S110 nRF51822

Bluetooth® low energy

SoftDevice Specification v1.2

Key Features

- *Bluetooth*® 4.0 compliant low energy single-mode protocol stack
    - Link layer
    - L2CAP, ATT, and SM protocols
    - GATT, GAP, and L2CAP
    - Peripheral and Broadcaster roles
    - GATT Client and Server
    - Full SMP support 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

- Computer peripherals and I/O devices
    - Mouse
    - Keyboard
    - Multi-touch trackpad
- Interactive entertainment devices
    - Remote control
    - 3D glasses
    - Gaming controller
- Personal Area Networks
    - Health and fitness sensor and monitor devices
    - Medical devices
    - Key fobs and wrist watches
- Remote control toys

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 product specification contains target specifications for product development.
v0.7	This product specification contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
v1.0	This product 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
November 2013	1.2	Updated for S110 v6.0.0 release. Added ***Chapter 6 "Flash memory API"*** on page 13; Added ***Chapter 8 "Bootloader"*** on page 17 Updated ***Table 1*** on page 7; Updated ***Table 4*** on page 9; Updated ***Table 10*** on page 11; Updated ***Chapter 7 "Radio Notification"*** on page 14; Updated ***Table 17*** on page 19.
March 2013	1.1	Updated for changes made as of S110 v5.0.0; Changed ***Section 10.2 "Processor availability"*** on page 24 and ***Section 11 "Power profiles"*** on page 27; Changed ***Table 23*** on page 24; Added ***Table 24*** on page 25; Changed ***Table 26*** on page 26; Changed ***Figure 9*** on page 28 and ***Figure 10*** on page 29.
February 2013	1.0	Changed Memory resource requirements in ***Table 16*** on page 19; Added ***Section 9.3 "Application signals - software interrupts"*** on page 21; Updated ***Chapter 10 "BLE performance"*** on page 23 and added ***Section 10.3 "Data throughput"*** on page 26; Updated diagrams in ***Chapter 11 "Power profiles"*** on page 27; Added ***Chapter 12 "SoftDevice identification and revision scheme"*** on page 30; Updated ***Chapter 12.1 "Notification of SoftDevice revision updates"*** on page 31.
September 2012	0.6	First release.

1 Introduction

The S110 SoftDevice is a *Bluetooth*® low energy (BLE) Peripheral protocol stack solution. It integrates a low energy controller and host, and provides a full and flexible 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 12.1 "Notification of SoftDevice revision updates"*** on page 31 for more information. To find information on any limitations or omissions, the SoftDevice release notes will contain a detailed summary of the release status.

1.1 Documentation

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.
nRF51822 PS	Contains a description of the hardware, modules, and electrical specifications specific to the nRF*51822* chip.
Bluetooth Core Specification	The *Bluetooth Core Specification* version 4.0, 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 description of SetChannelPeriod will be written as: SetChannelPeriod(*ChannelNumber, MessagingPeriod*).
- 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 S110 SoftDevice.

2.1 SoftDevice

The S110 SoftDevice is a pre-compiled and linked binary software implementing a *Bluetooth* 4.0 low energy (BLE) protocol stack on the nRF51822 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 S110 SoftDevice.

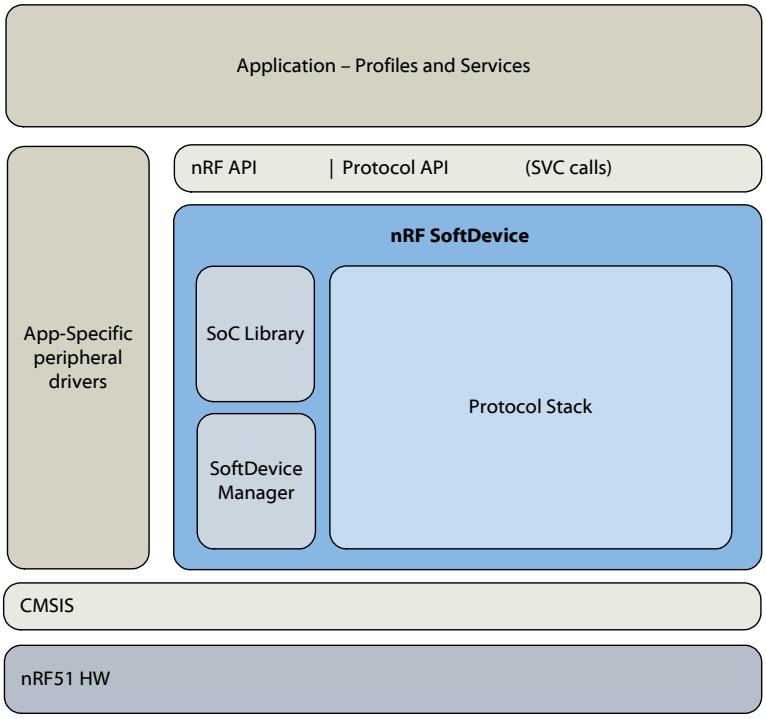

Figure 1 *System on Chip application with the SoftDevice*

The S110 SoftDevice can be programmed onto compatible nRF51 devices during both development and production. This specification outlines the supported features of a production S110 SoftDevice. Alpha and Beta versions may not support all features.

2.2 Multiprotocol support

The S110 SoftDevice supports non-concurrent multiprotocol implementations. This means a proprietary 2.4 GHz protocol can be implemented in the application program area. This protocol can access all hardware resources when the S110 SoftDevice is disabled.

3 *Bluetooth* low energy protocol stack

The *Bluetooth* 4.0 compliant low energy (BLE) Host and Controller embedded in the SoftDevice are fully qualified with multi-role support (Peripheral and Broadcaster). 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) completes 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).

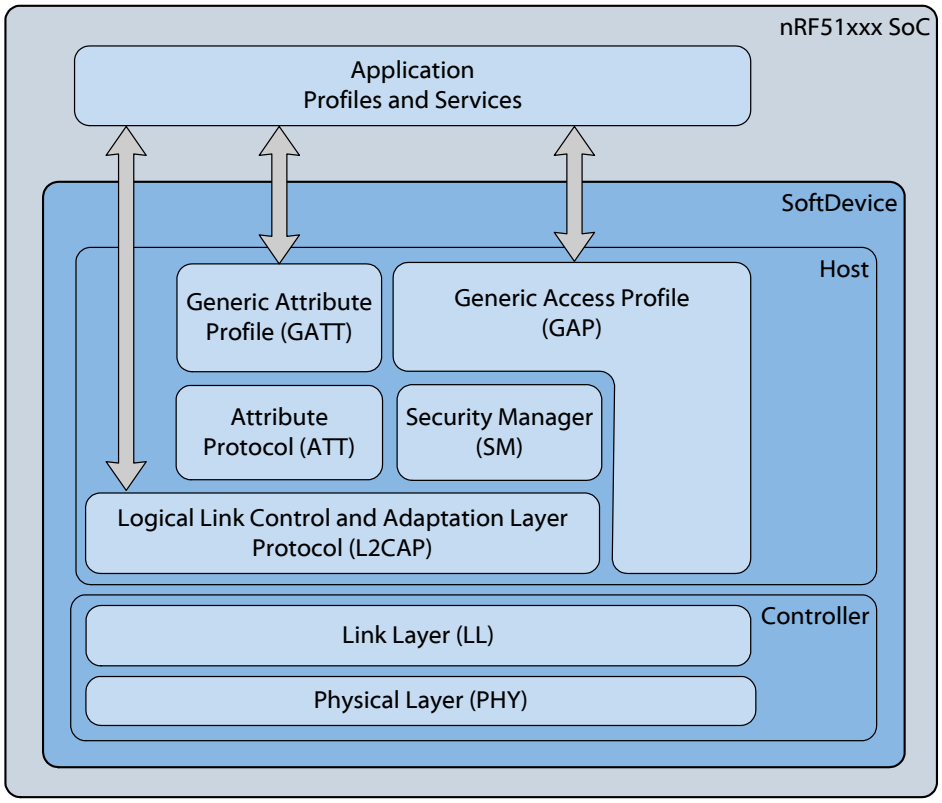

Figure 2 *SoftDevice stack architecture*

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
Proximity	Link Loss Immediate Alert TX Power	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. GAP and GATT procedures and modes that are common to most profiles, such as the handling of discoverability, connectability, pairing, and bonding, are implemented in the stack.

The BLE Application Programming Interface (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/L2CAP	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	Zero-copy buffer management.

Table 2 *API features in the BLE stack*

GAP Features	Description
Multi-role: Peripheral and Broadcaster	
Limited and general discoverable modes	
Multiple bond support	Keys and peer information stored in application space. No limitations in stack implementation.
User-defined Advertising data	Full control over advertising and scan response data for the application.
Security mode 1: Levels 1, 2, and 3	

Table 3 *GAP features in the BLE stack*

GATT Features	Description
Comprehensive GATT Server	
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
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
Lightweight key storage for reduced NV memory requirements	
Authenticated MITM (Man in the middle) protection	
Pairing methods: Just works, Passkey Entry, and Out of Band	

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*

L2CAP Features	Description
27 byte MTU size	
Dynamically allocated channels	

Table 7 *Logical Link Control and Adaptation Layer Protocol (L2CAP) features in the BLE stack*

Controller, Link Layer Features	Description
Slave role	
Slave connection update	
27 byte MTU	
Encryption	

Table 8 *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.
Application Latency	Enhanced management of power versus response time of an application to data sent from a peer device.
Resolvable address whitelisting based on IRK	Synchronous and low power solution for BLE enhanced privacy.

Table 9 *Proprietary features in the BLE stack*

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. This API safely implements an atomic operation 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 hook 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 INACTIVE. See ***Section 12.1 "Notification of SoftDevice revision updates"*** on page 31.
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 also safely be used during active BLE connections.
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.

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.

7 Radio Notification

The Radio Notification is a configurable feature which enables ACTIVE and INACTIVE (nACTIVE) signals from the SoftDevice to the application to notify when the Radio will be in use. The signal is sent using software interrupt, as specified in ***Section 7.3 "Application signals - software interrupts"*** on page 20.

The ACTIVE signal, if enabled, is sent before the Radio Event starts. The INACTIVE 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.

Figure 3 shows the active signal in relation to the Radio Event.

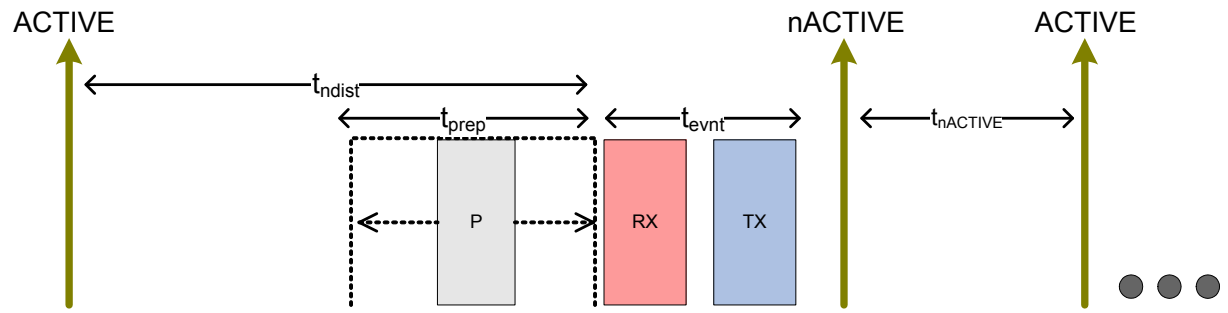

Figure 3 *Radio Notification*

Many packets can be sent and received in one Radio Event. Radio Notification events will be as shown in ***Figure 4***.

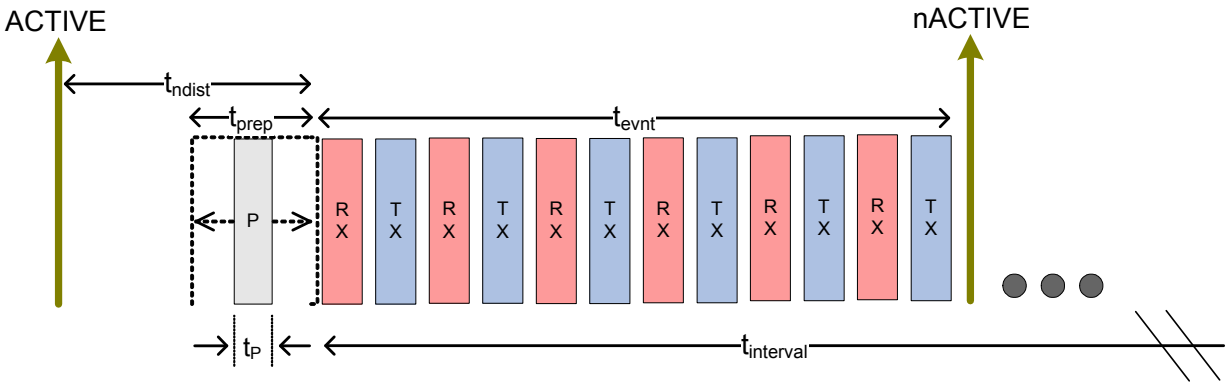

Figure 4 *Radio Notification, multiple packet transfers*

Table 12 describes the notation used in ***Figure 3*** and ***Figure 4*** on page 14.

Label	Description	Notes
ACTIVE	The ACTIVE signal prior to a Radio Event.	
nACTIVE	The INACTIVE signal after a Radio Event.	Because both ACTIVE and INACTIVE use the same software interrupt, it is up to the application to manage them. If both ACTIVE and INACTIVE are configured ON by the application, there will always be an ACTIVE signal before an INACTIVE 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_{evnt}	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 CPU processing in LowerStack interrupt.	
$t_{interval}$	Time between Radio Events as per the protocol.	

Table 12 *Radio Notification figure labels*

Table 13 shows the ranges of the timing symbols in ***Figure 3*** on page 14.

Value	Range (µs)
t_{ndist}	800, 1740, 2680, 3620, 4560, 5500
t_{evnt}	550 to 1300 Advertiser - 0 to 31 bytes payload, 3 channels 900 to 5400 Slave - 1 to 6 packets RX and TX unencrypted data when connected 1000 to 5800 Slave - 1 to 6 packets RX and TX encrypted data when connected
t_{prep}	200 to 1500
t_P	≤150

Table 13 *Radio Notification timing ranges*

Using the numbers from ***Table 13***, the amount of CPU time available between ACTIVE and a Radio Event is:

$$t_{ndist} - t_P$$

Shown here is the amount of time before stack interrupts begin. 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 larger 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 larger than 1500.

To ensure the notification signal is available to the application in the configured time, the following rule is applied:

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

To ensure this rule is true, the stack may limit the length of a Radio Event, t_{evnt}, thereby reducing the maximum throughput and effecting maximum data rate. ***Figure 5*** shows consecutive Radio Events with Radio Notification and illustrates the limitation in t_{evnt} which may be required to ensure t_{ndist} is preserved.

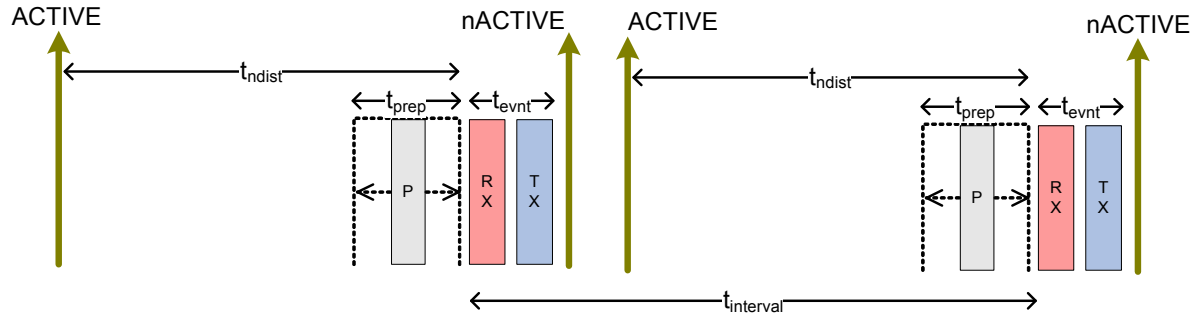

Figure 5 *Consecutive Radio Events with Radio Notification*

Table 14 shows the limitation on the maximum number of packets which can be transferred per Radio Event given a t_{ndist} and $t_{interval}$ combination.

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

Table 14 *Maximum packet transfer per Radio Event for given combinations of t_{ndist} and $t_{interval}$.*

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.

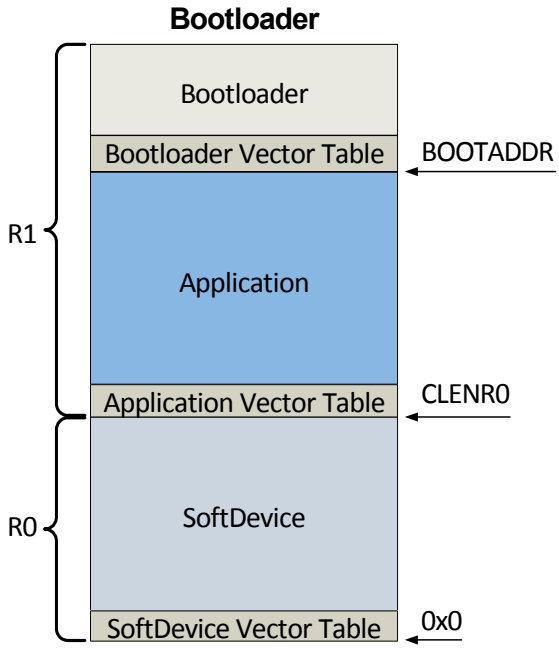

Figure 6 *R0 (SoftDevice) and R1 (Application + Bootloader).*

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 S110 resource requirements

The S110 SoftDevice uses on-chip resources including memory, system blocks, and peripheral blocks. The use of resources may change depending on if the SoftDevice is enabled or disabled. This chapter outlines how memory and hardware are used by the SoftDevice.

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 7*** below. Memory resource requirements, both when the SoftDevice is enabled and disabled, are shown in ***Table 16*** on page 19.

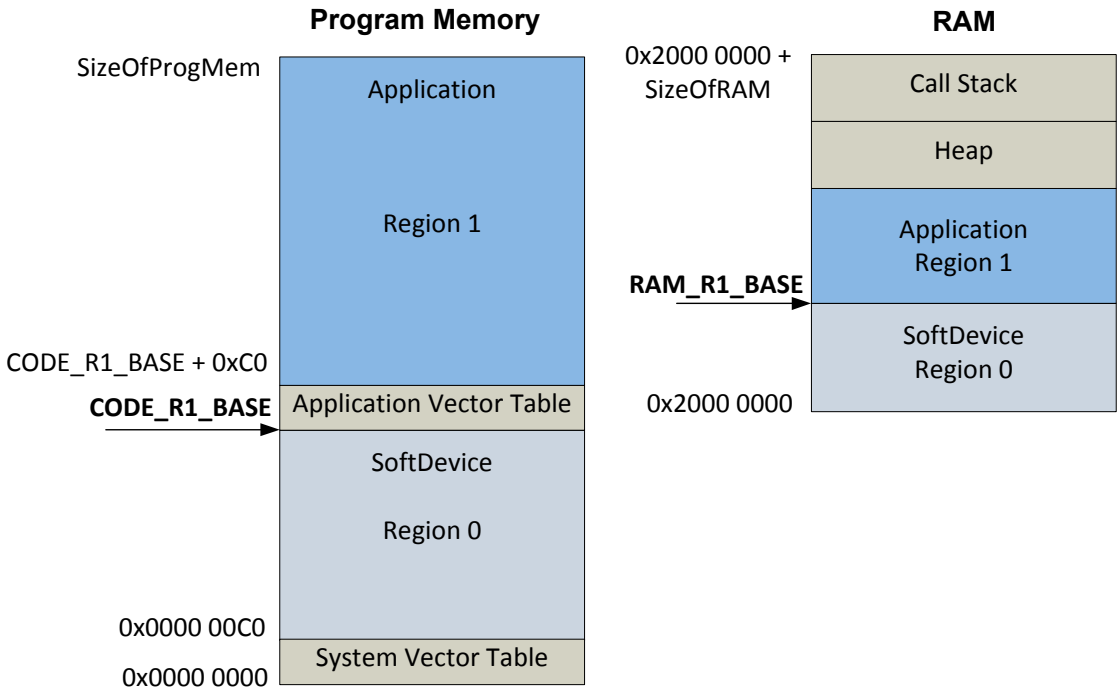

Figure 7 *Memory resource map*

Flash	S110 Enabled	S110 Disabled
Amount	80 kB	80 kB
CODE_R1_BASE	0x0001 4000	0x0001 4000

RAM	S110 Enabled	S110 Disabled
Amount	8 kB	4 bytes
RAM_R1_BASE	0x2000 2000	0x2000 0004

Call stack[1]	S110 Enabled	S110 Disabled
Maximum usage	1.5 kB	0 kB

Heap	S110 Enabled	S110 Disabled
Maximum allocated bytes	0 bytes	0 bytes

1. This is only the call stack 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 *S110 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 20 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 (S110 enabled)	Access (S110 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	Port 0 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	0x40013000	LCOMP	Open	Open
20	0x40014000	Software interrupt	Open	Open
21	0x40015000	SoC Radio Notification Events	Blocked	Open
22	0x40016000	ANT/BLE/SoC Events	Blocked	Open
23	0x40017000	Software interrupt	Restricted[1]	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 signals are configured. See ***Table 20*** on page 21 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 S110 SoftDevice to signal the application of 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	S110 enabled	S110 disabled
Application	Channels 0 - 7	Channels 0 - 15
SoftDevice	Channels 8 - 15	-

PPI group allocation	S110 enabled	S110 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	S310 enabled	S310 disabled
Application	0x00-0x0F	0x00-0x0F
SoftDevice	0x10-0xFF	0x10-0xFF

Table 21 *SVC number allocation*

10 BLE performance

This chapter documents key SoftDevice performance parameters for interrupt latency, processor availability, and data throughput with regards to the BLE stack.

10.1 Interrupt latency

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 10.2 "Processor availability"*** on page 24.

Interrupt	CPU cycles	Latency at 16 MHz
Open peripheral interrupt	50	3.2 µs
Blocked or restricted peripheral interrupt (only forwarded when SoftDevice disabled)	63	4 µs
Application SVC interrupt	14	1 µs

Table 22 *Additional latency due to SoftDevice processing*

See ***Table 18*** on page 20 for open, blocked, and restricted peripherals.

10.2 Processor availability

Appendix A: SoftDevice architecture in the *nRF51 Reference Manual* describes interrupt management in SoftDevices and is required knowledge for understanding this section.

Shown in ***Figure 8*** are the parameter values from ***Table 23***. The parameters are defined around LowerStack and UpperStack interrupts. These interrupts service real time protocol events and API calls (or deferred internal SoftDevice tasks) respectively. 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 Event for a time proportional to the number of packets transferred in the event. See ***Table 24*** on page 25 for more information.

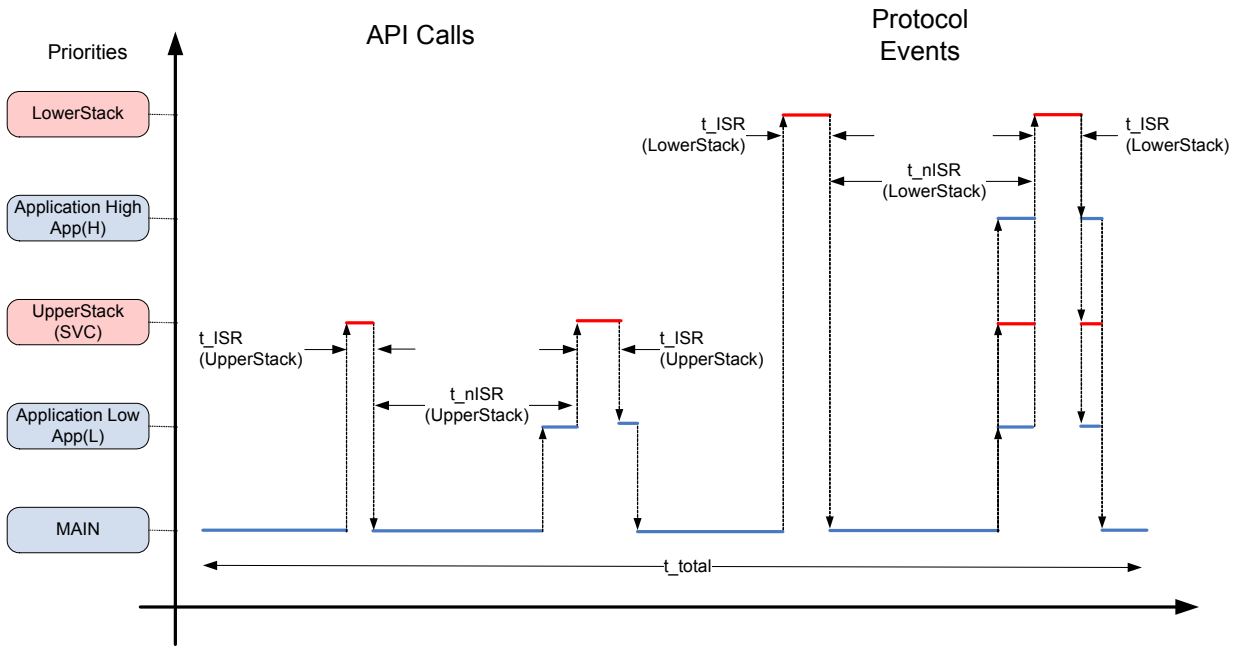

Figure 8 *Interrupt latencies due to SoftDevice processing*

		UpperStack		
Parameter	**Description**	**Min**	**Nom**	**Max**
$t_{ISR(upper\ stack)}$	Maximum interrupt processing time	-	-	250 µs
$t_{nISR(upper\ stack)}$	Minimum time between interrupts	Application dependent.[1]		

1. Calls to the SoftDevice API trigger the upper stack interrupt.

Table 23 *SoftDevice interrupt latency - UpperStack*

Parameter	Description	Packets	Nominal
$t_{ISR(lower\ stack)1}$	Maximum interrupt latency during Radio Event. Includes the time the CPU is used by the LowerStack for processing and the time the CPU suspended during radio activity. In each case, maximum encrypted packet length in both RX and TX are assumed.	1	1180 µs
$t_{ISR(lower\ stack)\ 2}$		2	2136 µs
$t_{ISR(lower\ stack)\ 3}$		3	3092 µs
$t_{ISR(lower\ stack)\ 4}$		4	4048 µs
$t_{ISR(lower\ stack)\ 5}$		5	5004 µs
$t_{ISR(lower\ stack)\ 6}$		6	5960 µs
$tn_{ISR(lower\ stack)}$	Minimum time between LowerStack interrupts.	n/a	1390 µs

Table 24 *SoftDevice interrupt latency LowerStack*

Application Low, App(L), can be blocked by the SoftDevice for a maximum of:

$$App(L)_{latency_{max}} = t_{ISR_{max(Upper\ Stack)}} + t_{ISR_{max(Lower\ Stack)}}$$

Application High, App(H), can be blocked by the SoftDevice for a maximum of:

$$App(H)_{latency\_max} = t_{ISR_{max(Lower\ Stack)}}$$

Table 25 shows expected CPU utilization percentages for the UpperStack and LowerStack given a set of typical stack connection parameters.

Note: UpperStack 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	LowerStack	UpperStack	CPU suspend	Remaining
Connection interval 4 s No data transfer	0.01%	0.01%	0.03%	~99%
Connection interval 100 ms 1 packet transfer per event	0.4%	0.6%	0.7%	~98%
Connection interval 7.5 ms 4 packet transfer per event	11%	27%	42%	~20%

Table 25 *Processor usage and remaining availability for example BLE connection configurations*

10.3 Data throughput

The maximum data throughput limits in ***Table 26*** 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.

Protocol	Role	Method	Maximum data throughput
L2CAP		Receive	140 kbps
		Send	140 kbps
		Simultaneous send and receive	130 kbps (each direction)
GATT	Client	Receive Notification	120 kbps
		Send Write command	120 kbps
		Send Write request	10 kbps
		Simultaneous receive Notification and send Write command	100 kbps (each direction)
GATT	Server	Send Notification	120 kbps
		Receive Write command	110 kbps
		Receive Write request	10 kbps
		Simultaneous send Notification and receive Write command	80 kbps (each direction)

Table 26 *L2CAP and GATT maximum data throughput*

11 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.

11.1 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} + $\int (I_{START,RX})$
(E)	Radio RX	ION + I_{RTC} + I_{X32k} + I_{X16M} + I_{RX}+ I_{CRYPTO}
(F)	Radio turn-around	ION + I_{RTC} + I_{X32k} + I_{X16M} + $\int (I_{START,TX})$
(G)	Radio TX	ION + I_{RTC} + I_{X32k} + I_{X16M} + $I_{TX,0dBM}$+ I_{CRYPTO}
(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 27 Connection event

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

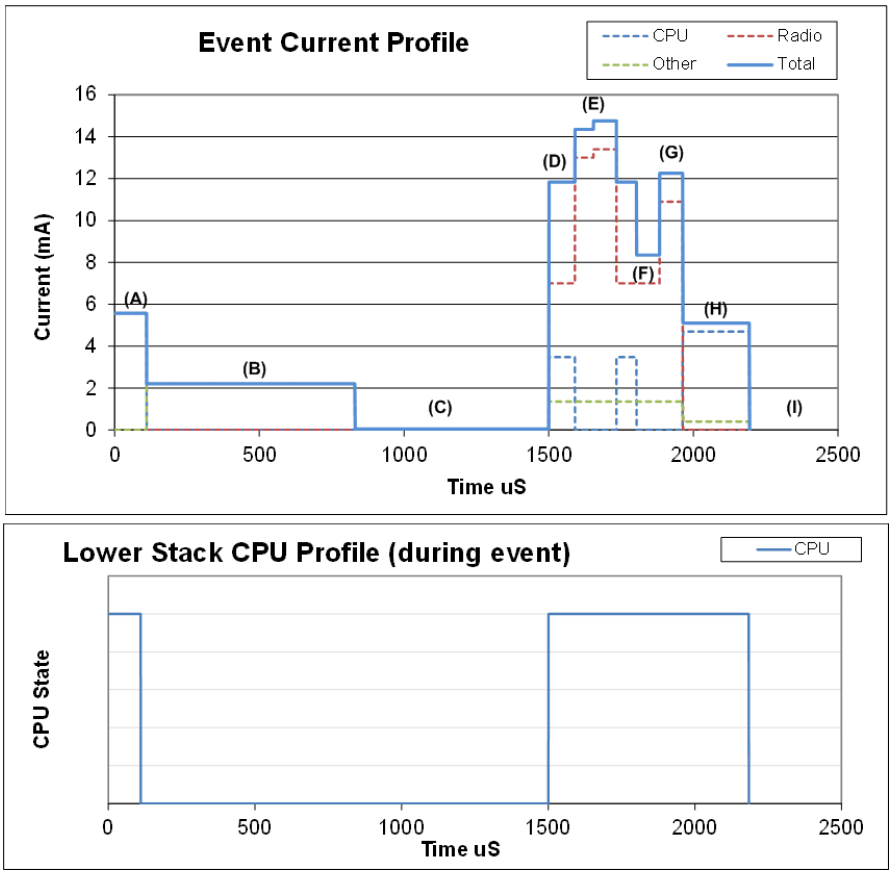

Figure 9 *Connection event*

11.2 Advertising event

Stage	Description	Current Calculation[1]
(A)	Pre-processing	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/switch	ION + I_{RTC} + I_{X32k} + I_{X16M} + ∫ ($I_{START,TX}$)
(E)	Radio TX	ION + I_{RTC} + I_{X32k} + I_{X16M} + $I_{TX,0dBM}$
(F)	Radio turn-around	ION + I_{RTC} + I_{X32k} + I_{X16M} + ∫ ($I_{START,RX}$)
(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	ION + I_{RTC} + I_{X32k}

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

Table 28 Advertising event

Note: IRC32k should be substituted for I_{X32k} when using the 32.768k R_{COSC}

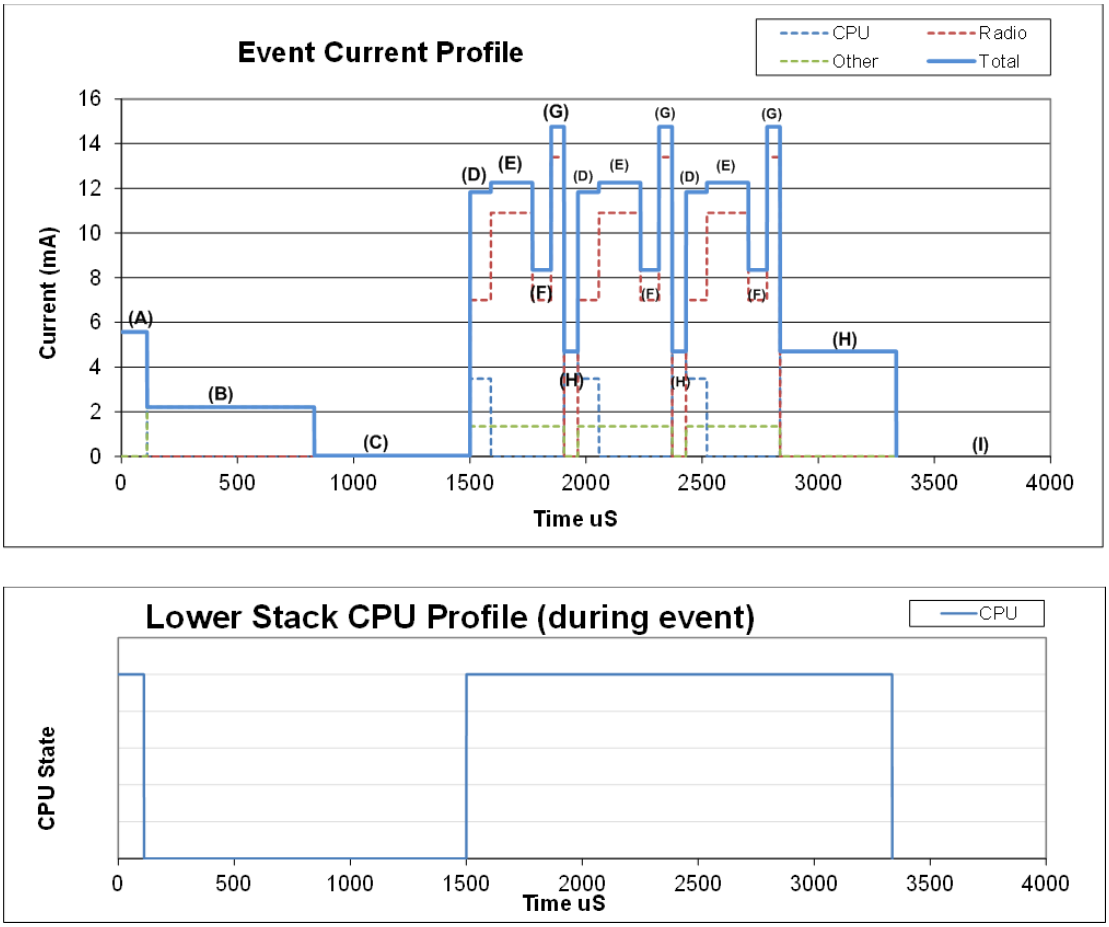

Figure 10 *Advertising event*

12 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 29***.

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 30***.

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 29 *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 30 *SoftDevice revision examples*

The test qualification levels are outlined in ***Table 31.***

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 31 *Test qualification levels*

12.1 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.