s14x_nrf5x migration document

Introduction to the s140_nrf52840 migration document

About the document

This document describes how to migrate to new versions of the s140 SoftDevices. The s140_nrf52840 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.

Example: To migrate from version 5.0.0 to version 5.2.0, first follow the instructions to migrate to 5.1.0 from 5.0.0, then follow the instructions to migrate to 5.2.0 from 5.1.0.

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

s140_nrf52840_5.0.0-2.alpha

This section describes how to migrate to s140_nrf52840_5.0.0-2.alpha from s140_nrf52840_5.0.0-1.alpha.

Required changes

SoftDevice RAM usage

The RAM usage of the SoftDevice has changed. `sd_ble_enable()` should be used to find the APP_RAM_BASE for a particular configuration.

New configuration API

Configuration parameters passed to `sd_ble_enable()` have been moved to the SoftDevice configuration API.

API updates

- A new SV call `sd_ble_cfg_set()` is added to set the configuration. This API can be called many times to configure different parts of the BLE stack. All configurations are optional. Configuration parameters not set by this API will take their default values.
- The SV call parameter `ble_enable_params_t * p_ble_enable_params` is removed from `sd_ble_enable()`. The SV call `sd_ble_cfg_set()` must be used instead. The parameters of this call are given in the following table:

Old API: `ble_enable_params_t` member	New API: `cfg_id` in `sd_ble_cfg_set()`
`common_enable_params.vs_uuid_count`	`BLE_COMMON_CFG_VS_UUID`
`common_enable_params.p_conn_bw_counts`	`BLE_CONN_CFG_GAP` (*)

`gap_enable_params.periph_conn_count`
`gap_enable_params.central_conn_count`
`gap_enable_params.central_sec_count`
	`BLE_GAP_CFG_ROLE_COUNT`

gap_enable_params.p_device_name	BLE_GAP_CFG_DEVICE_NAME
gatt_enable_params	BLE_CONN_CFG_GATT (*)
gatts_enable_params.service_changed	BLE_GATTS_CFG_SERVICE_CHANGED
gatts_enable_params.attr_tab_size	BLE_GATTS_CFG_ATTR_TAB_SIZE

(*) These configurations can be set per link.

Usage

Example pseudo code to set per link ATT_MTU using the new configuration API:

```
const uint16_t client_rx_mtu = 158;
const uint32_t long_att_conn_cfg_tag = 1;

/* set ATT_MTU for connections identified by long_att_conn_cfg_tag */
ble_cfg_t cfg;
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = long_att_conn_cfg_tag;
cfg.conn_cfg.params.gatt_conn_cfg.att_mtu = client_rx_mtu;
sd_ble_cfg_set(BLE_CONN_CFG_GATT, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t long_att_conn_handle;
/* Establish connection with long_att_conn_cfg_tag */
sd_ble_gap_adv_start(..., long_att_conn_cfg_tag);

[...]

/* Establish connection with BLE_CONN_CFG_TAG_DEFAULT, it will use default ATT_MTU
of 23 bytes */
sd_ble_gap_connect(..., BLE_CONN_CFG_TAG_DEFAULT);

[...]

/* Start ATT_MTU exchange */
sd_ble_gattc_exchange_mtu_request(long_att_conn_handle, client_rx_mtu);
```

BLE bandwidth configuration

The BLE bandwidth configuration and application packet concept has been changed. Previously, the application could specify a bandwidth setting, which would result in a given queue size and a correpsonding given radio time allocated. Now the queue sizes and the allocated radio time have been separated. The application can now configure:

- Event length
- Write without response queue size
- Handle Value Notification queue size

These settings are configurable per link.

Note that now the configured queue sizes are not directly related to on-air bandwidth:

- The application can configure one single packet to be queued in the SoftDevice, but still achieve full throughput if the application can queue packets fast enough during connection events.

- Even if the application configures a large number of packets to be queued, not all of them will be sent during a single connection event if the configured event length is not large enough to send the packets.

API updates

- The `ble_enable_params_t::common_enable_params.p_conn_bw_counts` parameter of the `sd_ble_enable()` SV call is replaced by the `sd_ble_cfg_set()` SV call with `cfg_id` parameter set to `BLE_CONN_CFG_GAP`. The following table shows how the old bandwidth configuration corresponds to the new one for the default ATT_MTU:

Old API: **`BLE_CONN_BWS`**	New API: **`ble_gap_conn_cfg_t::event_length`** in **`sd_ble_cfg_set()`**
`BLE_CONN_BW_LOW`	`BLE_GAP_EVENT_LENGTH_MIN`
`BLE_CONN_BW_MID`	`BLE_GAP_EVENT_LENGTH_DEFAULT`
`BLE_CONN_BW_HIGH`	`6`

  The bandwidth configuration is further described in the SDS.
- The `BLE_COMMON_OPT_CONN_BW` option is removed. Instead, during connection creation, the application should supply the `conn_cfg_tag` defined by the `ble_conn_cfg_t::conn_cfg_tag` parameter in the `sd_ble_cfg_set()` SV call.
- A new parameter `conn_cfg_tag` is added to `sd_ble_gap_adv_start()` and `sd_ble_gap_connect()` SV calls. To create a connection with a default configuration, `BLE_CONN_CFG_TAG_DEFAULT` should be provided in this parameter.
- The `BLE_EVT_TX_COMPLETE` event is split on two events: `BLE_GATTC_EVT_WRITE_CMD_TX_COMPLETE` and `BLE_GATTS_EVT_HVN_TX_COMPLETE`.
- The SV call `sd_ble_tx_packet_count_get()` is removed. Instead, the application can now configure packet counts per link, using the SV call `sd_ble_cfg_set()` with the `cfg_id` parameter set to `BLE_CONN_CFG_GATTC` and `BLE_CONN_CFG_GATTS`.

Usage

Example pseudo code to set configuration that corresponds to the old `BLE_CONN_BW_HIGH` bandwidth configuration both in throughput and packet queueing capability:

```
const uint32_t high_bw_conn_cfg_tag = 1;
ble_cfg_t cfg;

/* configure connections identified by high_bw_conn_cfg_tag */

/* set connection event length */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gap_conn_cfg.event_length = 6; /* 6 * 1.25 ms = 7.5 ms
corresponds to the old BLE_CONN_BW_HIGH for default ATT_MTU */
cfg.conn_cfg.params.gap_conn_cfg.conn_count = 1;   /* application needs one link
with this configuration */
sd_ble_cfg_set(BLE_CONN_CFG_GAP, &cfg, ...);

/* set HVN queue size */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gatts_conn_cfg.hvn_tx_queue_size = 7; /* application wants to
queue 7 HVNs */
sd_ble_cfg_set(BLE_CONN_CFG_GATTS, &cfg, ...);

/* set WRITE_CMD queue size */
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = high_bw_conn_cfg_tag;
cfg.conn_cfg.params.gattc_conn_cfg.write_cmd_tx_queue_size = 0; /* application is
not giong to send WRITE_CMD, so set to 0 to save memory */
sd_ble_cfg_set(BLE_CONN_CFG_GATTC, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t high_bw_conn_handle;
/* Establish connection with high_bw_conn_cfg_tag */
sd_ble_gap_adv_start(..., high_bw_conn_cfg_tag);
```

Data Length Update Procedure

The application now has to respond to the Data Length Update Procedure when initiated by the peer. See the description of the Data Length Update Procedure in the New functionality section for more details.

Required changes:

```
case BLE_GAP_EVT_DATA_LENGTH_UPDATE_REQUEST:
{
  /* Allow SoftDevice to choose Data Length Update Procedure parameters
automatically. */
  sd_ble_gap_data_length_update(p_ble_evt->evt.gap_evt.conn_handle, NULL, NULL);
  break;
}
case BLE_GAP_EVT_DATA_LENGTH_UPDATE:
{
  /* Data Length Update Procedure completed, see
p_ble_evt->evt.gap_evt.params.data_length_update.effective_params for negotiated
parameters. */
  break;
}
```

Access to `RAM[x].POWER` registers

SoftDevice APIs are updated to provide access to the `RAM[x].POWER` registers instead of the deprecated `RAMON/RAMONB`.

API updates

- `sd_power_ramon_set()` SV call is replaced with `sd_power_ram_power_set()`.
- `sd_power_ramon_clr()` SV call is replaced with `sd_power_ram_power_clr()`.
- `sd_power_ramon_get()` SV call is replaced with `sd_power_ram_power_get()`.

API rename

Some APIs were renamed. Applications that use the old names must be updated.

API updates

- `BLE_EVTS_PTR_ALIGNMENT` is renamed to `BLE_EVT_PTR_ALIGNMENT`.
- `BLE_EVTS_LEN_MAX` is renamed to `BLE_EVT_LEN_MAX`.
- `GATT_MTU_SIZE_DEFAULT` is renamed to `BLE_GATT_ATT_MTU_DEFAULT`.
- The GAP option `BLE_GAP_OPT_COMPAT_MODE` is renamed to `BLE_GAP_OPT_COMPAT_MODE_1`.
- `ble_gap_opt_compat_mode_t` structure is renamed to `ble_gap_opt_compat_mode_1_t`.
- `ble_gap_opt_compat_mode_t::mode_1_enable` structure member is renamed to `ble_gap_opt_compat_mode_1_t::enable`.
- `ble_gap_opt_t::compat_mode` structure member is renamed to `ble_gap_opt_t::compat_mode_1`.

Proprietary L2CAP API removed

The proprietary API for sending and receiving data over L2CAP is removed.

API updates

- The SV calls `sd_ble_l2cap_cid_register()`, `sd_ble_l2cap_cid_unregister()`, and `sd_ble_l2cap_tx()`are removed.
- `BLE_L2CAP_EVT_RX` event is removed.
- The following defines are removed: `BLE_L2CAP_MTU_DEF`, `BLE_L2CAP_CID_INVALID`, `BLE_L2CAP_CID_DYN_BASE`, `BLE_L2CAP_CID_DYN_MAX`.

New functionality

Data Length Update Procedure

The application is given control of the Data Length Update Procedure. The application can initiate the procedure and has to respond when initiated by the peer.

API updates

- A new SV call `sd_ble_gap_data_length_update()` is added to initiate or respond to a Data Length Update Procedure.
- The `BLE_EVT_DATA_LENGTH_CHANGED` event is replaced with `BLE_GAP_EVT_DATA_LENGTH_UPDATE`.
- A new event `BLE_GAP_EVT_DATA_LENGTH_UPDATE_REQUEST` is added to notify that a Data Length Update request has been received. `sd_ble_gap_data_length_update()` must be called by the application after this event has been received to continue the Data Length Update Procedure.
- The GAP option `BLE_GAP_OPT_EXT_LEN` is removed. The `sd_ble_gap_data_length_update()` SV call should be used instead.

Usage

- The Data Length Update Procedure can be initiated locally or by peer device.
- Following is the pseudo code for the case where Data Length Update Procedure is initiated by application:

```
const uint16_t client_rx_mtu = 247;
const uint32_t long_att_conn_cfg_tag = 1;

/* ATT_MTU must be configured first */
ble_cfg_t cfg;
memset(&cfg, 0, sizeof(ble_cfg_t));
cfg.conn_cfg.conn_cfg_tag = long_att_conn_cfg_tag;
cfg.conn_cfg.params.gatt_conn_cfg.att_mtu = client_rx_mtu;
sd_ble_cfg_set(BLE_CONN_CFG_GATT, &cfg, ...);

/* Enable the BLE Stack */
sd_ble_enable(...);

[...]

uint16_t long_att_conn_handle;
/* Establish connection */
sd_ble_gap_adv_start(..., long_att_conn_cfg_tag);

[...]

/* Start Data Length Update Procedure, can be done without ATT_MTU exchange */
ble_gap_data_length_params_t params = {
  .max_tx_octets  = client_rx_mtu + 4,
  .max_rx_octets  = client_rx_mtu + 4,
  .max_tx_time_us = BLE_GAP_DATA_LENGTH_AUTO,
  .max_rx_time_us = BLE_GAP_DATA_LENGTH_AUTO
};
sd_ble_gap_data_length_update(long_att_conn_handle, &params, NULL);

[...]

case BLE_GAP_EVT_DATA_LENGTH_UPDATE:
{
  /* Data Length Update Procedure completed, see
p_ble_evt->evt.gap_evt.params.data_length_update.effective_params for negotiated
parameters. */
  break;
}
```

New compatibility mode

A new compatibility mode is added to enable interoperability with central devices that may initiate version exchange and feature exchange control procedures in parallel. To enable this mode, use the `sd_ble_opt_set()` SV call with the `opt_id` parameter set to `BLE_GAP_OPT_COMPAT_MODE_2`.

Slave latency configuration

It is now possible to disable and enable slave latency on an active peripheral link. To disable or re-enable slave latency, use the `sd_ble_opt_set()` SV call with the `opt_id` parameter set to `BLE_GAP_OPT_SLAVE_LATENCY_DISABLE`.

Support for high accuracy LFCLK oscillator source

It is now possible to configure the SoftDevice with higher accuracy LFCLK oscillator source. Four new levels are defined:

```
#define NRF_CLOCK_LF_XTAL_ACCURACY_10_PPM  (8) /**< 10 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_5_PPM   (9) /**<  5 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_2_PPM  (10) /**<  2 ppm */
#define NRF_CLOCK_LF_XTAL_ACCURACY_1_PPM  (11) /**<  1 ppm */
```

New power failure levels

It is now possible to configure the SoftDevice with all the new power failure levels introduced in NRF52. Levels that are added:

```
NRF_POWER_THRESHOLD_V17       /**< Set the power failure threshold to 1.7 V. */
NRF_POWER_THRESHOLD_V18       /**< Set the power failure threshold to 1.8 V. */
NRF_POWER_THRESHOLD_V19       /**< Set the power failure threshold to 1.9 V. */
NRF_POWER_THRESHOLD_V20       /**< Set the power failure threshold to 2.0 V. */
NRF_POWER_THRESHOLD_V22       /**< Set the power failure threshold to 2.2 V. */
NRF_POWER_THRESHOLD_V24       /**< Set the power failure threshold to 2.4 V. */
NRF_POWER_THRESHOLD_V26       /**< Set the power failure threshold to 2.6 V. */
NRF_POWER_THRESHOLD_V28       /**< Set the power failure threshold to 2.8 V. */
```

s140_nrf52840_5.0.0-1.alpha

This section describes how to migrate to s140_nrf52840_5.0.0-1.alpha from s132_nrf52_3.0.0. This SoftDevice is designed to take advantage of the new features of the nrf52840 chip.

Required changes

SoftDevice flash and RAM usage

The size of the SoftDevice has changed and therefore a change to the application project file is required.

For Keil this means:

1. Go into the properties of the project and find the Target tab
2. Change IROM1 Start to `0x20000`.

If the project uses a scatter file or linker script instead, then these must be updated accordingly.

The RAM usage of SoftDevice has also changed. `sd_ble_enable()` should be used to find the APP_RAM_BASE for a particular configuration.

Renamed defines

Some defines have been renamed to make the API more consistent. Any code using these defines has to be updated with the new names:

- `GATT_MTU_SIZE_DEFAULT` renamed to `BLE_GATT_MTU_SIZE_DEFAULT`
- `BLE_EVTS_LEN_MAX` renamed to `BLE_EVT_LEN_MAX`
- `BLE_EVTS_PTR_ALIGNMENT` renamed to `BLE_EVT_PTR_ALIGNMENT`

New functionality

Multiple PHYs

The SoftDevice introduces support for using multiple PHYs to adapt the speed and reliability of data transmission to the channel capacity. For higher throughput, a 2 Mbps PHY is supported. For higher reliability, a 125kbps Coded PHY is supported.

API updates

- A new GAP option, `BLE_GAP_OPT_PREFERRED_PHYS_SET`, has been added to indicate to the controller about which PHYs the controller shall prefer so it can respond to any requests to update PHYs by peers.
- A new SV call, `sd_ble_gap_phy_request()`, has been added to request the controller to attempt to change to a new PHY.
- A new event, `BLE_GAP_EVT_PHY_UPDATE`, has been added to indicate that the PHY of a connection has changed or that a local initiated PHY update procedure has finished.

Usage

Example pseudo code for setting the preferred PHYs for new connections
Note: This will only have an effect if the peer device initiates the procedure to change the PHY. The stack will not initiate a PHY Update procedure autonomously.

```
ble_opt_t opts;
opts.gap_opt.preferred_phys.tx_phys = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_2MBPS;
opts.gap_opt.preferred_phys.rx_phys = BLE_GAP_PHY_1MBPS | BLE_GAP_PHY_2MBPS;
TEST_SD_UTIL_NRF_SUCCESS_OR_ASSERT(sd_ble_opt_set(BLE_GAP_OPT_PREFERRED_PHYS_SET,
&opts) );

[ Advertise and connect / Scan and connect ]
```

Request the controller to attempt to change to a new PHY for an established connection:

```
ble_gap_phys_t phys = {BLE_GAP_PHY_CODED, BLE_GAP_PHY_CODED};
sd_ble_gap_phy_request(conn_handle, &phys);
```

Handle PHY Update event:

```
/* Handle the event */
case BLE_GAP_EVT_PHY_UPDATE:
 if (ble_event.evt.gap_evt.params.phy_update.status == BLE_HCI_STATUS_CODE_SUCCESS)
 {
   // The PHY was changed (after either the application or the peer requested it)
   // ble_event.evt.gap_evt.params.phy_update.tx_phy and
ble_event.evt.gap_evt.params.phy_update.rx_phy contain the new PHYs
 }
 else
 {
   // A PHY update was requested which could not be performed successfully
 }
```

Higher TX power on nRF52840

The SoftDevice now supports configuring higher TX power to be used with nRF52840.

The following additional values are supported by the `sd_ble_gap_tx_power_set()` SV-call +2dBm, +5dBm, +6dBm, +7dBm, +8dBm, +9dBm.

These power levels can be used in the same way the existing power levels are used in the s132_nrf52_3.0.0 SoftDevice.