s210_nrf51422 migration document

Table of Contents

Introduction to the s210_nrf51422 migration document

This document describes how to migrate to a new version of the s210_nrf51422 SoftDevice. The s210_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 order.

S210_nrf51422_5.0.0

This section describes how to migrate to s210_nrf51422_5.0.0 from s210_nrf51422_4.0.1

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 remains the same. This results in no change in the application RAM starting address.

SVC number changes:

The SVC numbers in use by the SoftDevice have been changed so application(s) and bootloader(s) need 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.

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()`

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 Configurable ANT stack channel configuration 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 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.

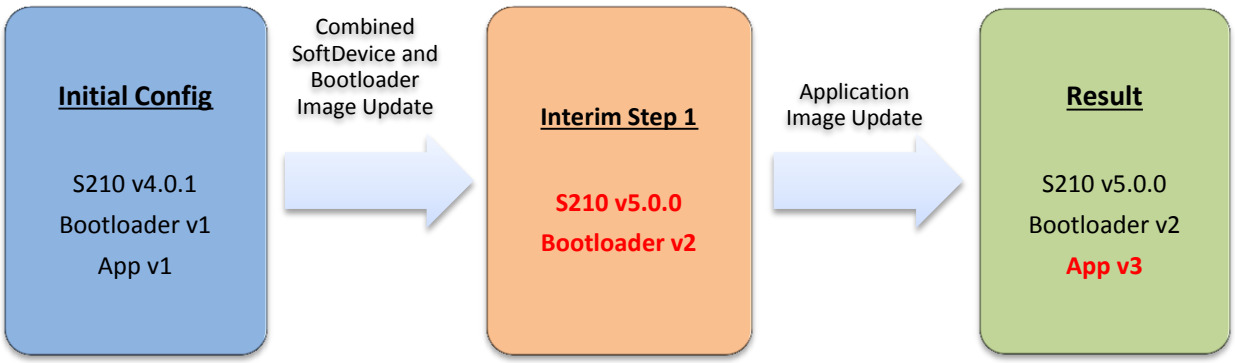
ANT specific

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.

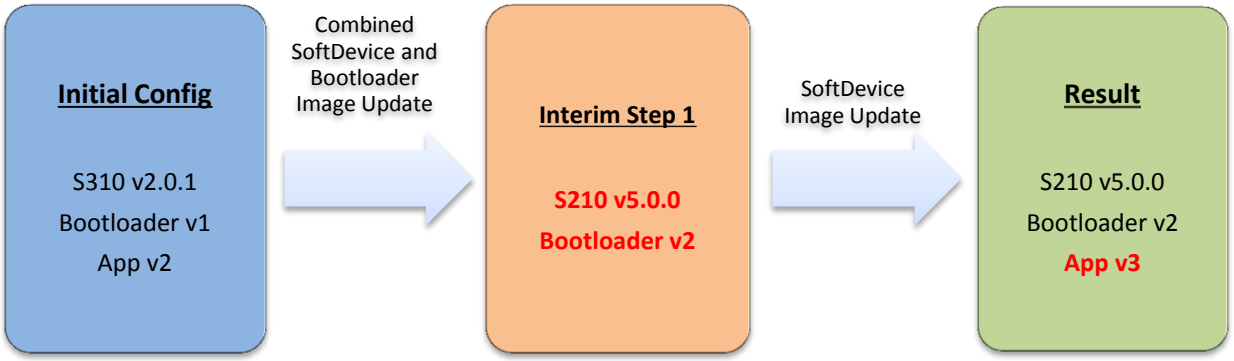
If not compatible, bootloaders must be recompiled with the updated SoftDevice headers (along with any other potential changes reflecting bootloader compatibility and/or identification such as version strings). When performing over-the-air updates, both bootloader and SoftDevice must be upgraded at the same time using a combined image.

The following depicts the supported upgrade paths to the new S210 SoftDevice when performing over-the-air firmware updates using an ANT bootloader (eg. ANT-WP bootloader).

Updating from S210 v4.0.1 to S210 v5.0.0



Updating from S310 v2.0.1 to S210 v5.0.0



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 S210 v5.0.0

S210_nrf51422_4.0.0

This section describes how to migrate to s210_nrf51422_4.0.0 from s210_nrf51422_3.0.0

Required changes

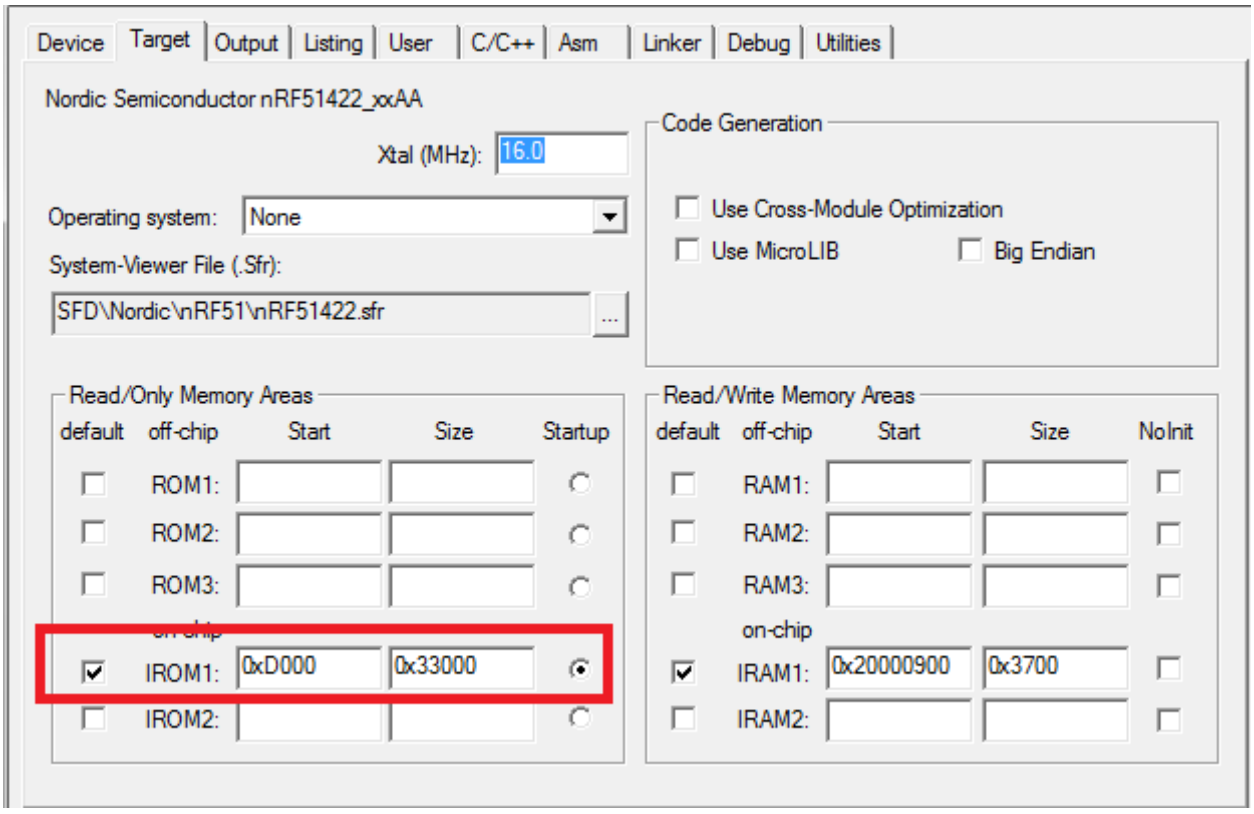
SoftDevice size

The size of the SoftDevice has changed to accommodate the inclusion of the Master Boot Record (MBR). The required application project file changes are as follows:

Example Keil project properties under 'Target' tab:

- Change IROM1 Start to 0xD000
- Ensure that IROM1 size is no more than 0x33000

If the project uses a scatter file instead of the settings from the Target tab, the scatter file must be updated accordingly.



Disabled SoftDevice reserved RAM requirement changes:

The RAM requirement when the SoftDevice is disabled has been increased in order to support interrupt forwarding to be done by the MBR.

- Applications wishing to use the available RAM when the SoftDevice is disabled must not overwrite the first 8 bytes of RAM.

	s210_nrf51422_3.0.0 SoftDevice Disabled RAM requirements	s210_nrf51422_4.0.0 SoftDevice Disabled RAM requirements
Size	4 bytes	8 bytes
Application useable RAM start address	0x2000 0004	0x2000 0008

SVC number changes:

A limited set of SVC numbers have changed. The application is required to be re-compiled against the new headers.

`sd_ant_prox_search_set()` now takes an additional argument:

- `uint8_t ucCustomProxThreshold` parameter allows applications to specify a custom minimum RSSI threshold value instead of using predefined ANT indexed values in `uint8_t ucProxThreshold`. The custom value is only applied if the `uint8_t ucProxThreshold` is set with the `PROXIMITY_THRESHOLD_CUSTOM` bit. If the custom proximity field is not used, set it to 0.

Redirecting interrupts to an application from a bootloader has changed:

- `sd_softdevice_forward_to_application()` has been replaced with `sd_softdevice_vector_table_base_set(address)`.

Interrupts can now be directed to anywhere in the application flash area. This also enables using more than one application. See the SoftDevice API documentation for details on how to use this call.

The Radio Disable API functionality has been replaced by the 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 Timeslot API feature set.

- `nrf_radio_disable.h` header file removed. Definitions consolidated into `nrf_soc.h`.
- `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`.
    - Use `nrf_radio_request_earliest_t` and set `timeout_us = 100000L`instead of using `distance_us` = 0 in a normal request type).
    - 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` return parameter type has changed.

- `return_code` field has been renamed to `callback_action`.

Refer to the SoftDevice API documentation for more details.

New functionality

The SoftDevice hex file no longer contains the SoftDevice size in UICR.CLENR0 register:

The SoftDevice region size is no longer specified in the UICR.CLENR0 in order to allow full flexibility in SoftDevice size changes for Device Firmware Updates. However, memory protection can be enabled during development to detect illegal memory/peripheral accesses.

- If programming the SoftDevice using nRFgo Studio 1.17 or newer, use the "Enable SoftDevice protection" checkbox in "Program SoftDevice" dialog to enable/disable protection.
- If programming using nrfjprog.exe version 5.1.1 or newer, using the `--programs` option will enable protection. Specifying `--programs --dfu` will disable protection.

Multiprotocol Timeslot API features not previously available in Radio Disable API:

The Concurrent Multi-Protocol Timeslot API implementation enables the application to schedule timeslots. During a timeslot the SoftDevice gives control over 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 the SoftDevice protocol, or to schedule timeslots where the SoftDevice is guaranteed to be idle, for example to improve latency for the application, or to reduce peak power consumption.

From the previous Radio Disable API feature, the Multiprotocol Timeslot API introduces more flexible scheduling options that allow applications to:

- Perform radio and timer0 operations during session callback.
- Tail chain radio session requests from a previous session callback.
- Manage ongoing application session activities through the use of session extension requests.

Existing APIs from Radio Disable feature transferred to Concurrent Timeslot API:

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

Existing SoC Events transferred to Concurrent 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`

The following is a list of additions to the Multiprotocol Timeslot API from 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.

New LFCLK oscillator sources are available:

The following source types have been introduced:

- `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 signal. Additional power saving is achieved by performing calibration at the specified interval only if the temperature has changed.

Master Boot Record (MBR) API included with SoftDevice:

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.

- `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 to which the MRB will forward interrupts.

ANT RSSI proximity can now be configured and used in ANT RX scanning channel:

- Specifying ANT proximity settings or custom RSSI values using the `sd_ant_prox_search_set()` API will apply to the ANT RX scanning channel.

When running an ANT RX scanning channel, received packets that do not meet the specified minimum RSSI threshold will not be sent to the application.

Wildcard IDs can now be used when sending uplink transmission from an ANT RX scanning channel:

Previously, the ID of the target device must be known before uplink transmission from an ANT RX scanning channel can be sent to it. When implementing a system where the scanning channel has to receive and reply to multiple devices with different IDs, having to read and set the received ID prior to sending the reply transmission would result in delayed reply occurring at the next instance of the matching ID received packet.

In order to reduce the reply latency, the application can now assign wildcard ("0") channel IDs to an ANT transmission channel and by pending transmission on that channel, the RX scanning channel will be able to reply back in the same instance of a received packet. Please note that the transmission channel ID will be assigned to the matching received ID once this has occurred; therefore the ID must be changed back to wildcard if the application is intending to reply back to multiple IDs.

Example:

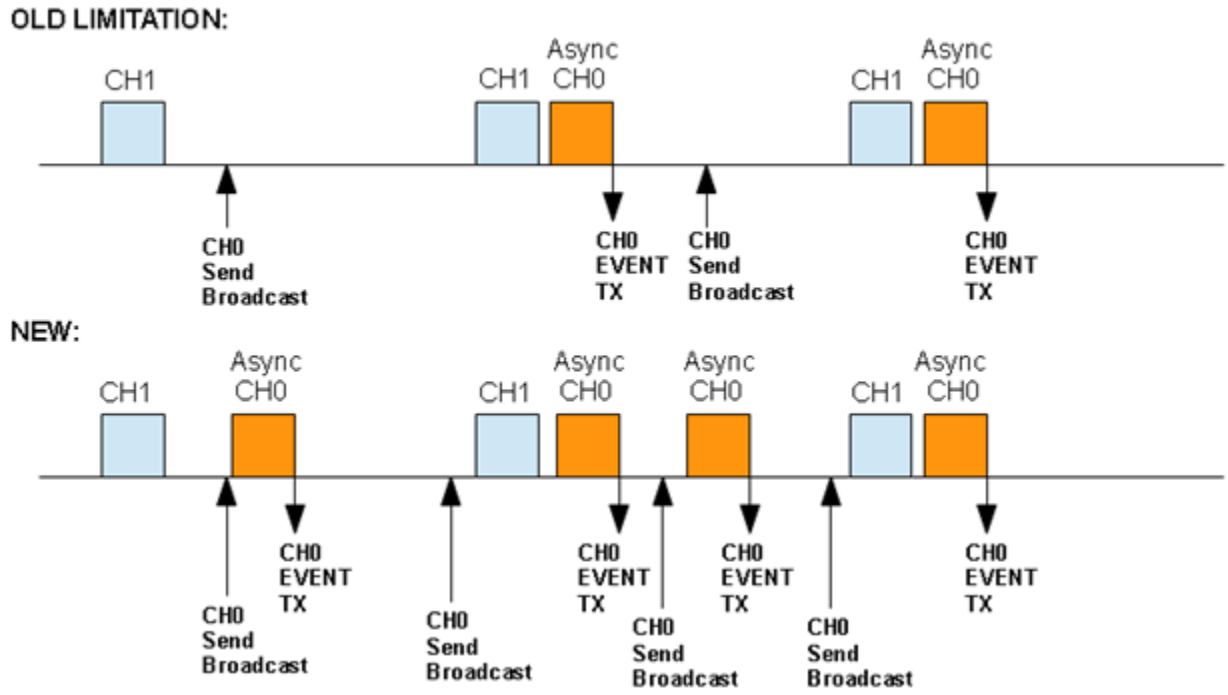
- …Prior channel configuration setup (e.g. RF frequency…etc.).
- `sd_ant_channel_id_set (0, 0, 1, 1)`
  - Channel 0 as RX scanning channel.
  - Wildcard device ID for RX scanning channel in order to listen for all device IDs.
- `sd_ant_channel_id_set (1, 0, 1, 1)`
  - Channel 1 used as the RX scanning transmission channel.
  - Wildcard device ID to allow transmission to any device ID packet received from scan.
- `sd_ant_broadcast_message_tx(1,8, buf)`
  - Set pending broadcast transmission on RX scanning transmission channel.
- `sd_ant_rx_scan_mode_start(0)`
  - Open channel RX scanning channel.
  - 0 parameter assigned to not listen for only synchronous RX packets.
- EVENT_RX received from channel 0 in application code.
  - Indicating data received from scanning channel.
- EVENT_TX received from channel 1 in application code.
  - Indicating uplink transmission occurrence of the pending broadcast transmission. Occurs in the same instance as EVENT_RX.

- `sd_ant_channel_id_set (1, 0, 1, 1)`
  - Re-assign RX scanning transmission channel back to wildcard as the channel ID of the last received packet would have been assigned to this channel.
- `sd_ant_broadcast_message_tx(1,8, buf)`
  - Prepare next pending broadcast transmission on RX scanning transmission channel.
- Etc.…

ANT asynchronous transmit channel events now occur asynchronously in the presence of other ANT channels.

In previous SoftDevices, 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.

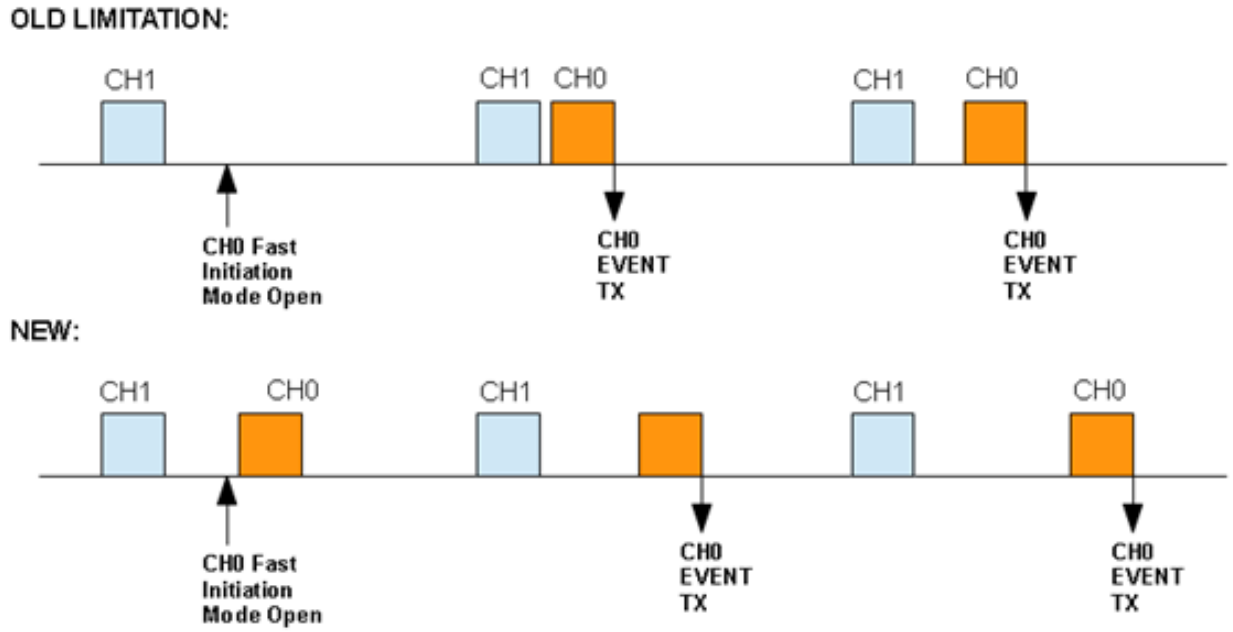
This limitation has now been removed. Asynchronous transmissions in the presence of other running channels will now be performed as soon as possible unless there is insufficient time before the next scheduled activity, which will then cause the transmission to be performed after the previously scheduled ANT activity has run.



ANT fast initiation channel now start as soon as possible in the presence of other ANT channels.

In previous SoftDevices, 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 as soon as possible unless there is insufficient time before the next scheduled activity, which will then cause the channel to start after the previously scheduled ANT activity has run.



Improved ANT RX Scanning Channel operation during application flash write/timeslot activity:

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