s310_nrf51422 migration document

Table of Contents

- Introduction to the s310_nrf51422 migration document
- S310_nrf51422_3.0.0
  - Required changes
    - Common
    - BLE specific
    - ANT specific
  - New Functionality
    - Common
    - BLE specific
    - ANT specific
  - Bootloaders
    - BLE specific
    - ANT specific
- S310_nrf51422_2.0.0 (updated for version 2.0.1)
  - Required changes
    - Common
    - BLE
    - ANT
  - New functionality
    - Common
    - BLE
    - ANT

Introduction to the s310_nrf51422 migration document

This document describes how to migrate to a new version of the s310_nrf51422 SoftDevice. The s310_nrf51422 release notes should be read in conjunction with this document.

For each version, we have the following sections:

- "Required changes" describes how an application would have used the previous version of the SoftDevice, and how it must now use this version for the given change.
- "New functionality" describes how to use new features and functionality offered by this version of the SoftDevice. **Note:** Not all new functionality may be covered; the release notes will contain a full list of new features and functionality.

Each section describes how to migrate to a given version from the previous version. If you are migrating to the current version from the previous version, follow the instructions in that section. To migrate between versions that are more than one version apart, follow the migration steps for all intermediate versions in consecutive order.

S310_nrf51422_3.0.0

This section describes how to migrate to s310_nrf51422_3.0.0 from s310_nrf51422_2.0.1

Required changes

Common

SoftDevice size

- The SoftDevice CODE size remains the same. This results in no change in application CODE starting address.
- The SoftDevice RAM size has changed resulting in the default application RAM starting address changing from 0x20002400 to 20002200. In this case, it is not required to change the application RAM starting address; however, due to application memory cost associated with enabling ANT channels (described in Default ANT stack channel configuration and availability has changed under Required changes), changing the application start address may be done to offset some of the impact in memory constrained designs.

	16kB RAM		32B RAM	
	Start Address	**Size**	**Start Address**	**Size**
RAM (IRAM)	0x20002200*	0x1E00	0x20002200*	0x5E00
Flash (IROM	0x1D000	0x23000	0x1D000	0x23000

* Assuming default GATT Server Attribute Table size (see below for details on configuring size)

SVC number changes

The SVC numbers used by the stack have been changed, therefore the application needs to be recompiled against the new header files. For systems using a bootloader to perform over-the-air device SoftDevice updates, refer to Bootloaders section.

The `NRF_POWER_DCDC_MODES` enumeration has been simplified

Instead of the previous `OFF`, `ON` and `AUTOMATIC` modes, it can now only be set to `NRF_POWER_DCDC_DISABLE` or `NRF_POWER_DCDC_ENABLE`. This affects the `sd_power_dcdc_mode_set()` SV call. Note that the DC/DC converter is only supported on nRF51 series IC revision 3.

BLE specific

The GATT Server Attribute Table size can now be configured

It is now possible to configure the size of the GATT Server Attribute table by using the parameter `attr_tab_size` in `ble_gatts_enable_params_t` when calling `sd_ble_enable()`.

Configuration of the Attribute Table size is optional. The default Attribute Table size (0x700 bytes) will be used if attr_tab_size is set to BLE_GATTS_ATTR_TAB_SIZE_DEFAULT. The default size is equal to the Attribute Table size in previous version of the S110 SoftDevice.

If the application wants to use an Attribute Table size different from the default one, the linker configuration of the application **must be changed** accordingly to match the total amount of RAM used by the SoftDevice. The following formula can be used to calculate the amount of RAM that the SoftDevice (together with the MBR) will use and therefore the address in RAM where the application memory area begins:

```
SoftDevice RAM size = (0x1B00 + attr_tab_size) default: 0x2200
```

The address at which the application memory area begins is therefore:

```
Application RAM begin = (0x20001B00 + attr_tab_size) default: 0x20002200
```

ble_gap_adv_params_t now contains an additional channel_mask field that must be filled in

When calling sd_ble_gap_adv_start(), a pointer to an instance of ble_gap_adv_params_t must be provided. This structure now contains an additional field (channel_mask) allowing the application to disable specific advertising channels, which must be filled in by the application. To enable all advertising channels simply set this whole field to zero. The SoftDevice behavior will then remain unchanged from previous versions.

The BLE_GAP_EVT_CONNECTED event now includes the device's own address

The new own_addr field in the ble_gap_evt_connected_t structure allows the application to find out which address was used to establish a particular connection, which can be useful when using privacy features.

The GAP options member name has changed

The ble_gap_opt_t instance inside ble_opt_t has been renamed from gap to gap_opt.

The GAP advertising timeout source macro has been renamed

The BLE_GAP_TIMEOUT_SRC_ADVERTISEMENT macro has been renamed to BLE_GAP_TIMEOUT_SRC_ADVERTISING.

The connection RSSI interface has been expanded

Two new parameters to the sd_ble_gap_rssi_start() SV call, threshold_dbm and skip_count, now allow the application to specify how often and under which conditions it wishes to receive connection RSSI events from the SoftDevice. Set these new parameters to 0 if you wish to receive the same number of events as with previous versions of the SoftDevice.

The new sd_ble_gap_rssi_get() SV call allows applications to poll the connection RSSI of the last connection interval.

The maximum value for slave latency has been reduced

The `BLE_GAP_CP_SLAVE_LATENCY_MAX` value has been changed from `0x03E8` to `0x01F3` to comply with versions 4.1 and above of the Bluetooth specification.

The security APIs have been redesigned and improved

All the application-facing security interfaces have been modified for improved speed, memory efficiency, future-proofing, and compatibility with other SoftDevice series. All application developers will have to adapt to the new security APIs.

Key distribution selection is now configurable by the application

Two instances of a new `ble_gap_sec_kdist_t` structure are now part of `ble_gap_sec_params_t` and allow the application to select which keys will be distributed by each side during a bonding procedure. Applications are therefore now required to carefully select which keys will be necessary during the bond's lifetime. The application should check the keys requested by the peer (propagated to the application inside the `ble_gap_sec_params_t` present in the `ble_gap_evt_sec_params_request_t` structure) before finally selecting the exchanged keys. The key selection needs to be included in the security parameters passed to `sd_ble_gap_sec_params_reply()`.

The recommended key distribution selection for applications is shown below. This will distribute the LTK and IRK in both directions when bonding and only when requested by the peer:

```
case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
if(bonding)
{
peer_params = p_ble_evt->evt.gap_evt.params.sec_params_request.peer_params;
   own_params.kdist_periph.enc = peer_params.kdist_periph.enc;
   own_params.kdist_periph.id = peer_params.kdist_periph.id;
   own_params.kdist_periph.sign = 0;
   own_params.kdist_central.enc = peer_params.kdist_central.enc;
   own_params.kdist_central.id = peer_params.kdist_central.id;
   own_params.kdist_central.sign = 0;
   sd_ble_gap_sec_params_reply(.., &own_params, ...);
}
break;
```

To emulate the behavior of previous SoftDevices, the key selection would be performed as follows:

```
case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
if(bonding)
{
peer_params = p_ble_evt->evt.gap_evt.params.sec_params_request.peer_params;
   own_params.kdist_periph.enc = peer_params.kdist_periph.enc;
   own_params.kdist_periph.id = peer_params.kdist_periph.id;
   own_params.kdist_periph.sign = 0;
   own_params.kdist_central.enc = 0;
   own_params.kdist_central.id = peer_params.kdist_central.id;
   own_params.kdist_central.sign = peer_params.kdist_central.sign;
   sd_ble_gap_sec_params_reply(.., &own_params, ...);
}
break;
```

The application can no longer specify a timeout for security procedures

The timeout field has been removed from ble_gap_sec_params_t, and the timeout length is now set internally by the stack according to the specification.

All encryption keys are now identified using EDIV/RAND pairs

The former LTK identifier, div, has now been removed from ble_gap_enc_info_t and instead LTKs are now identified using a ble_gap_master_id_t instance (EDIV/RAND pair). The application must now use the new master_id field in the ble_gap_evt_sec_info_request_t instead of the former div field to uniquely identify the LTK for that peer device.

Exchanged keys are now stored directly in application-provided memory

The application now supplies pointers to memory to store exchanged keys instead of receiving them in the BLE_GAP_EVT_AUTH_STATUS event. A number of changes are made to support this:

- Two new structures have been added to the API for this purpose, ble_gap_enc_key_t and ble_gap_id_key_t.
- The ble_gap_sec_keys_t has been refactored from a bitfield into a set of pointers to keys in application memory. p_enc_key, p_id_key and p_sign_key are all pointers that must be initialized by the application to point to its own instances of key structures in memory.
- A new structure, ble_gap_sec_keyset_t, has been introduced and is required as a parameter to sd_ble_gap_sec_params_reply(). This allows the application to provide the dual set of pointers to the stack (one for each of the devices participating in the bonding procedure). The keys are stored directly in the application's instances as soon as they are received over-the-air or generated and sent.

Please note that a special exception applies to the IRK distributed by the peripheral (ble_gap_sec_keyset_t::keys_periph::p_id_key). If this p_id_key pointer is initialized to a valid value, the IRK and device address pointed to will be distributed over the air by the SoftDevice to the peer device. If, however, the application sets p_id_key to NULL, the device's currently configured IRK and device address will be distributed. Most applications will want to set this pointer to NULL unless they plan to provide their own IRK and device address instead of using the one generated by the SoftDevice or configured using the BLE_GAP_OPT_PRIVACY option and the sd_ble_gap_address_set() SV call.

The authentication (pairing or bonding) procedure result structure has been revamped

The existing ble_gap_evt_auth_status_t structure, which encapsulates a BLE_GAP_EVT_AUTH_STATUS event, has been modified to fit the new key storage model. A new bonded field has been added that allows the application to check whether the procedure ended up in the generation of a new bond. A new structure, ble_gap_sec_kdist_t, which is simply a bitfield of the keys distributed during a bonding procedure, has been added to the API. Two instances of it have been included in the ble_gap_evt_auth_status_t structure as the kdist_periph and kdist_central fields. The application can check those new fields to find out which keys ended up being distributed during the bonding procedure after it has completed.

The security information reply now accepts IRKs:

The existing sd_ble_gap_sec_info_reply() SV call now takes an additional parameter in the form of p_id_info, a pointer to an optional IRK. This can be set to NULL by the application as it's currently ignored by the SoftDevice.

The GATTS set and get operations for local values have changed

The new function prototypes require a connection handle (since this is required for certain multi-value attributes) and also use a new structure `ble_gatts_value_t` for parameter input:

- `sd_ble_gatts_value_set(uint16_t conn_handle, uint16_t handle, ble_gatts_value_t *p_value)`
- `sd_ble_gatts_value_get(uint16_t conn_handle, uint16_t handle, ble_gatts_value_t *p_value)`

The GATTS set and get operations for system attributes have changed

The new function prototypes include an additional parameter, `flags`, to allow for partial retrieval or storage of system attributes. If the application does not want to make use of this new functionality, it can simply set the `flags` parameter to 0.

- `sd_ble_gatts_sys_attr_set(uint16_t conn_handle, uint8_t const *p_sys_attr_data, uint16_t len, uint32_t flags)`
- `sd_ble_gatts_sys_attr_get(uint16_t conn_handle, uint8_t *p_sys_attr_data, uint16_t *p_len, uint32_t flags)`

The GATTC write parameters have been modified

The field `flags` within the `ble_gattc_write_params_t` has been moved up in the structure for better alignment.

ANT specific

Default ANT stack channel configuration and availability has changed

In previous SoftDevices supporting ANT, the total number of channels supported has been statically defined and the required memory pre-allocated. This SoftDevice introduces the capability for applications to tailor and scale the following ANT stack options using an ANT Stack Enable Configuration API. Applications will need to configure the stack specifically.

ANT Stack Enable Configuration:

- Total number of ANT channels
- Number of encrypted channels
- Transmit burst queue size

The stack options are specified using the new API:

```
uint32_t sd_ant_enable(ANT_ENABLE * const pstChannelEnable)
```

- Returns NRF_SUCCESS if parameters were accepted; NRF_ERROR_INVALID_PARAM otherwise
- Call after enabling Softdevice, `sd_softdevice_enable`(), and before any ANT related functions
- Usage of this API is optional, except as noted below

Upon calling `sd_softdevice_enable`(), the ANT stack defaults to supporting 1 ANT channel (with encryption support) and a 64 byte transmit burst buffer. If advanced features (additional channels, encrypted channels, larger TX burst buffers) are needed by the application, then sd_ant_enable() must be used to specify the desired configuration. Application RAM memory must be supplied to the SoftDevice in order to increase the aforementioned stack options beyond the default configuration.

The ANT_ENABLE input structure consists of:

- **ucTotalNumberOfChannels** – total number of channels desired by the application (1 to 15)
- **ucNumberOfEncryptedChannels** – total number of encrypted ANT channels desired by the application (0 to ucTotalNumberOfChannels)
- **pucMemoryBlockStartLocation** – pointer to the RAM buffer location supplied by the application. Memory buffer is reserved for use by the SoftDevice in order to support specified ANT stack configuration.
- **usMemoryBlockByteSize** – size of provided memory buffer location by the application

The value of usMemoryBlockByteSize can be determined by using the provided macro definition in the ant_parameters.h header file.

- `#define ANT_ENABLE_GET_REQUIRED_SPACE (ucTotalNumberOfChannels, ucNumberOfEncryptedChannels, usTxQueueByteSize)`
    - Note: usTxQueueByteSize should be 64, 128 or 256 bytes.

Usage example:

To specify the same ANT stack options supported in previous SoftDevices (For example: S310 v2.0.1, S210 v4.0.1):

- 8 ANT channels
- 1 encrypted channel
- 128 byte TX burst buffer

```
#define ANT_NUM_TOTAL_CHANNELS         8
#define ANT_NUM_ENCRYPTED_CHANNELS     1
#define ANT_TX_BURST_QUEUE_SIZE        128

static ANT_ENABLE stANTEnableParams;
static uint8_t aucANTEnableMem[ANT_ENABLE_GET_REQUIRED_SPACE(ANT_NUM_TOTAL_CHANNELS,
ANT_NUM_ENCRYPTED_CHANNELS, ANT_TX_BURST_QUEUE_SIZE)];

// configure ANT stack options
stANTEnableParams.ucTotalNumberOfChannels = ANT_NUM_TOTAL_CHANNELS;
stANTEnableParams.ucNumberOfEncryptedChannels = ANT_NUM_ENCRYPTED_CHANNELS;
stANTEnableParams.pucMemoryBlockStartLocation = aucANTEnableMem;
stANTEnableParams.usMemoryBlockByteSize =
ANT_ENABLE_GET_REQUIRED_SPACE(ANT_NUM_TOTAL_CHANNELS, ANT_NUM_ENCRYPTED_CHANNELS,
ANT_TX_BURST_QUEUE_SIZE);

// enable softdevice
```

```
ulErrorCode = sd_softdevice_enable(NRF_CLOCK_LFCLKSRC_XTAL_50_PPM,
softdevice_assert_callback);

// enable ANT stack options
ulErrorCode = sd_ant_enable(&stANTEnableParams);

// ... configure and use ANT channels ...etc
```

New Functionality

Common

The SoftDevice info structure is now documented

A set of new macros has been introduced to access the SoftDevice info structure directly from hex or bin SoftDevice images. This can be useful when doing device firmware upgrades or when generic information about a SoftDevice in binary form is to be retrieved. See `nrf_sdm.h` for details on the new macros listed below.

- `MBR_SIZE`
- `SOFTDEVICE_INFO_STRUCT_OFFSET`
- `SOFTDEVICE_INFO_STRUCT_ADDRESS`
- `SOFTDEVICE_INFO_STRUCT_OFFSET`
- `SD_FWID_OFFSET`
- `SD_SIZE_GET()`
- `SD_FWID_GET()`

BLE specific

Scan Request reports are now available

An application can now request to receive events whenever an observer issues a scan request by using the new `BLE_GAP_OPT_SCAN_REQ_REPORT` option. The scan request reports will be propagated to the application through the new `ble_gap_evt_scan_req_report_t`, and more information on the feature can be found under the documentation for the new option structure `ble_gap_opt_scan_req_report_t`.

The connection channel map is now retrievable

Applications can now retrieve the channel map used for connections by using the new `BLE_GAP_OPT_CH_MAP` option. This feature is completely independent of the advertising channel mask listed in the "Required changes" section of this document. See documentation of `ble_gap_opt_ch_map_t` for more information.

ANT specific

Increased total number of channels supported

The maximum number of channels supported by the ANT stack has been increased from 8 to 15. The default number of channels supported by the ANT stack upon enabling the SoftDevice is 1. Each additional channel requires SIZE_OF_NON_ENCRYPTED_ANT_CHANNEL bytes of memory as defined in ant_parameters.h.

For details on how to enable additional channels, refer to Default ANT stack channel configuration and availability has changed under Required changes section.

Increased number of channels supporting encryption

The maximum number of encrypted channels supported by the ANT stack has been increased from 1 to 15. The number of encrypted channels cannot exceed the total number of ANT channels configured. The default number of encrypted channels supported by the ANT stack upon enabling the SoftDevice is 1. Each additional encryption channel supported requires SIZE_OF_ENCRYPTED_ANT_ANT_CHANNEL bytes of memory as defined in ant_parameters.h. This is an additional memory cost on top of SIZE_OF_NON_ENCRYPTED_ANT_CHANNEL bytes required to support an ANT channel.

For details on how to enable additional encrypted channels, refer to Default ANT stack channel configuration and availability has changed under Required changes section.

Increased supported number of encryption keys

The number of encryption keys supported now increase proportionally to the number of encrypted channels configured. The following APIs are no longer bound to 1 key and may use a key index (ucKeyNum) that is bounded between [0 to (numEncryptedChannels – 1)], where numEncryptedChannels > 1.

- `sd_ant_crypto_channel_enable()`
- `sd_ant_crypto_key_set()`

Encryption channel pool

An available encryption channel pool is created for the number of encryption channels desired by the application. It represents the total number of channels allowed to run concurrently with encryption enabled. Any ANT channel may enable encryption as long as the available encryption channel pool is not fully used. Encryption channel pool usage example is shown below:

```
//.. ANT stack configured for total 8 channels, 2 encrypted channels

// total available encryption channel pool = 2
ulErrorCode = sd_ant_crypto_channel_enable(0, 1, 0, 1); // channel 0 encrypt enable
if (ulErrorCode == NRF_SUCCESS)
{
   // Successful API call, available encryption pool reduced by 1
   // total available encryption channel pool = 1
}

ulErrorCode = sd_ant_crypto_channel_enable(7, 1, 7, 1); // channel 7 encrypt enable
if (ulErrorCode == NRF_SUCCESS)
{
```

```
   // Successful API call, available encryption pool reduced by 1
   // total available encryption channel pool = 0
}

// ...

ulErrorCode = sd_ant_crypto_channel_enable(0, 0, 3, 1); // channel 0 encrypt disable
if (ulErrorCode == NRF_SUCCESS)
{
   // Successful API call, available encryption pool increased by 1
   // total available encryption channel pool = 1
}
```

Note: Successful or failed encryption negotiations reported by ANT events: EVENT_ENCRYPT_NEGOTIATION_SUCCESS and EVENT_ENCRYPT_NEGOTIATION_FAIL have no effect on encrypted channel pool status. Encrypted channel pool assignment/un-assignment are handled strictly by the sd_ant_crypto_channel_enable() API.

Transmit burst queue size configurability

The transmit burst queue is used by the ANT stack in order for buffer packets to be sent during burst transfers. The size of the burst queue does not represent the maximum burst transfer size. Its use is to help manage and source data packets from the application to the over-the-air ANT transfer protocol in a timely manner and help offset application processing latency. Calls made to sd_ant_burst_handler_request() API result in filling this buffer.

In previous SoftDevices, the transmit burst queue size was fixed at 128 bytes. With the introduction of the ANT stack enable configuration interface, the transmit burst queue size can be adjusted via the sd_ant_enable() API. The default queue size upon enabling the SoftDevice is 64 bytes.

Size of the transmit burst queue cannot be less than 64 bytes and no greater than 256 bytes and must be a value that is a power of 2. For details on how to allocate transmit burst queue size, refer to Default ANT stack channel configuration and availability has changed under Required changes section.

Bootloaders

BLE specific

There are no BLE specific bootloader changes since the 2.0.0 release.

ANT specific

For systems using a bootloader to perform over-the-air firmware updates using ANT, users must first verify the compatibility of the existing bootloader with this SoftDevice. Bootloaders built with the previous SoftDevice versions (eg. S310 v2.0.1, S210 v4.01) will not be compatible with this SoftDevice due to changes in the SVC numbering and ordering.

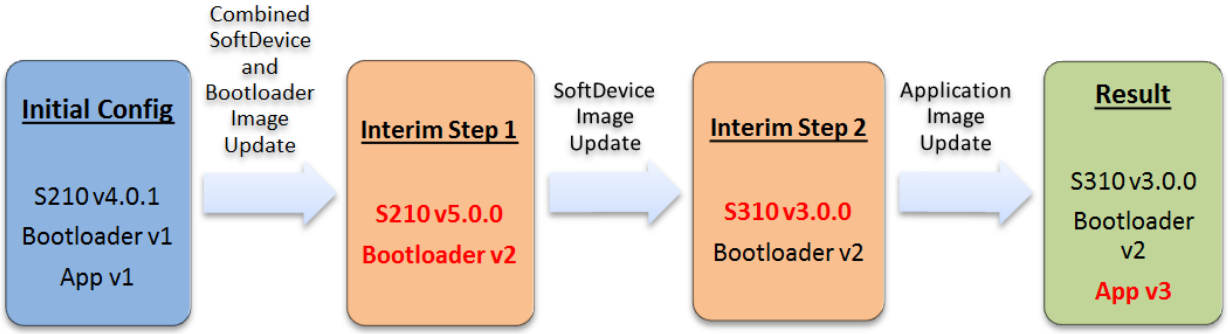
If not compatible, bootloaders must recompiled with the updated SoftDevice headers (along with any other potential changes reflecting bootloader compatibility and/or identification such as version strings).

When performing over-the-air updates, both bootloader and SoftDevice must be upgraded at the same time using a combined image.

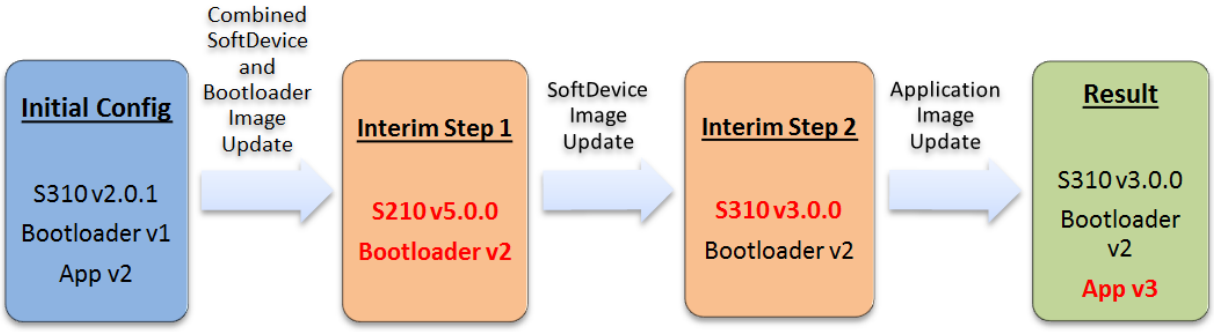
Since the flash size on the nRF51 restricts combined SoftDevice S310 + Bootloader updates, a temporary swap to the latest compatible S210 SoftDevice **must** take place first, before updating to the new S310 SoftDevice.

The following depicts the supported upgrade paths to the new S310 SoftDevice when performing over-the-air firmware updates using an ANT bootloader (eg. ANT-WP bootloader).

Updating from S210 v4.0.1 to S310 v3.0.0



Updating from S310 v2.0.1 to S310 v3.0.0



Notes:

- Bootloader v1 compatible for use with S210 v4.0.1 and S310 v2.0.1
- Bootloader v2 compatible for use with S210 v5.0.0 and S310 v3.0.0
- Application v1 compatible with S210 V4.0.1
- Application v2 compatible with S310 v2.0.1
- Application v3 compatible with S310 v3.0.0

S310_nrf51422_2.0.0 (updated for version 2.0.1)

This section describes how to migrate to s310_nrf51422_2.0.0 from s310_nrf51422_1.0.0.

The S310 SoftDevice HEX file in version 2.0.1 is identical to version 2.0.0. This document has been updated for version 2.0.1 to include additional information on the sd_ant_cw_test_mode() function.

Required changes

Common

SoftDevice size

The reserved code size of the SoftDevice has been changed to free up more code space for the application. The application scatter file or the Keil target settings must be updated to the following values:

	16 kB RAM		32 kB RAM	
	Start address	**Size**	**Start address**	**Size**
RAM (IRAM)	0x20002400	0x1C00	0x20002400	0x5C00
Flash (IROM)	0x1D000	0x23000	0x1D000	0x23000

However, if the SoftDevice is disabled, the RAM available to the application can start at 0x20000008 instead of 0x20000004. The additional four bytes have been added to support interrupt forwarding performed by the Master Boot Record (MBR).

	16 kB RAM		32 kB RAM	
	Start address	**Size**	**Start address**	**Size**
RAM (IRAM)	0x20000008	0x3FF8	0x20000008	0x7FF8
Flash (IROM)	0x1D000	0x23000	0x1D000	0x23000

Note: The available RAM start address with the SoftDevice disabled has increased by 4 bytes (from 0x20000004 to 0x20000008) in this version of the SoftDevice.

SVC numbers

The SVC numbers in use by the stack have been changed. Applications must recompile against new header files to work with the new SoftDevice.

Redirecting interrupts to an application from a bootloader

The function `sd_softdevice_forward_to_application()` has been replaced with `sd_softdevice_vector_table_base_set()`. Interrupts can now be directed to anywhere in the application flash area.

Radio Disable API replaced by Concurrent Multiprotocol Timeslot API

The functionality of the previous Radio Disable API, which allowed the application to schedule timeslots of radio inactivity, is now a part of the new Concurrent Multiprotocol API feature set. This involves the following changes:

- The `nrf_radio_disable.h` header has been removed. Definitions are now consolidated into `nrf_soc.h`.
- The `nrf_radio_request_t` parameter type used in `sd_radio_request()` has been changed:
  - Structure of `nrf_radio_request_t` has changed to support two request types as defined by request_type field: `nrf_radio_request_normal_t` and `nrf_radio_request_earliest_t`.
  - To schedule a radio event as soon as possible, use `nrf_radio_request_earliest_t` with `timeout_us = 100000L`. Previously this was achieved by using a normal `nrf_radio_request_t` and setting `distance_us = 0`.
  - The member `hfclk` replaces `nrf_radio_request_reserved1` and should be set to `NRF_RADIO_HFCLK_CFG_DEFAULT`.
- The `nrf_radio_signal_callback_return_param_t` type has changed:
  - `return_code` field has been renamed to `callback_action`.

Existing APIs from the Radio Disable API have been moved to the Concurrent Multiprotocol Timeslot API:

- `sd_radio_session_open()`
- `sd_radio_session_close()`
- `sd_radio_request()`

Existing SoC events from the Radio Disable API have been moved to the Concurrent Multiprotocol Timeslot API:

- `NRF_EVT_RADIO_BLOCKED`
- `NRF_EVT_RADIO_CANCELED`
- `NRF_EVT_RADIO_SIGNAL_CALLBACK_INVALID_RETURN`
- `NRF_EVT_RADIO_SESSION_IDLE`
- `NRF_EVT_RADIO_SESSION_CLOSED`

Refer to the SoftDevice API documentation for more details.

BLE

sd_ble_enable()

An additional call has to be made after `sd_softdevice_enable()` before any BLE related functionality in the SoftDevice can be used:

- `sd_ble_enable(p_ble_enable_params)`

Using this call, the application can select whether to include the Service Changed characteristic in the GATT Server. The default in all previous releases has been to include the Service Changed

characteristic, but this affects how GATT clients behave. Specifically, it requires clients to subscribe to this attribute and not to cache attribute handles between connections unless the devices are bonded. If the application does not need to change the structure of the GATT server attributes at runtime, this adds unnecessary complexity to the interaction with peer clients. If the SoftDevice is enabled with the Service Changed Characteristics turned off, then clients are allowed to cache attribute handles making applications simpler on both sides. See the SoftDevice API documentation for details.

Due to an issue present in this release, the application is required to set the device address to the factory default right after calling `sd_ble_enable()`, this can be achieved with the following lines:

```
sd_ble_enable(p_ble_enable_params);
sd_ble_gap_address_get(&addr);
sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_NONE, &addr);
```

sd_ble_gap_address_set()

`sd_ble_gap_address_set()` now takes an additional argument which is used to describe the private address cycle mode, `addr_cycle_mode`. The new prototype is:

- `uint32_t sd_ble_gap_address_set(uint8_t addr_cycle_mode, ble_gap_addr_t const * const p_addr)`

The `addr_cycle_mode` argument can be one of:

- `BLE_GAP_ADDR_CYCLE_MODE_NONE`
- `BLE_GAP_ADDR_CYCLE_MODE_AUTO`

To set all types of device addresses explicitly as with earlier versions of the SoftDevice, use

```
sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_NONE, p_addr)
```

To let the SoftDevice automatically cycle private addresses as defined by Bluetooth Core specification 4.1, use

```
sd_ble_gap_address_set(BLE_GAP_ADDR_CYCLE_MODE_AUTO, p_addr)
```

where `p_addr->addr_type` is
either `BLE_GAP_ADDR_TYPE_RANDOM_PRIVATE_RESOLVABLE` or `BLE_GAP_ADDR_TYPE_RANDOM_PRIVATE_NON_RESOLVABLE`.

sd_ble_gap_authenticate()

`The function sd_ble_gap_authenticate()` can now return an additional error code `NRF_ERROR_TIMEOUT` to indicate an SMP timeout.

ANT

sd_ant_prox_search_set()

The `sd_ant_prox_search_set()` function now takes an additional argument: `ucCustomProxThreshold`. The new prototype is:

- `uint32_t sd_ant_prox_search_set(uint8_t ucChannel, uint8_t ucProxThreshold, uint8_t ucCustomProxThreshold)`

The `ucCustomProxThreshold` parameter allows applications to specify a custom minimum RSSI threshold value instead of using predefined ANT indexed values in `ucProxThreshold`. The custom value only applies if the `PROXIMITY_THRESHOLD_CUSTOM` bit is set in `ucProxThreshold`. Set `ucCustomProxThreshold` to 0 if unused.

sd_ant_cw_test_mode()

The `sd_ant_cw_test_mode()` function now takes an additional argument: `ucMode`. The old and new prototype is seen below.

- `uint32_t sd_ant_cw_test_mode(uint8_t ucRadioReq, uint8_t ucTxPower, uint8_t CustomTxPower)`
- `uint32_sd_ant_cw_test_mode(uint8_t ucRadioReq, uint8_t ucTxPower, uint8_t CustomTxPower, uint8_t ucMode)`

The `ucMode` parameter allows the application to specify the ANT CW test mode where a value of 0 denotes TX carrier transmission test mode (original test mode) and a value of 1 denotes TX continuous modulated transmission test mode (new test mode).

New functionality

Common

SoftDevice size removed from UICR.CLENR0

The SoftDevice HEX file no longer contains the SoftDevice size in the `UICR.CLENR0` register. This means that the Memory Protection Unit is no longer configured to protect the SoftDevice code, memory space and protected peripherals, unless this is deliberately enabled. Memory protection must be disabled to allow Device Firmware Upgrade of the SoftDevice. However, it may be useful to have the protection enabled during development to ease the detection of illegal memory and peripheral accesses.

For device programming with nRFgo Studio 1.17 or newer, in the **Program SoftDevice** tab, check **Enable SoftDevice protection** to enable or disable the protection.

For device programming with nrfjprog version 5.1.1 or newer, append the `--dfu` command line option with `--programs` to disable memory protection. For example:

```
nrfjprog --programs softdevice.hex --dfu # This will disable memory protection and program SoftDevice.
nrfjprog --programs softdevice.hex       # This will enable memory protection and program SoftDevice.
```

Concurrent Multiprotocol Timeslot API

A new Concurrent Multiprotocol Timeslot API has been introduced, replacing the Radio Disable API. This enables an application to schedule timeslots in which the SoftDevice releases the `RADIO` and `TIMER0` hardware peripherals to the application. This feature can be used to implement a separate radio protocol

in application space that can run concurrently with a SoftDevice protocol, or to schedule timeslots where the SoftDevice is guaranteed to be idle to improve latency or reduce peak power consumption.

The following is a list of additions to the Concurrent Multiprotocol Timeslot API, not present in the previous Radio Disable API.

- Additional `p_radio_signal_callback_types` have been added:
  - `NRF_RADIO_CALLBACK_SIGNAL_TYPE_TIMER0` - generated whenever `NRF_TIMER0` interrupts occur.
  - `NRF_RADIO_CALLBACK_SIGNAL_TYPE_RADIO` - generated whenever `NRF_RADIO` interrupts occur.
  - `NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_FAILED` - generated whenever session extension has failed.
  - `NRF_RADIO_CALLBACK_SIGNAL_TYPE_EXTEND_SUCCEEDED` - generated whenever session extension has succeeded.
- Additional return types for `nrf_radio_signal_callback_return_param_t` have been added:
  - `NRF_RADIO_SIGNAL_CALLBACK_ACTION_EXTEND` - used to request an extension to the current timeslot. Timeslot extension parameters must be specified in `extend` struct.
  - `NRF_RADIO_SIGNAL_CALLBACK_ACTION_REQUEST_AND_END` - used to request a new radio timeslot and end the current timeslot. New radio timeslot request parameters must be specified in request struct.

See the SoftDevice Specification document and the SoftDevice API documentation for details and guidelines on how to use the new features.

New LFCLK oscillator sources

The following oscillator clock sources have been added for the `LFCLK`:

- `NRF_CLOCK_LFCLKSRC_RC_250_PPM_TEMP_1000MS_CALIBRATION`
- `NRF_CLOCK_LFCLKSRC_RC_250_PPM_TEMP_2000MS_CALIBRATION`
- `NRF_CLOCK_LFCLKSRC_RC_250_PPM_TEMP_4000MS_CALIBRATION`
- `NRF_CLOCK_LFCLKSRC_RC_250_PPM_TEMP_8000MS_CALIBRATION`
- `NRF_CLOCK_LFCLKSRC_RC_250_PPM_TEMP_16000MS_CALIBRATION`

The new clock source types use the on-chip RC oscillator to generate a 250 PPM clock signal. Additional power saving is achieved by performing calibration at the specified interval only if the temperature has changed.

Master Boot Record (MBR) API

An MBR API is introduced with the SoftDevice that allows for switching between bootloader(s) and application(s). In addition, the API can be used to replace the bootloader and SoftDevice when performing Device Firmware Updates. The MBR exposes one function:

- `sd_mbr_command(command)`

The following command types are supported:

- `SD_MBR_COMMAND_COPY_BL` – used to copy a new bootloader into place.

- `SD_MBR_COMMAND_COPY_SD` – used to copy a new SoftDevice into place.
- `SD_MBR_COMMAND_INIT_SD` – used to initialize SoftDevice from bootloader and enable interrupt forwarding to it.
- `SD_MBR_COMMAND_COMPARE` – used to compare flash memory blocks.
- `SD_MBR_COMMAND_VECTOR_TABLE_BASE_SET` - used to set the address that the MBR will forward interrupts to.

BLE

Options API

The SoftDevice now includes a new Options API to allow the application to set and get advanced configuration options. The API exposes two functions:

- `sd_ble_opt_get()`
- `sd_ble_opt_set()`

The following options are defined in this version of the SoftDevice:

- `BLE_COMMON_OPT_RADIO_CPU_MUTEX` can be used to configure application access to the CPU while the radio is active.
- `BLE_GAP_OPT_LOCAL_CONN_LATENCY` can be used to override the connection latency specified by the central.
- `BLE_GAP_OPT_PASSKEY` can be used to specify a 6-digit display passkey that will be used during pairing instead of a randomly generated one.
- `BLE_GAP_OPT_PRIVACY` can be used to tune the behavior of the SoftDevice when advertising with private addresses.

See the SoftDevice API documentation for details on how to use the Options API.

New advertising data types

The SoftDevice now has built-in support for more advertising data types. See `ble_gap.h` for the full list of supported advertising data types.

ANT

RSSI proximity now supported in Continous Scanning Mode

Specifying ANT proximity settings or custom RSSI values using the sd_ant_prox_search_set() API will now apply to a channel opened with `sd_ant_rx_scan_mode_start()`. Received packets that do not meet the specified minimum RSSI threshold will not be sent to the application.

Transmission from continuous scanning device now supports wild card ID parameters

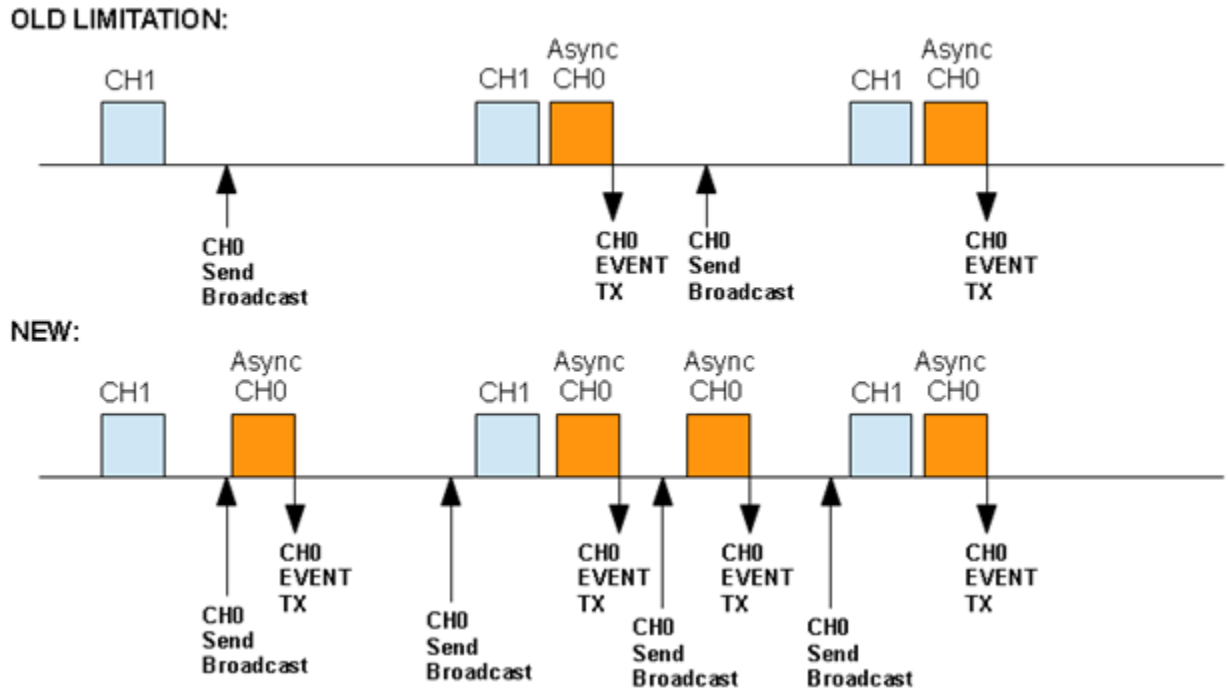
In order to allow continuous scanning devices to respond to incoming messages that matches one or more of the scanning channel ID fields, wildcard parameters (“0” value field in Device Number, Device Type and/or Transaction Type) can be specified when configuring the ID of the channel used for generating the response.

By using wildcard ID fields in this manner, the overall response latency can be reduced for situations where the continuous scanning device needs to reply immediately to incoming messages sent by multiple devices with different IDs.

Asynchronous transmit channel events in the presence of other channels

In S310 v1.0.0, when sending a transmission via a channel configured with the `EXT_PARAM_ASYNC_TX_MODE` extended channel type in the presence of other running ANT channels, the transmission would not occur until after the next scheduled ANT activity has run.

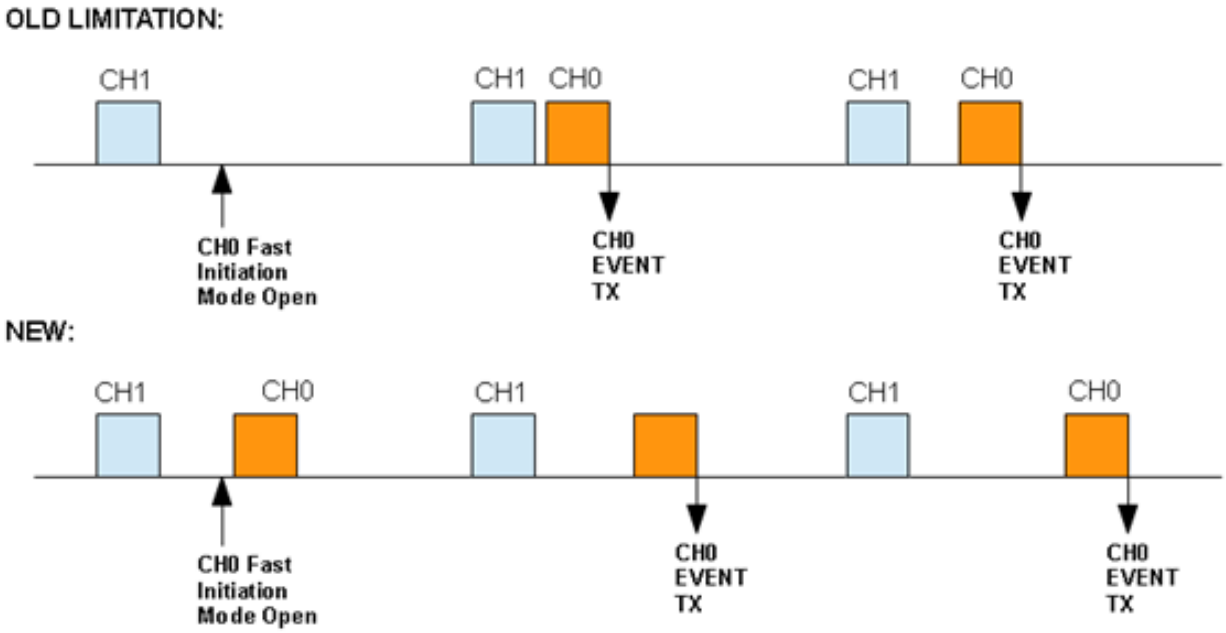
This limitation has now been removed. Asynchronous transmissions, in the presence of other running channels, are now performed immediately unless it is blocked by a previously scheduled radio activity. See the figure below.



Fast initiation channel in the presence of other channels

In S310 v1.0.0, when opening a channel configured with the `EXT_PARAM_FAST_INITATION_MODE` extended channel type in the presence of other running ANT channels, the channel would not start until after the next scheduled ANT activity has run.

This limitation has now been removed. Channels configured with fast channel initiation will now start immediately, unless it is blocked by a previously scheduled radio activity. See the figure below.



Improved continuous scanning channel operation during timeslot activity

ANT continuous scanning channel behavior has been optimized to use more of the available free time around concurrent application timeslot and flash scheduling activity.