and failure.

Hard Faults are triggered if an application attempts to access memory contrary to the sandbox rules or peripheral configurations at run time.

An assertion mechanism through a registered callback can indicate fatal failures in the SoftDevice to the application.