S120 nRF51822

Bluetooth® low energy Central SoftDevice

SoftDevice Specification v1.0

Key Features

- *Bluetooth*® 4.1 compliant low energy single-mode protocol stack
    - Link layer
    - L2CAP, ATT, and SM protocols
    - GATT and GAP APIs
    - Central and Observer roles - up to 8 simultaneous connections
    - GATT Client and Server
    - Security Manager including MITM and OOB pairing
- Complementary nRF51 SDK including *Bluetooth* profiles and example applications
- 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 non-concurrent multiprotocol operation
    - Alternate protocol stack running in application space

Applications

- A4WP wireless charging
- Sports & Fitness devices
    - Sports watches
    - Bike computers

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Life support applications

Nordic Semiconductor's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest distributor, please visit http://www.nordicsemi.com.
Information regarding product updates, downloads, and technical support can be accessed through your My Page account on our homepage.

Main office:
Otto Nielsens veg 12
7052 Trondheim
Norway
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89

Mailing address:
Nordic Semiconductor
P.O. Box 2336
7004 Trondheim
Norway

Document Status

Status	Description
v0.5	This specification contains target specifications for product development.
v0.7	This specification contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
v1.0	This specification contains final product specifications. Nordic Semiconductor ASA reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

Revision History

Date	Version	Description
May 2014	1.0	First production release. **Added content:** • ***Section 3.1 "Profile and service support"*** on page 7 • ***Chapter 6 "Flash memory API"*** on page 14 • ***Chapter 7 "Radio Notification"*** on page 18 • ***Chapter 8 "Bootloader"*** on page 19 • ***Chapter 13 "BLE power profiles"*** on page 38 **Updated content:** • Front page • ***Section 1.1 "Documentation"*** on page 4 • ***Section 3.1 "Profile and service support"*** on page 7 • ***Section 3.2 "Bluetooth low energy features"*** on page 8 • ***Section 3.3 "Limitations on procedure concurrency"*** on page 11 • ***Chapter 4 "SoC library"*** on page 12 • ***Section 9.2 "Hardware blocks and interrupt vectors"*** on page 21 • ***Chapter 9.4 "Programmable Peripheral Interconnect (PPI)"*** on page 23 • ***Chapter 10 "Multi-link Central role scheduling"*** on page 25 • ***Section 11.2 "Processor availability"*** on page 32 • ***Section 11.4 "BLE Central Connection Event"*** on page 35 • ***Chapter 12 "BLE data throughput"*** on page 37
November 2013	0.5	Preliminary release.

1 Introduction

The S120 SoftDevice is a *Bluetooth*® low energy (BLE) Central protocol stack solution supporting up to eight simultaneous Central role connections. It integrates a low energy controller and host, and provides a full and flexible application programming interface (API) for building *Bluetooth* low energy System on Chip (SoC) solutions.

This document contains information about the SoftDevice features and performance.

Note: The SoftDevice features and performance are subject to change between revisions of this document. See ***Section 14.1 "Notification of SoftDevice revision updates"*** on page 40 for more information. To find information on any limitations or omissions, please refer to the SoftDevice release notes, which contain a detailed summary of the release status.

1.1 Documentation

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

Document	Description
nRF51 Series Reference Manual	"Appendix A: SoftDevice architecture" in the *nRF51 Series Reference Manual* is essential reading for understanding the resource usage and performance related chapters of this document.
nRF51 822 Product Specification (PS)	Contains a description of the hardware, modules, and electrical specifications specific to the nRF51*822* chip.
nRF51822 Product Anomaly Notification (PAN)	Contains information on anomalies related to the nRF51*822* chip.
Bluetooth Core Specification	The *Bluetooth Core Specification* version 4.1, Volumes 1, 3, 4, and 6 describes *Bluetooth* terminology which is used throughout the SoftDevice Specification.

1.2 Writing conventions

This SoftDevice Specification follows a set of typographic rules to ensure that the document is consistent and easy to read. The following writing conventions are used:

- Command, event names, and bit state conditions are written in `Lucida Console`.
- Pin names and pin signal conditions are written in **`Consolas.`**
- File names and User Interface components are written in **bold**.
- Internal cross references are italicized and written in ***semi-bold.***
- Placeholders for parameters are written in *italic regular font*. For example, a syntax of the function initialize will be written as: *initialize(parameter1, parameter2)*.
- Fixed parameters are written in regular text font. For example, a syntax description of SetChannelPeriod will be written as: SetChannelPeriod(0, Period).

2 Product overview

This section provides an overview of the SoftDevice.

2.1 SoftDevice

The SoftDevice is a precompiled and linked binary software that integrates a *Bluetooth* 4.1 low energy (BLE) protocol stack on the nRF51x22 chip.

The Application Programming Interface (API) is a standard C language set of 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 needing to integrate with proprietary chip-vendor software frameworks. This means that any ARM® Cortex™-M0 compatible toolchain can be used to develop *Bluetooth* low energy applications with the SoftDevice.

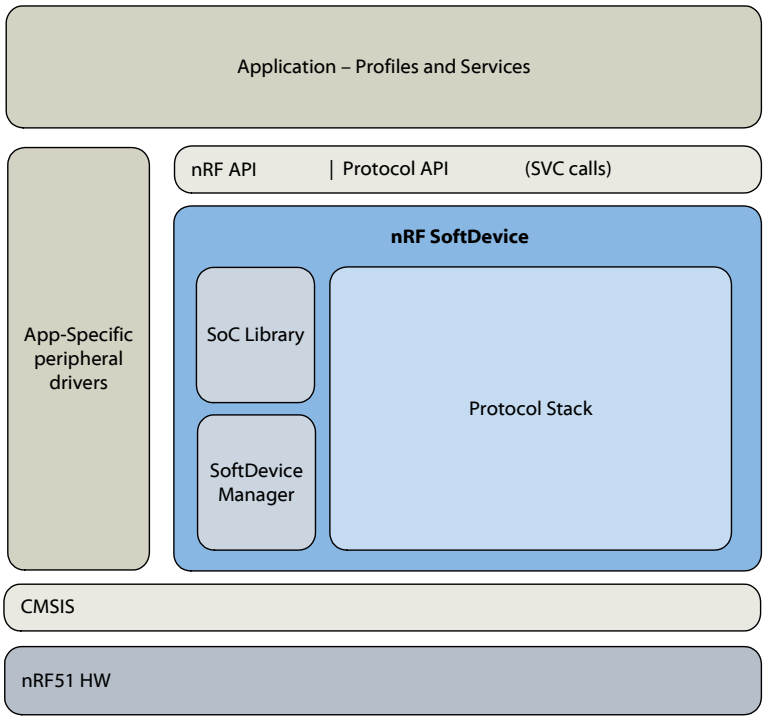

Figure 1 *System on Chip application with the SoftDevice*

The SoftDevice can be programmed onto compatible nRF51 Series chips during both development and production. This specification outlines the supported features of a production level SoftDevice. Alpha and beta versions may not support all features.

2.2 Multiprotocol support

The SoftDevice supports non-concurrent multiprotocol implementations. This means 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.

3 *Bluetooth* low energy protocol stack

The *Bluetooth* 4.1 compliant low energy Host and Controller embedded in the SoftDevice are fully qualified with multi-role support (Central and Observer). The API is defined above the Generic Attribute Protocol (GATT), Generic Access Profile (GAP), and Logical Link Control and Adaptation Protocol (L2CAP). The SoftDevice allows applications to implement standard *Bluetooth* low energy profiles as well as proprietary use case implementations.

The nRF51 Software Development Kit (SDK) complement the BLE protocol stack with Service and Profile implementations. Single-mode System on Chip (SoC) applications are enabled by the full BLE protocol stack and nRF51xxx integrated circuit (IC).

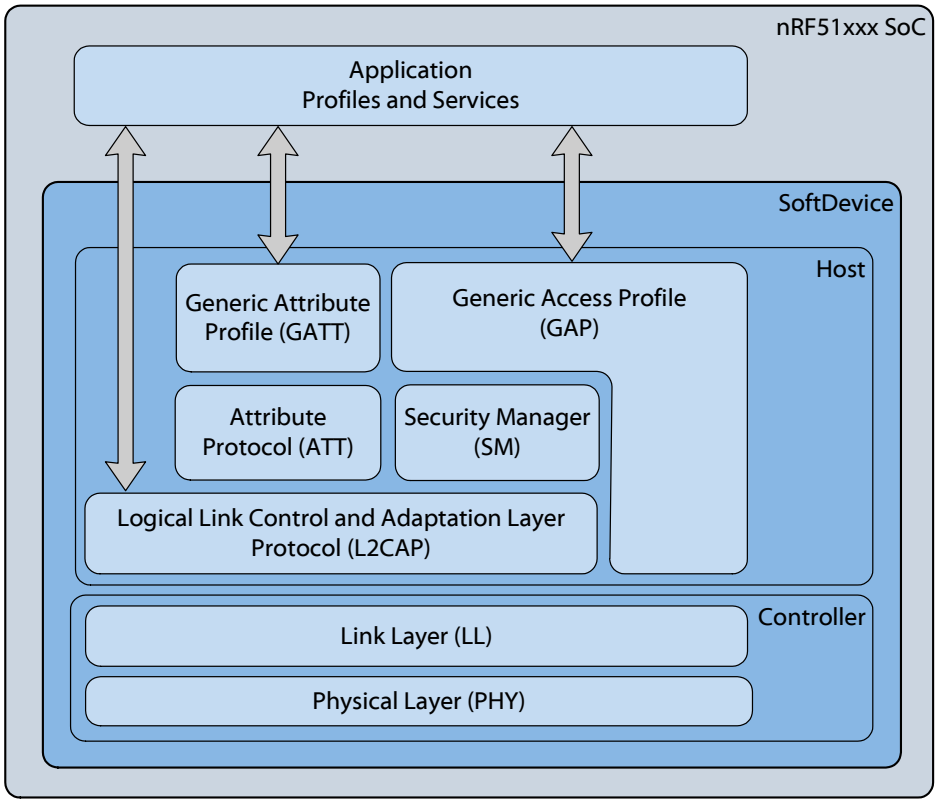

Figure 2 *SoftDevice stack architecture*

3.1 Profile and service support

The Profiles and corresponding Services supported by the SoftDevice are shown in ***Table 1***.

Adopted Profile	Adopted Services	Supported
HID over GATT	HID Battery Device Information	YES
Heart Rate	Heart Rate Device Information	YES
Blood Pressure	Blood pressure	YES
Health Thermometer	Health Thermometer	YES
Glucose	Glucose	YES
Phone Alert Status	Phone Alert Status	YES
Alert Notification	Alert Notification	YES
Time	Current Time Next DST Change Reference Time Update	YES
Find Me	Immediate Alert	YES
Cycling speed and cadence	Cycling speed and cadence Device information	YES
Running speed and cadence	Running speed and cadence Device information	YES
Location and Navigation	Location and Navigation	YES
Cycling Power	Cycling Power	YES
Scan Parameters	Scan Parameters	YES

Table 1 *Adopted Profile and Service support*

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

3.2 *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, pairing, 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.

API Features	Description
Interface to: GATT/GAP	Consistency between APIs including shared data formats.
GATT DB population and access	Full flexibility to populate the DB at run time, attribute removal is not supported.
Thread-safe, asynchronous, and event driven	Minimizes exposure to concurrency issues.
Vendor-specific (128 bit) UUIDs for proprietary profiles	Compact, fast, and memory efficient management of 128 bit UUIDs.
Packet flow control	

***Table 2** API features in the BLE stack*

GAP features	Description
Multi-role: Central and Observer	Up to 8 concurrent links supported as a Central.
Multiactivity	Observer can run concurrently with a central in a connection.
Multiple bond support	Keys and peer information stored in application space. No limitations in stack implementation.
Security mode 1: Levels 1, 2, and 3	Support for all levels of SM 1 independently per link.

***Table 3** GAP features in the BLE stack*

GATT Features	Description
Comprehensive GATT Server	Support for 8 concurrent ATT server sessions
Support for authorization: R/W characteristic value R/W descriptors	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 Support for 8 concurrent ATT client sessions
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 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 4 *GATT features in the BLE stack*

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

Table 5 *Security Manager (SM) features in the BLE stack*

ATT Features	Description
Server protocol	
Client protocol	

Table 6 *Attribute Protocol (ATT) features in the BLE stack*

Controller, Link Layer Features	Description
Central role Observer/Scanner/Initiator roles	The SoftDevice supports concurrent central connections and an additional Observer/Scanner/Initiator role. When the maximum number of simultaneous connections are established, the Observer and Scanner roles will be supported for new device discovery though the initiator is not available at that time.
Central connection update	
Encryption	

Table 7 *Controller, Link Layer (LL) features in the BLE stack*

Proprietary Feature	Description
TX Power control	Access for the application to change TX power settings anytime.
Channel map configuration	Setup of channel map for all connections from the application.

Table 8 *Proprietary features in the BLE stack*

3.3 Limitations on procedure concurrency

When there are multiple connections in the Central role, the concurrency of protocol procedures will have some limitations. The Host instantiates both GATT and GAP server instances for each connection, while the 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.

The limitations are outlined in ***Table 9*** below.

Protocol procedures	Limitation with multiple connections active
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, that is, each SM procedure must run to completion before the next procedure begins across all connections. For example sd_ble_gap_authenticate().
LL	LL Disconnect procedure has no limitations and can be executed on any, or all links simultaneously. LL connection parameter update and encryption establishment can only execute on one link at a time.

Table 9 *Procedure concurrency*

4 SoC library

The following features ensure the Application and SoftDevice coexist with safe sharing of common SoC resources.

Feature	Description
Mutex	Atomic mutex API. Disabling global interrupts in the application could cause dropped packets or lost connections. The API safely implements atomic mutex acquire and release operations for the application to use.
NVIC	Gives the application access to all NVIC features without corrupting SoftDevice configurations.
Rand	Gives access to the random number generator hardware.

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
	  • OFF
	  • ON
	  • AUTOMATIC - The SoftDevice will manage the DC/DC converter state by switching it on for all Radio Events and off all other times.

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.
Radio notification	Configure Radio Notification signals on ACTIVE and/or nACTIVE. See ***Chapter 7 "Radio Notification"*** on page 18.
Block encrypt (ECB)	Safe use of 128 bit AES encrypt HW accelerator.
Event API	Fetch asynchronous events generated by the SoC library.
Flash memory API	Application access to flash write, erase, and protect. Can be safely used during all protocol stack states.
Temperature	Application access to the temperature sensor.

Table 10 *System on Chip features*

5 SoftDevice Manager

The following feature enables 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 selects between the following options: • RC oscillator • Synthesized from high frequency clock • Crystal oscillator

Table 11 *SoftDevice Manager*

6 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.

6.1 BLE

The flash memory access is scheduled in between the protocol radio events. For short connection intervals, the time required for the flash memory access may be larger than the connection interval. In this case, protocol radio events may be skipped, up to a maximum of three events.

The flash memory access is scheduled in between the protocol radio events. For short connection intervals, the time required for the flash memory access may be larger than the connection or advertisement interval. In this case, protocol radio events may be skipped, up to a maximum of three connection events. The flash memory access may also be delayed slightly to minimize the disturbance of the BLE radio protocol. In some cases as described below, the flash memory access may fail and generate a timeout event: `NRF_EVT_FLASH_OPERATION_ERROR`. In this case, retry the flash erase or write operation.

BLE activity	Flash write
BLE Connectable Undirected Advertising BLE Nonconnectable Advertising BLE Scannable Advertising	Typically allows full write size (256 words) attempts.
BLE Connectable Directed Advertising	Does not allow write attempts while advertising is active (maximum 1.28 seconds). In this case, retrying flash writes will only succeed after the advertising activity has finished.
BLE Connected state	Typically allows full write size (256 words) attempts. May generate flash timeout event: `NRF_EVT_FLASH_OPERATION_ERROR` if critical radio events need to occur. Critical radio events are expected at connection setup, at connection update, at disconnection and just before supervision timeout. In this case, retry the flash write operation.
BLE activity	**Flash erase**
BLE Connectable Undirected Advertising BLE Nonconnectable Advertising BLE Scannable Advertising	Typically allows flash erase attempts.
BLE Connectable Directed Advertising	Does not allow flash erase attempts while advertising is active (maximum 1.28 seconds). In this case, retrying flash erase will only succeed after the directed advertising is finished.
BLE Connected state	Typically allows flash erase attempts. May generate flash timeout event: `NRF_EVT_FLASH_OPERATION_ERROR` if critical radio events need to occur. Critical radio events are expected at connection setup, at connection update, at disconnection and just before supervision timeout. In this case, retry the flash erase operation.

***Table 12** Behavior with BLE traffic and concurrent flash write/erase*

7 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 19*** on page 23.

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 and packet transfers. 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.

The following figures show the radio notification signal in relation to different combinations of active links and scanning events. See ***Table 13*** on page 17 for a description of notifications used in the figures and ***Chapter 10 "Multi-link Central role scheduling"*** on page 25 to understand the scheduling of links and scanner.

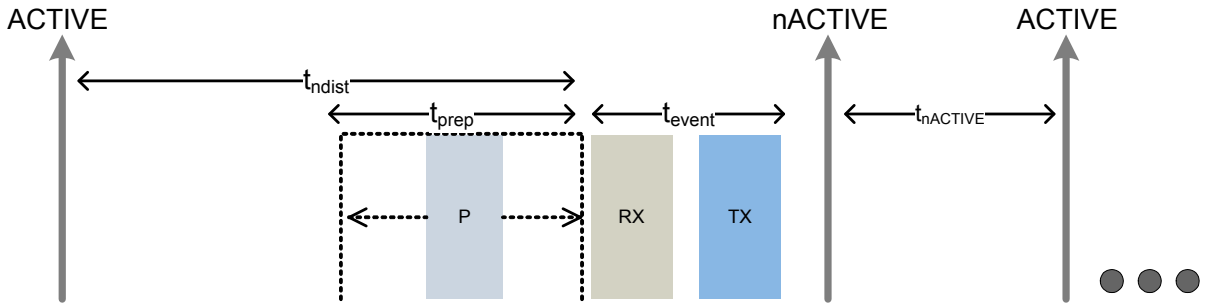

Figure 3 *BLE Radio Notification in relation to a single active link*

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

$$t_{ndist} + EEO + t_{prep(maximum)} < t_{interval}$$

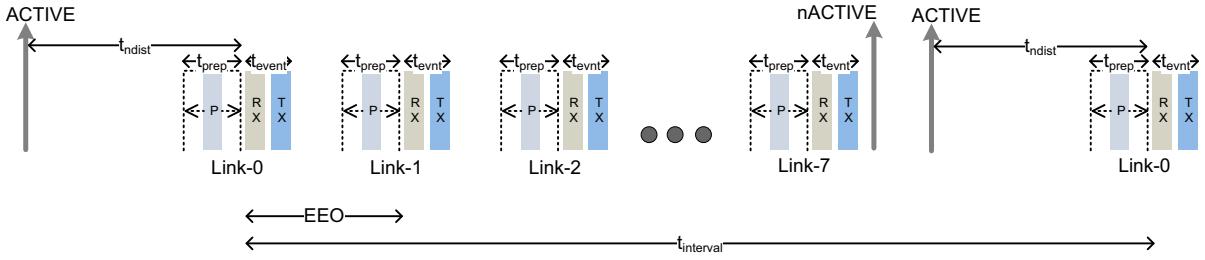

Figure 4 *BLE Radio Notification signal in relation to 8 active links*

To ensure the notification signal is available to the application at the configured time when 8 links are established, the following rule must be followed:

$$t_{ndist} + 8xEEO + t_{prep(maximum)} < t_{interval}$$

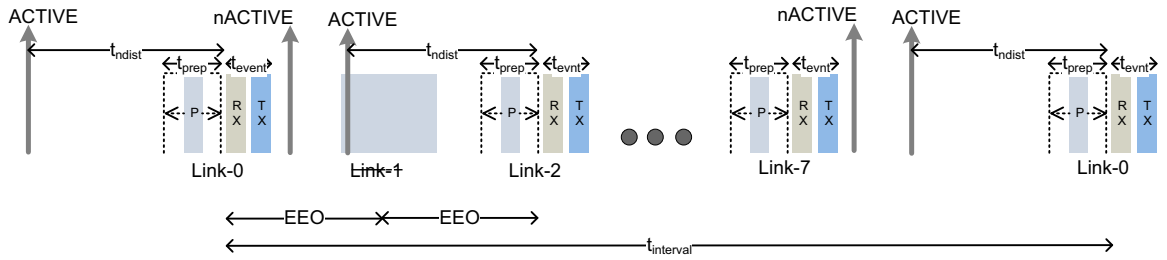

Figure 5 *BLE Radio Notification signal when the number of active links is greater than 1 but less than 8*

To ensure the notification signal is available to the application in the gap left by inactive links, the gap should be greater than t_{ndist}.

$$gap < t_{prep(maximum)} \text{ AND } gap > t_{ndist}$$

OR

$$gap \geq t_{prep(maximum)} \text{ AND } EEO + t_{prep(maximum)} > t_{ndist}$$

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

$$EEO > t_{ndist}$$

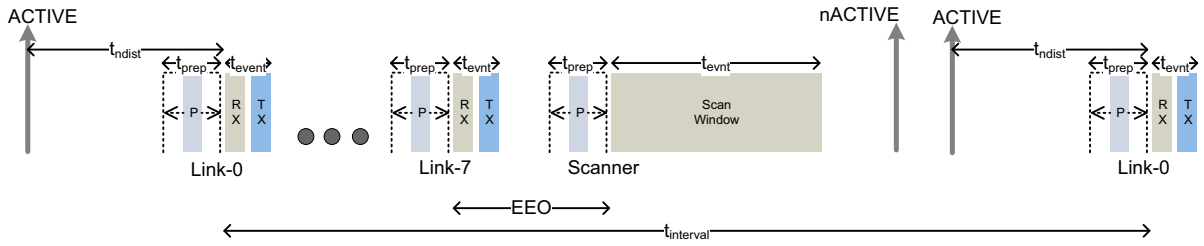

Figure 6 *BLE Radio Notification signal in relation to 8 active links and running scanner*

To ensure the notification signal is available to the application at the configured time with 8 links established and a scanner started, the following rule must be followed:

$$t_{ndist} + 9xEEO + \text{Scan\_window} < t_{interval}$$

Table 13 describes the notation used in the figures in this section.

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 t_{prep} before RX.
RX	Reception of packet.	
TX	Transmission of packet.	
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 and can be changed in between Radio Events.
t_{event}	The time used in a Radio Event.	
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}. **Note**: 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.	

Table 13 *Radio Notification figure labels*

Table 14 shows the ranges of the timing symbols in the figures in this section.

Value	Range (µs)
t_{ndist}	800, 1740, 2680, 3620, 4560, 5500 (Configured by the application)
t_{event}	Refer to ***Section 11.2 "Processor availability"*** on page 32
t_{prep}	290 to 1550
t_P	Refer to ***Section 11.2 "Processor availability"*** on page 32

Table 14 *BLE Radio Notification timing ranges*

Using the numbers from ***Table 14***, 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.

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

Note: 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 1500.

8 Bootloader

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

The use of a bootloader is supported in the SoftDevice architecture by dividing the application code space region (R1) into two separate regions. The lower region, from CLENR0 and upwards, contains the application, while the upper region contains the bootloader. The start of the upper region, the bootloader's base address, is set by the UICR.BOOTADDR register.

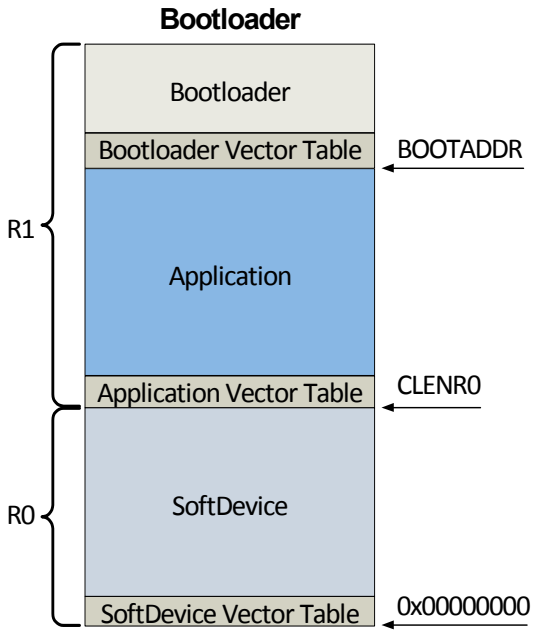

Figure 7 *R0 (SoftDevice) and R1 (Application + Bootloader).*

At reset, the SoftDevice checks the UICR.BOOTADDR register. If this register is blank (0xFFFFFFFF), the SoftDevice assumes that no bootloader is present. It then forwards interrupts to the application and executes the application as usual. If the BOOTADDR register is set to an address different from 0xFFFFFFFF, the SoftDevice assumes that the bootloader vector table is located at this address. Interrupts are then forwarded to the bootloader at this address and execution will be started at the bootloader reset handler.

For a bootloader to transfer execution from itself to the application, the bootloader should first call the `sd_softdevice_forward_to_application()` SoC function to forward interrupts to the application instead of to the bootloader. The bootloader should then branch to the application's reset handler after reading the address of the handler from the Application Vector Table.

UICR.BOOTADDR contents	Interpretation
0xFFFFFFFF	No bootloader present or enabled.
<ADDR>	Bootloader present with base address <ADDR>.

Table 15 *UICR.BOOTADDR register contents*

9 SoftDevice resource requirements

After the SoftDevice is installed on a System on Chip (SoC) it is located in the lower part of the code memory space. When enabled, the SoftDevice controls and uses resources from the chip, including reserving RAM space for its operation and access to hardware peripherals. This chapter describes how the SoftDevice – when both enabled and disabled - uses memory and hardware resources.

9.1 Memory resource map and usage

The memory map for program memory and RAM at run time with the SoftDevice enabled is illustrated in ***Figure 8*** below. Memory resource requirements, both when the SoftDevice is enabled and disabled, are shown in ***Table 16*** on page 21.

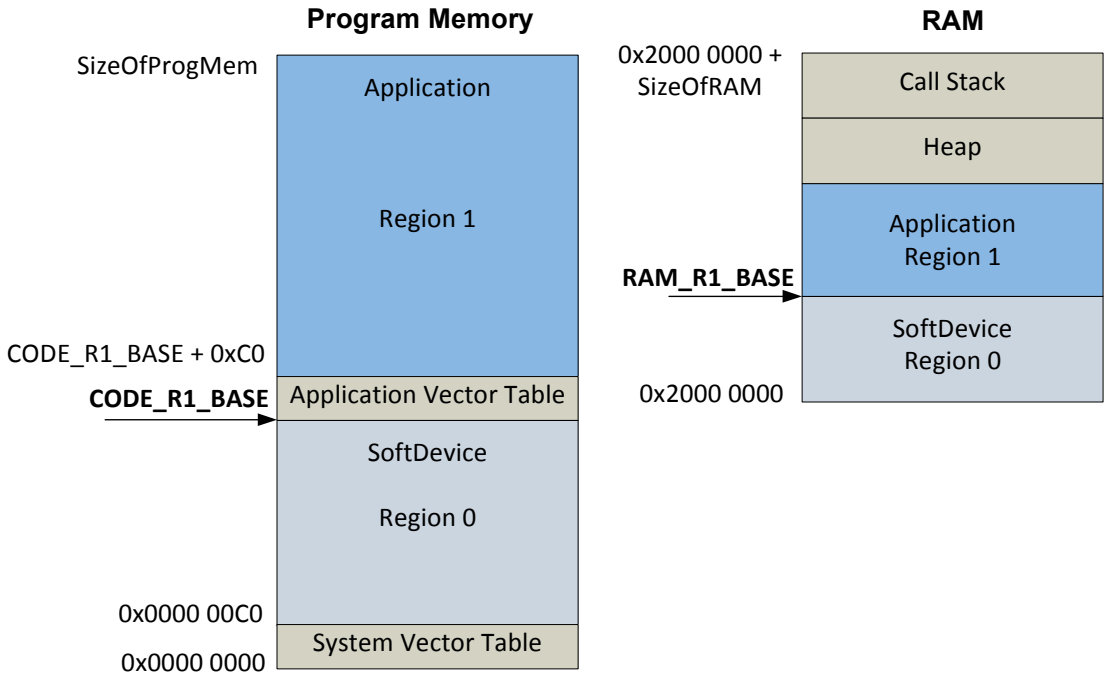

Figure 8 *Memory resource map*

Flash	S120 Enabled	S120 Disabled
Amount	96 kB	96 kB
CODE_R1_BASE	0x0001 8000	0x0001 8000

RAM	S120 Enabled	S120 Disabled
Amount	10 kB	4 bytes
RAM_R1_BASE	0x2000 2800	0x2000 0004

Call stack[1]	S120 Enabled	S120 Disabled
Maximum usage	1.5 kB (0x600)	0 bytes (0x00)

Heap	S120 Enabled	S120 Disabled
Maximum allocated bytes	0 bytes (0x00)	0 bytes (0x00)

1. This is only the callstack used by the SoftDevice at run time. The application call stack memory usage must be added for the total call stack size to be set in the user application.

Table 16 *S120 Memory resource requirements*

9.2 Hardware blocks and interrupt vectors

Table 17 defines access types used to indicate the availability of hardware blocks to the application. ***Table 18*** on page 22 specifies the access the application has, per hardware block, both when the SoftDevice is enabled and disabled.

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

Table 17 *Hardware access type definitions*

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/TWI1/SPIS1	Open	Open
		...		
6	0x40006000	GPIOTE	Open	Open
7	0x40007000	ADC	Open	Open
8	0x40008000	TIMER0	Blocked	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	Open
15	0x4000F000	AAR	Blocked	Open
16	0x40010000	WDT	Open	Open
17	0x40011000	RTC1	Open	Open
18	0x40012000	QDEC	Open	Open
19	00x4001300	LCOMP	Open	Open
20	0x40014000	Software interrupt	Open	Open
21	0x40015000	Radio Notification	Restricted[1]	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	Restricted	Open
NA	0x50000000	GPIO P0	Open	Open
NA	0xE000E100	NVIC	Restricted[2]	Open

1. Blocked only when radio notification signal is enabled. See ***Table 19*** on page 23 for software interrupt allocation.
2. 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.

Table 18 *Peripheral protection and usage by SoftDevice*

9.3 Application signals - software interrupts

Software interrupts are used by the SoftDevice to signal a change in events. ***Table 19*** shows the allocation of software interrupt vectors to SoftDevice signals.

Software interrupt (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.

Table 19 *Software interrupt allocation*

9.4 Programmable Peripheral Interconnect (PPI)

When the SoftDevice is enabled, the PPI is restricted with only some PPI channels and groups available to the application. ***Table 20*** shows how channels and groups are assigned between the application and SoftDevice.

Note: All PPI channels are available to the application when the SoftDevice is disabled.

PPI channel allocation	SoftDevice enabled	SoftDevice disabled
Application	Channels 0 - 7	Channels 0 - 15
SoftDevice	Channels 8 - 15	-

Preprogrammed channels	SoftDevice enabled	SoftDevice disabled
Application	-	Channel 20 - 31
SoftDevice	Channel 20 - 31	-

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

Table 20 *PPI channel and group availability*

9.5 SVC number ranges

Table 21 shows which SVC numbers an application program can use and which numbers are used by the SoftDevice.

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

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

Table 21 *SVC number allocation*

10 Multi-link Central role scheduling

The S120 stack supports up to eight Central role connections and an Observer or Scanner simultaneously. An Initiator can only be started if there are less than eight Central role connections established running.

10.1 Connection timing

The link scheduling system in the S120 SoftDevice adds Central link events relative to the first connected link. ***Figure 9*** shows a scenario where there are two links established. C0 events corresponds to the first Central connection made and C1 events corresponds to the second connection made. C1 events are initially offset from C0 events by "T" 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 ms.

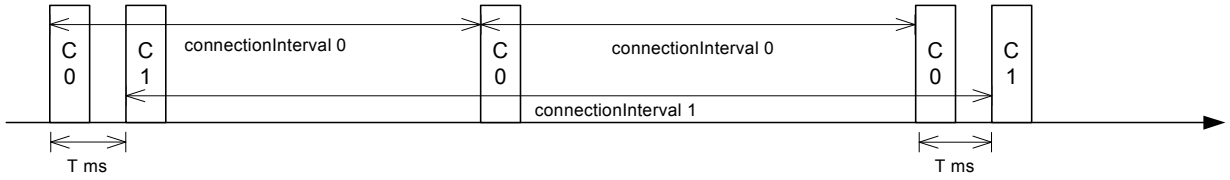

Figure 9 *Multi-link scheduling - one or more connections, factored intervals*

In ***Figure 10*** 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.

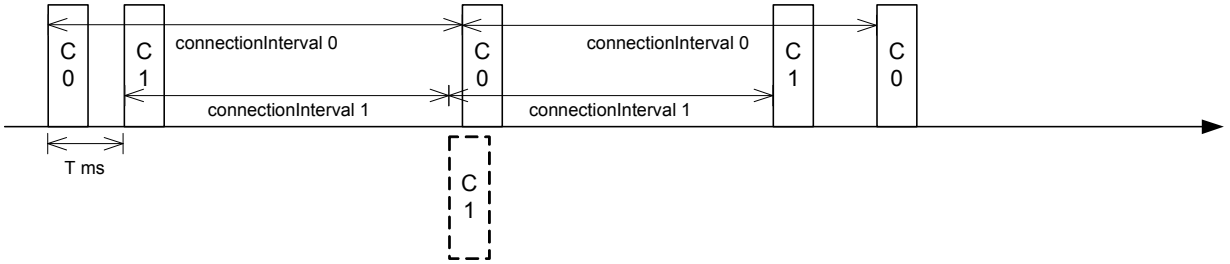

Figure 10 *Multi-link scheduling - one or more connections, unfactored intervals*

Figure 11 shows the maximum number of Central links possible at one time (8) with the minimum connection interval possible without having event collisions and dropped packets (20 ms). In this case, all available event time is used for the Central links.

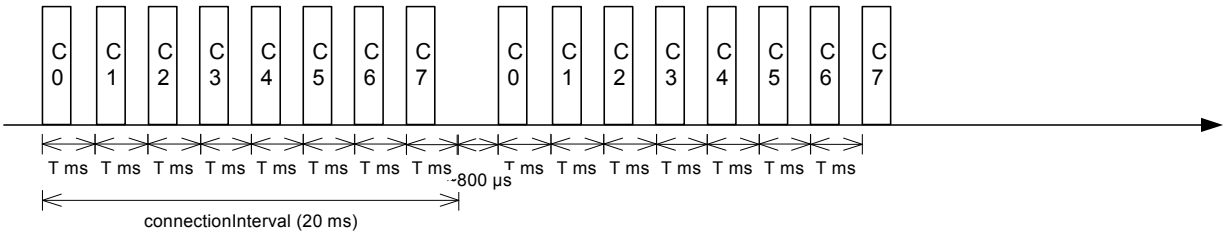

Figure 11 *Multi-link scheduling - max connections, min interval*

Figure 12 shows a scenario where the connInterval is longer than the minimum, and Central 2 and 4 have been disconnected or do not have events in this time period. It shows idle event time for each connection interval and the remaining Central connections maintain their timing offsets without the other links.

Figure 12 *Multi-link scheduling - max connections, interval > min*

10.2 Scanner timing

Figure 13 shows that when scanning for advertisers with no active connections, the scan interval and window can be any value within the *Bluetooth Core Specification*.

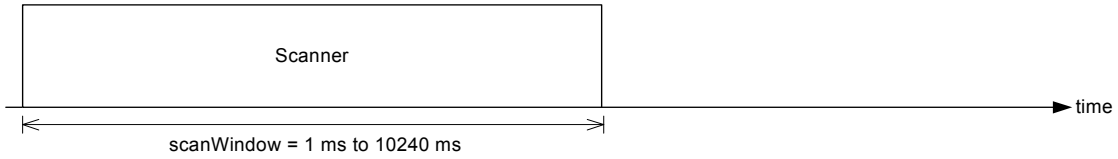

Figure 13 *Scanner timing - no active connections*

Figure 14 shows that when there is an active connection, the scanner or observer role will be started synchronously with the first connected Central link at a distance of 8(T) ms. With scanInterval equal to the connectionInterval and a scanWindow ≤ connectionInterval - (8*T + 0.8) ms, scanning will proceed without packet loss.

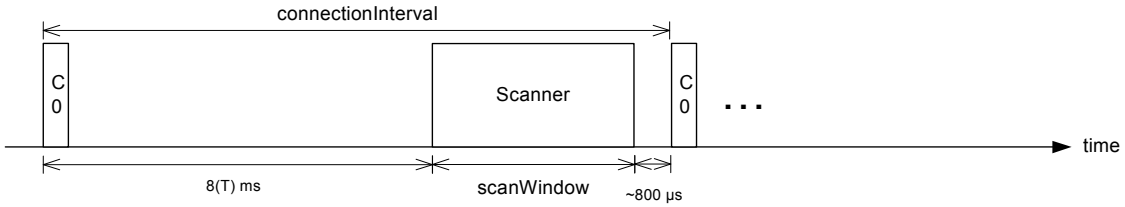

Figure 14 *Scanner timing - one connection*

Figure 15 shows a scanner with a long scanWindow which will cause some connection events to be dropped.

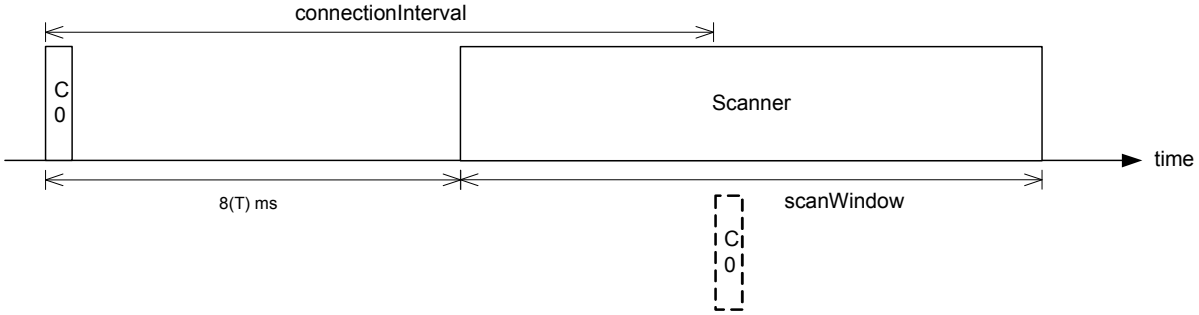

Figure 15 *Scanner timing - one connection, long window*

If all links have a short connection interval (20 ms) and the scanner is started, the scanner events will collide with Central link events causing packets on connections to be dropped as shown in ***Figure 16***.

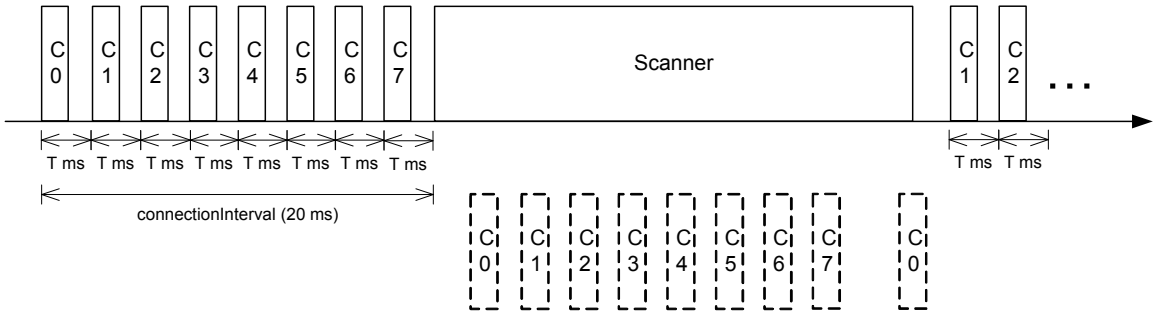

Figure 16 *Scanner timing - minimum connection interval*

10.3 Initiator timing

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 17***.

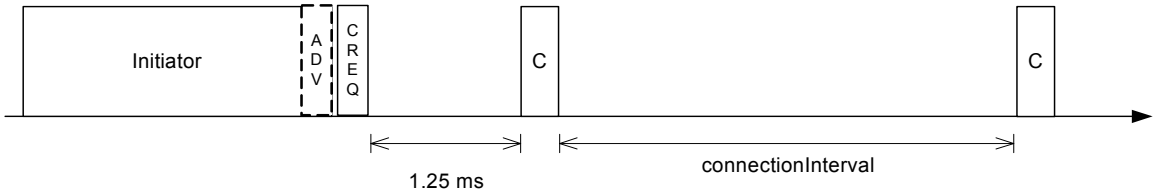

Figure 17 *Initiator - first connection*

When establishing a new connection with other connections already made, 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 events. ***Figure 18*** illustrates this when all existing connections have the same connection interval and the initiator starts around the same time as the 7th Central connection (C6) event in the schedule. The new connection, C2, is positioned in the available slot between C1 and C3.

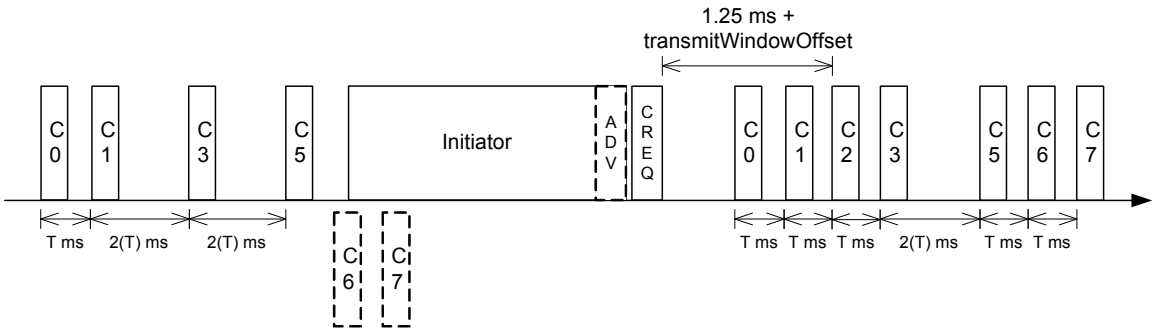

Figure 18 *Initiator - one or more connections*

When establishing connections to newly discovered devices, the scanner may be used for discovery followed by the initiator. In ***Figure 19***, 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. Central events scheduled in the transmit window offset will not be dropped (C3). In this case the 5th peer connection schedule is available (C4), and is allocated for the new connection.

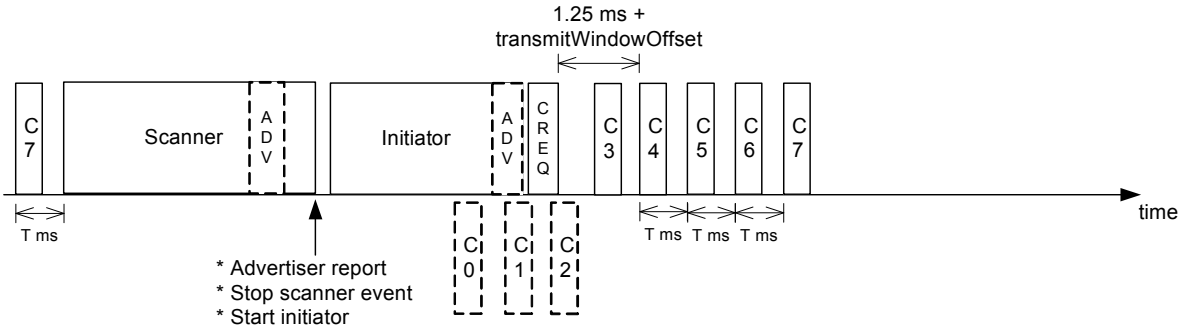

Figure 19 *Initiator - fast connection*

Note: It is not currently possible to schedule the initiator synchronously with connection events. In a future version of the SoftDevice, the initiator may have an option to start either synchronously, like the scanner, or asynchronously to prioritize a new connection.

10.4 Suggested intervals and windows

The value of "T" in the figures shown in previous sections is always 2.25 ms. This allows each Central link to send and receive one full length BLE packet before another event starts. There is an additional 750 µs before the first event of the first Central connection event in each case as shown in ***Figure 11*** on page 26. Therefore eight link events can complete in a minimum time of 8(2.25)+0.75 = 18.75 ms.

The minimum connection interval recommended for eight connections is 20 ms. Note that this does not leave time in the schedule for scanning or initiating new connections (when the number of connections already established is less than eight). Scanner, Observer, and Initiator events will therefore cause connection packets to be dropped as in ***Figure 16*** on page 28.

It is recommended that all connections have an interval equal to or some multiple of each other. In the case of using 20 ms as the shortest connection interval, all connections would have an interval of 20 ms or a multiple of 20 ms like 40 ms, 60 ms, 80 ms, etc.

If short connections 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 by having a connection interval of 50 ms or a multiple of 50 ms. In this case, eight connection events and a 31.25 ms scanner/initiator window can complete within each connection interval as in ***Figure 18*** on page 29.

To summarize, a recommended configuration for operation without dropped packets is:

1. The minimum Central connectionInterval should be ≥ 18.75 ms + scanWindow.
2. All Central connections should have a connectionInterval which can be factored by the smallest connection interval. For example [50 ms, 100 ms, 150 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 (as with the Central role) and the window should be ≤ connectionInterval – 20 ms.

If the application configures connection intervals between 7.5 ms and 20 ms and/or scan windows larger than recommended, the application should tolerate dropped packets by setting the supervision timeout for connections long enough to avoid loss of connection when packets are dropped because of Scanner, Observer, or Initiator events.

11 Processor availability and interrupt latency

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

11.1 Interrupt latency due to SoC framework

Latency, additional to ARM® Cortex™-M0 hardware architecture latency, is introduced by SoftDevice logic to manage interrupt events. This latency occurs when an interrupt is forwarded to the application from the SoftDevice and is part of the minimum latency for each application interrupt. The maximum application interrupt latency is dependent on protocol stack activity as described in ***Section 11.2 "Processor availability"*** on page 32.

Interrupt	CPU cycles	Latency at 16 MHz
Open peripheral interrupt	49	3.1 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	67	4.2 µs
Application SVC interrupt	43	2.68 µs

Table 22 *Additional latency due to SoftDevice processing*

See ***Table 18*** on page 22 for open, blocked, and restricted peripherals.

11.2 Processor availability

"Appendix A: SoftDevice architecture" in the *nRF51 Reference Manual* 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. ***Figure 20*** illustrates UpperStack activity (API calls) and LowerStack activity (Protocol events) and the time reserved/not reserved for those interrupts.

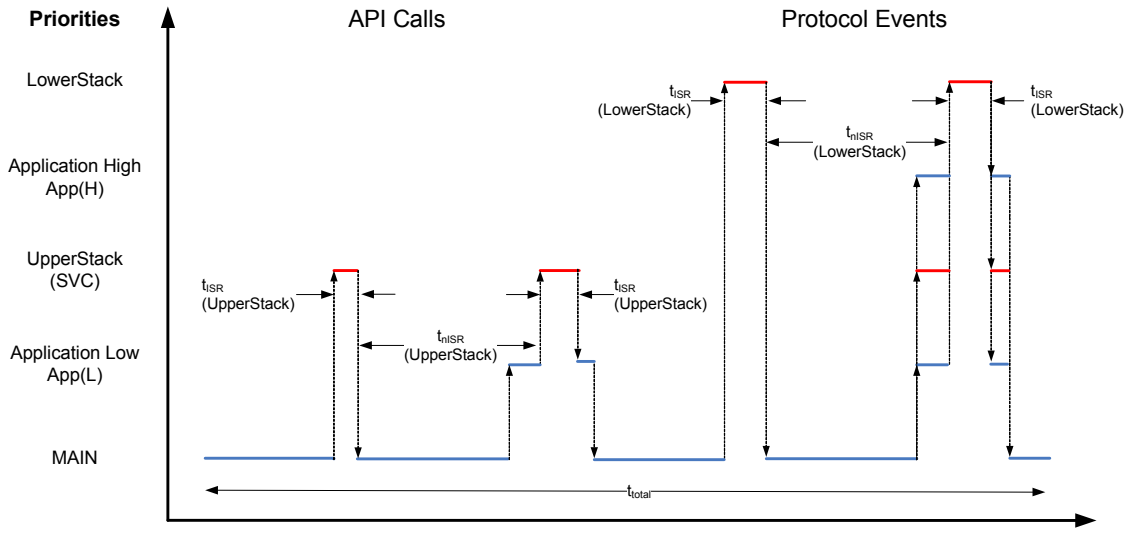

Figure 20 *UpperStack and LowerStack activity*

Table 23 describes the terms used for interrupt latency timings.

Parameter	Description
t_{ISR} (LowerStack)	Interrupt processing time in LowerStack. This is the 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 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.

Table 23 *SoftDevice interrupt latency definitions*

11.3 BLE central performance

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

During BLE protocol events, LowerStack interrupts are extended by a CPU Suspend state during radio activity to improve link integrity. This means 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.

11.3.1 BLE central scanning

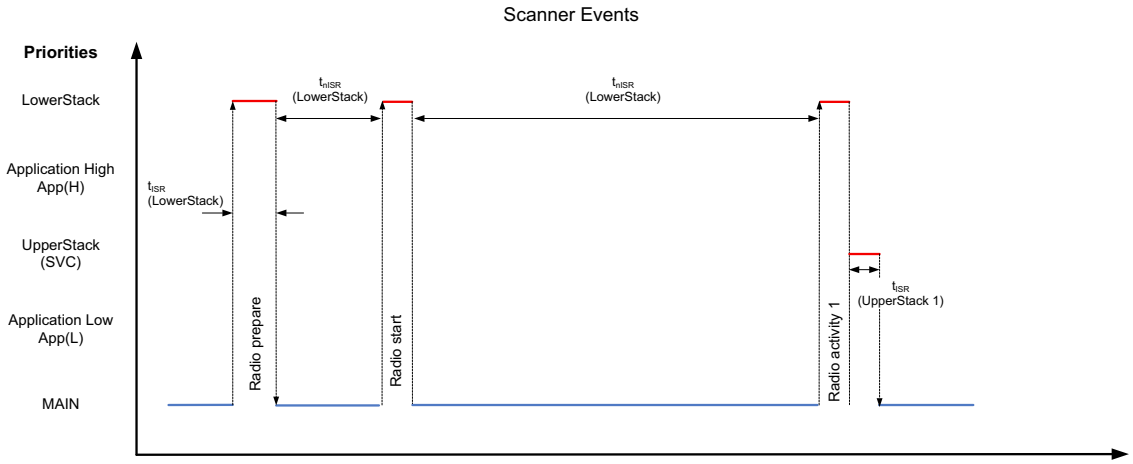

Figure 21 *Scanning, no packets received*

For scanning without packet reception, the pattern of LowerStack activity is as follows: there is first a Radio prepare, followed by a Radio start, then no activity during the scan interval and finally, at the end of the scan window, there is a Radio interrupt (Radio Activity 1) followed by some UpperStack activity. The CPU is not blocked during scanner receive.

Parameter	Description	Min.	Typ.	Max.
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for scanning event.			148 µs
$t_{ISR(LowerStack),RadioStart}$	Interrupt latency starting the scan or connection event.			41 µs
$t_{ISR(LowerStack),RadioActivity1}$	Interrupt latency at the end of a scanner event.		135 µs	165 µs
$t_{nISR(LowerStack)}$	Distance between interrupts during scanning	30 µs		Scan Window
$t_{ISR(UpperStack),1}$	UpperStack interrupt at the end of a scanner event.			68 µs

Table 24 *Interrupt latency when scanning, no packets received*

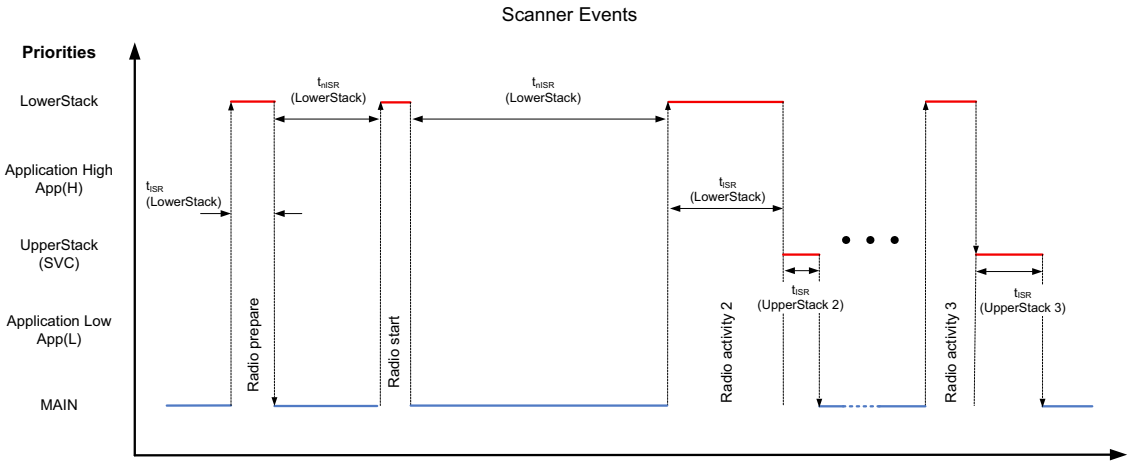

Figure 22 *Scanning, packets received*

When packets are received by the scanner, there is a different series of LowerStack activity as follows: Radio prepare and Radio start activity is the same. During active scanning the reception of an advertising packet will result in Radio activity (Radio activity 2) where the scan request is processed and sent and scan response is received and forwarded to the application via the UpperStack (UpperStack 2). During the scan request transmission and scan response reception, the CPU is blocked. Alternatively, if the scanner receives a connectable advertising packet, a connect request will be sent to the peer and a connected event generated for the application (Radio activity 3 and UpperStack 3).

Parameter	Description	Min.	Typ.	Max.
$t_{ISR(LowerStack),RadioActivity2}$	Interrupt latency for a scan request and response.	400 µs		680 µs[1]
$t_{ISR(LowerStack),RadioActivity3}$	Interrupt latency when receiving an ADV_IND or CONN_REQ packet.	100 µs		278 µs
$t_{ISR(UpperStack),2}$	UpperStack interrupt processing scan response.			160 µs
$t_{ISR(UpperStack),2}$	UpperStack interrupt processing a connect request.			300 µs

1. Maximum latency will be when scan response data size is maximum.

Table 25 *Interrupt latency when scanning, packets received*

11.4 BLE Central Connection Event

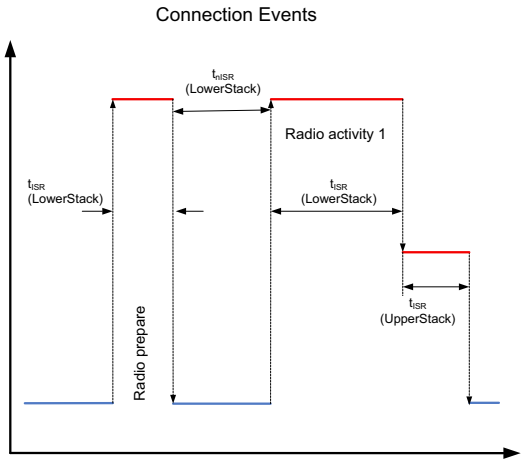

Figure 23 *Connection events*

In a connection event, the CPU is blocked by the S120 during the radio event (Radio activity 1) resulting in extended LowerStack interrupt length. The length of the transmitted and received data packets will affect this interrupt length. After each packet exchange, 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 are outlined in ***Table 26***.

Parameter	Description	Min.	Typ.	Max.
$t_{ISR(LowerStack),RadioPrepare}$	Interrupt latency preparing the radio for connection event.			140 µs
$t_{ISR(LowerStack),RadioActivity1}$	Interrupt length for data exchange and event scheduling.	680 µs		1860 µs[1]
$t_{ISR(UpperStack),Event}$	UpperStack interrupt processing for host to application events.		215 µs	
$t_{ISR(UpperStack),GATTRead/Write}$	UpperStack interrupt processing of GATTS operations, Read or Write and application event.		600 µs	
$t_{ISR(UpperStack),GATTReadByType}$	UpperStack interrupt processing of GATTS Read by Type.		790 µs	1130 µs[2]
$t_{ISR(UpperStack),Connect}$	UpperStack interrupt processing of connection establishment and generating the connected event.		1630 µs	
$t_{nISR(LowerStack)}$	Distance between connection event interrupts.	30 µs		1400 µs

1. Variation in the time taken for data exchange varies with packet length. The scheduler will take variable time if events have collided in time.
2. Read by type operations vary with ATT Database configuration and size.

Table 26 *Interrupt latency when connected*

Table 27 shows expected CPU utilization percentages for the upper stack and lower stack given a set of typical stack connection parameters.

Note: Upper stack utilization is based only on the processing required to update the database and transfer data to and from the application when the data is transferred.

BLE connection configuration	Lower stack	Upper stack	CPU suspend	Remaining
Connection interval 1s, 8 connections, 1 packet transfer per event (bi-directional).	1%	0.5%	0.5%	98%
Connection interval 20 ms, 8 connections, 1 packet transfer per event (bi-directional)	45%	25%	30%	~0%
Connection interval 100 ms 1 packet transfer per event (bi-directional)	10%	5%	5%	~80%

Table 27 *Processor usage and remaining availability for example BLE connection configurations*

11.5 Performance with Flash memory API

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

12 BLE data throughput

The maximum data throughput limits in ***Table 28*** apply to encrypted packet transfers. 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 S120 SoftDevice limits packet transmission and reception to 1 packet each per connection event. All numbers displayed below are per connection.

Protocol	Role	Method	Number of connected slaves	Interval (ms)	Maximum data throughput
GATT	Client	Receive Notification	1 - 8 1 - 8	20 50	7.8 kbps 3.1 kbps
		Send Write command	1 - 8 1 - 8	20 50	7.8 kbps 3.1 kbps
		Send Write request	1 - 8 1 - 8	20 50	3.8 kbps 1.5 kbps
		Simultaneous receive Notification and send Write command	1 1 - 8 1 - 8	7.5 20 50	21 kbps (each direction) 7.8 kbps (each direction) 3.1 kbps (each direction)
GATT	Server	Send Notification	1 - 8 1 - 8	20 50	7.8 kbps 3.1 kbps
		Receive Write command	1 - 8 1 - 8	20 50	7.8 kbps 3.1 kbps
		Receive Write request	1 - 8 1 - 8	20 50	3.9 kbps 1.5 kbps
		Simultaneous send Notification and receive Write command	1 1 - 8 1 - 8	7.5 20 50	21 kbps (each direction) 6.7 kbps (each direction) 3.1 kbps (each direction)

Table 28 *L2CAP and GATT maximum data throughput*

13 BLE power profiles

This chapter provides power profiles for MCU activity during *Bluetooth* low energy Radio Events implemented in the SoftDevice. These profiles give a detailed overview of the stages of a Radio Event, the approximate timing of stages within the event, and how to calculate the peak current at each stage using data from the product specification. The LowerStack CPU profile (including CPU activity and CPU suspend) during the event is shown separately. These profiles are based on events with empty packets.

13.1 Connection event

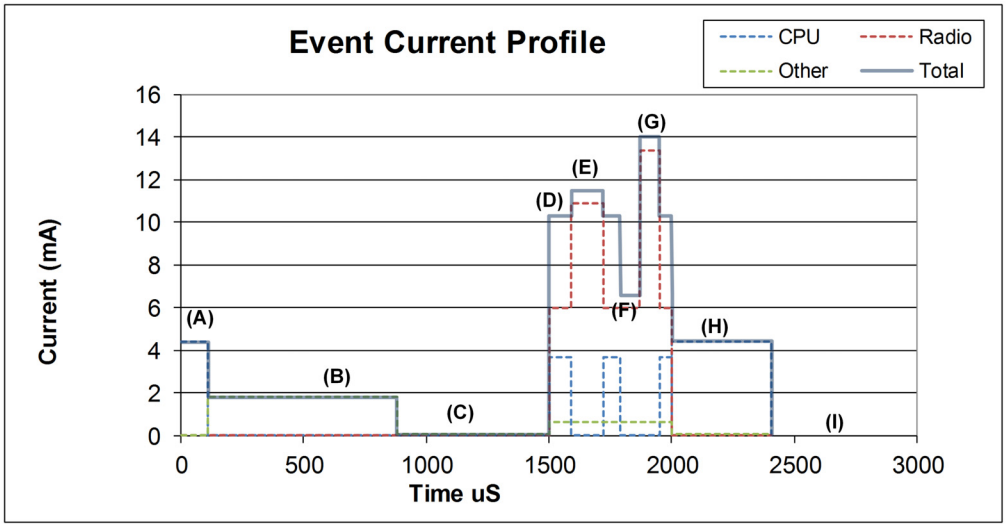

Figure 24 *Connection event*

Stage	Description	Current Calculations[1]
(A)	Preprocessing	$ION + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(B)	Standby + XO ramp	$ION + I_{RTC} + I_{X32k} + I_{START,X16M}$
(C)	Standby	$ION + I_{RTC} + I_{X32k} + I_{X16M}$
(D)	Radio Start	$ION + I_{RTC} + I_{X32k} + I_{X16M} + \int (I_{START,TX})$
(E)	Radio TX	$ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{TX,0dBM}$
(F)	Radio turn-around	$ION + I_{RTC} + I_{X32k} + I_{X16M} + \int (I_{START,RX})$
(G)	Radio RX	$ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}$
(H)	Post-processing	$ION + I_{RTC} + I_{X32k} + I_{CPU,Flash}$
(I)	Idle - connected	$ION + I_{RTC} + I_{X32k}$

1. See the corresponding product specification for the symbol values.

Table 29 *Advertising event*

14 SoftDevice identification and revision scheme

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

For revisions of the SoftDevice which are production qualified, the version string consists of major, minor, and revision numbers only, as described in ***Table 30***.

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

For example: s110_nrf51822_1.2.3-4.alpha, where major = 1, minor = 2, revision = 3, build number = 4 and test qualification level is alpha. Additional SoftDevice revision examples are given in ***Table 31***.

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 Revision 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 version.

Table 30 *Revision scheme*

Sequence number	Description
s110_nrf51822_1.2.3-1.alpha	Revision 1.2.3, first build, qualified at alpha level
s110_nrf51822_1.2.3-2.alpha	Revision 1.2.3, second build, qualified at alpha level
s110_nrf51822_1.2.3-5.beta	Revision 1.2.3, fifth build, qualified at beta level
s110_nrf51822_1.2.3	Revision 1.2.3, qualified at production level

Table 31 *SoftDevice revision examples*

The test qualification levels are outlined in ***Table 32.***

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 but may not be feature complete and may contain known issues. Protocol implementations are tested for conformance and interoperability.
Production	Qualified release suitable for product integration. In addition to beta qualification: Hardware integration tested over supported range of operating conditions. Stable and complete with no known issues. Protocol implementations conform to standards.

Table 32 *Test qualification levels*

14.1 Notification of SoftDevice revision updates

When new versions of a SoftDevice become available or the qualification status of a given revision of a SoftDevice is changed, product update notifications will be automatically forwarded, by email, to all users who have a profile configured to receive notifications from the Nordic Semiconductor website.

The SoftDevice will be updated with additional features and/or fixed issues if needed. Supported production versions of the SoftDevice will remain available after updates, so products do not need requalification on release of updates if the previous version is sufficiently feature complete for your product.