s115 migration document

Introduction to the s115 migration document

About the document

This document describes how to migrate to new versions of the s115 SoftDevice. The s115 release notes should be read in conjunction with this document.

For each version, we have the following sections:

- "Required changes" describes the changes that need to be done in the application when migrating from an older version of the SoftDevice.
- "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.

Contents

s115_9.0.0-4.prototype

This section describes how to adapt to changes and use the new features of s115_9.0.0-4.prototype when migrating from s115_9.0.0-3.prototype.

Required changes

Recommended changes

New functionality

s115_9.0.0-3.prototype

This section describes how to adapt to changes and use the new features of s115_9.0.0-3.prototype when migrating from s112_nrf52_7.2.0.

Required changes

- `sd_flash_protect` has been removed. RRAM Immutable Boot Region can be configured in UICR. Access privileges for RRAM regions can be configured using `RRAMC->REGION`. See the RRAM chapter in the datasheet for more information.
- The `nrf_nvic.h` header has been removed and CMSIS functions shall be used instead. The application shall not touch the SoftDevice interrupts while the SoftDevice is enabled. The SoftDevice does not support disabling interrupts globally.

    - Use `NVIC_ClearPendingIRQ` instead of `sd_nvic_ClearPendingIRQ`
    - Use `NVIC_DisableIRQ` instead of `sd_nvic_DisableIRQ`
    - Use `NVIC_EnableIRQ` instead of `sd_nvic_EnableIRQ`
    - Use `NVIC_GetPendingIRQ` instead of `sd_nvic_GetPendingIRQ`
    - Use `NVIC_GetPriority` instead of `sd_nvic_GetPriority`
    - Use `NVIC_SetPendingIRQ` instead of `sd_nvic_SetPendingIRQ`
    - Use `NVIC_SetPriority` instead of `sd_nvic_SetPriority`
    - Use `NVIC_SystemReset` instead of `sd_nvic_SystemReset`

  For critical regions the `BASEPRI` register can be used to disable all interrupts except the SoftDevice high priority interrupts:

```
/* Old API */
uint32_t nested;
sd_nvic_critical_region_enter(&nested);
[...] /* Critical region */
sd_nvic_critical_region_exit(nested);

/* New API */
uint32_t bp = __get_BASEPRI();
__set_BASEPRI_MAX(2U << (8U - __NVIC_PRIO_BITS));
__ISB();
[...] /* Critical region */
__set_BASEPRI(bp);
__ISB();
```

  It is not possible to call SoftDevice APIs while in a critical region.

- The SoftDevice PPI API has been removed. The application should use the `nrfx_dppi` and `nrfx_ppib` drivers or the `nrfx_gppi` helper provided by nrfx. The application shall not touch the PPI resources used by the SoftDevice (see `nrf_sd_def.h`).
- `sd_power_dcdc_mode_set` has been removed and `NRF_REGULATORS->VREGMAIN.DCDCEN` should be used instead.
- `sd_power_system_off` has been removed and `NRF_REGULATORS->SYSTEMOFF` should be used instead.
- `sd_power_reset_reason_get` and `sd_power_reset_reason_clr` have been removed and `NRF_RESET->RESETREAS` should be used instead.

- sd_power_ram_power_set, sd_power_ram_power_get and sd_power_ram_power_clr have been removed and NRF_MEMCONF->CONTROL should be used instead.
- It is required to seed the random number generator whenever the NRF_EVT_RAND_SEED_REQUEST event is raised. Seeding is done using the new sd_rand_seed_set API. Generating random numbers (sd_rand_application_vector_get) or enabling Bluetooth (sd_ble_enable) will fail if the random number generator has not been seeded.

  The following pseudo code shows how seeding can be done:

```
case NRF_EVT_RAND_SEED_REQUEST:
{
  uint8_t seed[SD_RAND_SEED_SIZE]:

  /* Fetch NIST SP 800-90B compliant entropy */
  trng_get(seed, sizeof(seed));
  sd_rand_seed_set(seed);

  break;
}
```

Recommended changes

- sd_app_evt_wait is deprecated and should no longer be used. Instead do the following:

```
/* Wait for an event. */
__WFE();

/* Clear Event Register */
__SEV();
__WFE();
```

- The SoftDevice no longer has pools for secure random numbers and secure random numbers are generated on-demand when sd_rand_application_vector_get is called. sd_rand_application_pool_capacity_get and sd_rand_application_bytes_available_get are deprecated and should not be used.

New functionality

LE Data Packet Length Extension (DLE)

The Data Length Update procedure enables the SoftDevice to use longer packets on the link layer level. Now link layer packets with up to 251 byte payloads are supported. The application can initiate the procedure and has to respond when it is 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.
- A new event BLE_GAP_EVT_DATA_LENGTH_UPDATE is added to notify that the link layer PDU length has changed.
- 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.

Usage

- The Data Length Update procedure can be initiated locally or by the peer device.
- The following pseudo code is for the case where the Data Length Update procedure is initiated by the 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;
}
```