s310_nrf51422 migration document

Table of Contents

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.

Copyright (c) Nordic Semiconductor ASA. All rights reserved.

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 area for application can start at 0x20000008 instead of 0x20000004. The additional four bytes have been added to support interrupt forwarding to be done 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

`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 has 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 has 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()

`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, use the checkbox "Enable SoftDevice protection" in the "Program SoftDevice" tab 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 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

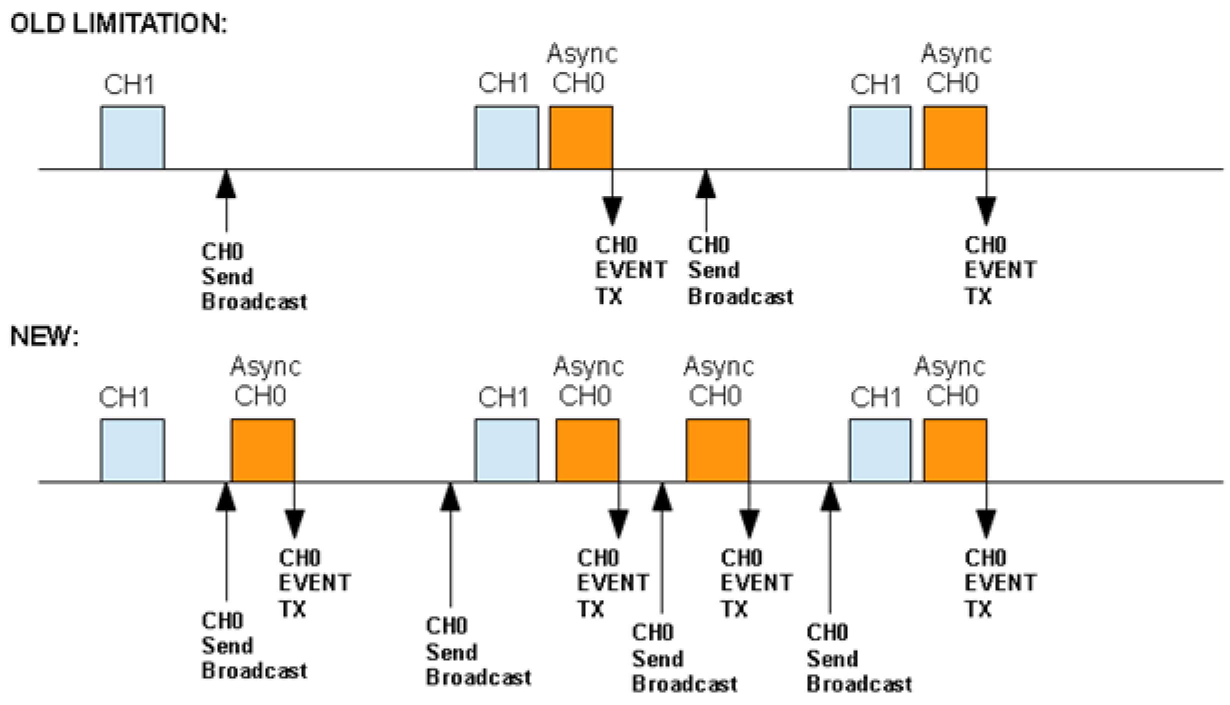
In order to allow continuous scanning devices to respond to incoming messages that matches one or more of the scanning channel ID fields, wildcard parameters (“0” value field in Device Number, Device Type and/or Transaction Type) can be specified when configuring the ID of the channel used for generating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

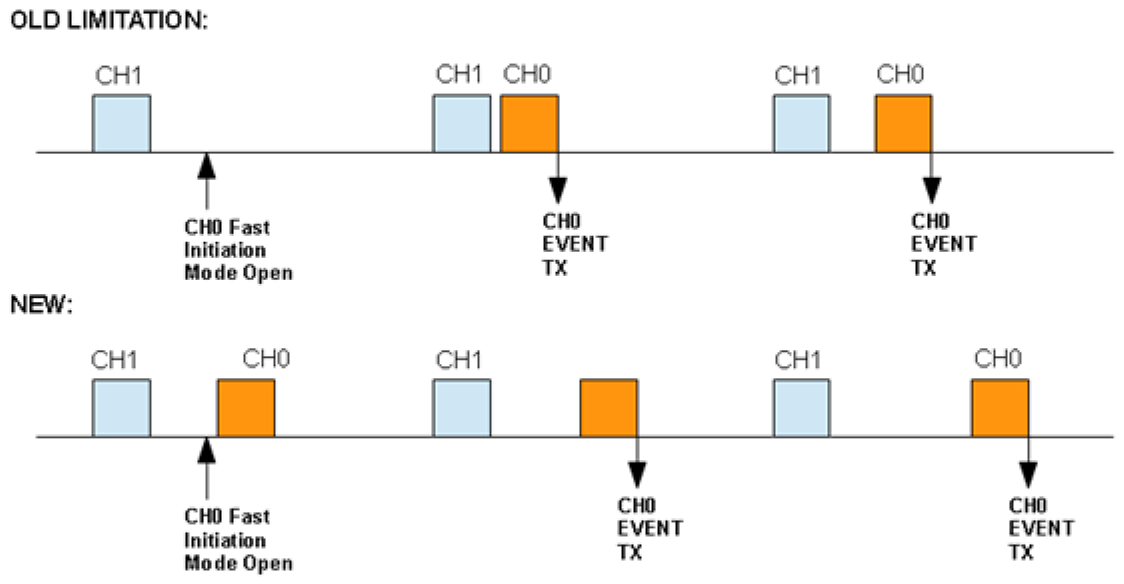
This limitation has now been removed. Asynchronous transmissions, in the presence of other running channels, are now performed immediately unless it is blocked by a previously scheduled radio activity. See the figure below.



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.