# Nordic SDK-Based Application Development

## BL654PA Module

*Application Note* *v1.1*

## 1 Introduction

This application note describes the steps involved in developing Nordic SDK based applications for the BL654PA module. It focuses on regulatory requirements and how these are met by the user application.

## 2 Nomenclature

- Source code snippets are displayed using the `Courier New font`.
- References to Nordic SDK API content in code snippets and main body text are displayed using ***bold, italicized text***.
- References to user code in the main body text are displayed using **bold text**.
- Datatypes used throughout are defined by the Nordic SDK.

## 3 Overview

Laird BL654PA modules incorporate a Skyworks SKY66112-11 front-end module. This integrates a Power Amplifier (PA) for transmitted data and a Low Noise Amplifier (LNA) for received data. The addition of this part facilitates greater operating range for the BL654PA when compared to the standard BL654 module.

When operating at the greater transmission power levels offered by the power amplifier, it is possible to exceed duty-cycle related stipulations specified by regulatory authorities. The *smart*BASIC core application made available to end-users implements the measures necessary to ensure all regulatory requirements are complied with. Should the end-user wish to develop a Nordic SDK-based application however, implementation of these measures will be necessary.

This application note describes how interfacing with and control of the Skyworks front-end module is achieved and the measures necessary within the Nordic SDK based application to comply with regulatory requirements.

## 4 Assumptions

This document assumes that the end-user application is developed with the Nordic SDK in conjunction with a Nordic Softdevice-based protocol stack.

Code snippets presented herein are based upon SDK version 16.0 for the SD140 SoftDevice, v7.0.1 API.

End-user applications are assumed for Bluetooth Low Energy; Bluetooth Classic is not considered in this application note.



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

# 5 SKYWORKS FRONT-END MODULE INTERFACE

The control interface to the Skyworks front-end module from the host Nordic NRF52840 is shown in Figure 1.

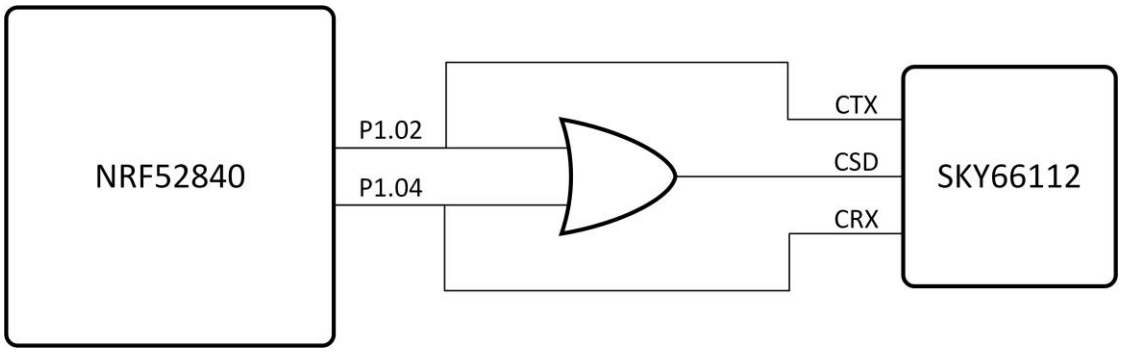


***Figure 1: SKY66112 control interface***

Two NRF52840 GPIO lines, P1.02 and P1.04, are used to control the front-end module. The GPIO are lines are driven to a high logic level to activate a front-end module mode. Table 1 describes the pins with which the front-end module interfaces.

***Table 1: Skyworks SKY66112 control interface***

| Pin | Description |
|---|---|
| CRX | Enables the front-end module low noise amplifier for reception |
| CSD | Places the front-end module in sleep mode, disabling both the power amplifier and low noise amplifier, when at a low logic level |
| CTX | Enables the front-end module power amplifier for transmission |

The CSD pin is connected via an OR gate to both P1.02 and P1.04 to ensure sleep mode is disabled when transmission or reception is taking place. Table 2 summarizes the logic associated with the control interface.

***Table 2: Skyworks SKY66112 control interface truth table***

| Mode | NRF52840 | | SKY66112 | | |
|---|---|---|---|---|---|
| | P1.02 | P1.04 | CTX | CSD | CRX |
| Receive LNA Mode | 0 | 1 | 0 | 1 | 1 |
| Sleep Mode | 0 | 0 | 0 | 0 | 0 |
| Transmit High Power Mode | 1 | 0 | 1 | 1 | 0 |

Manual control of the NRF52840 GPIO lines that form the control interface to the SKY66112 is not necessary. Rather, during initialization of the user application, details of these are passed to the Softdevice. Control of the GPIO lines is then performed by the Softdevice during transmit and receive operations.

**Note:** The NRF52840 GPIO lines that form the control interface must be initialised by the end-user application.



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

# 6 Interaction with the SKY66112 via the Nordic SDK

The Nordic SDK incorporates support for front-end modules via predefined types and API calls.

To allow event-driven control of the pins associated with the front-end module, the NRF52840 GPIOTE (General Purpose Input/Output Tasks and Events) and PPI (Programmable Peripheral Interconnect) modules are used by the Softdevice.

At the Softdevice level, events of type EVENTS_READY for TX result in the pin used to control the PA being set to its active level. Events of type EVENTS_READY for RX result in the pin used to control the LNA being set to its active level.

In both cases, events of type EVENTS_DISABLED for the radio result in both pins being driven to the defined inactive level.

Timers associated with the EVENTS_READY_FOR_TX and EVENTS_READY_FOR_RX events are used to delay commencement of the radio transmit or receive operation following assertion of the associated front-end module control pin. These delays are necessary to allow the appropriate front-end module section to stabilize.

No delay is necessary following completion of the radio transmit or receive operation.

The timings and associated events are shown in Figure 2.

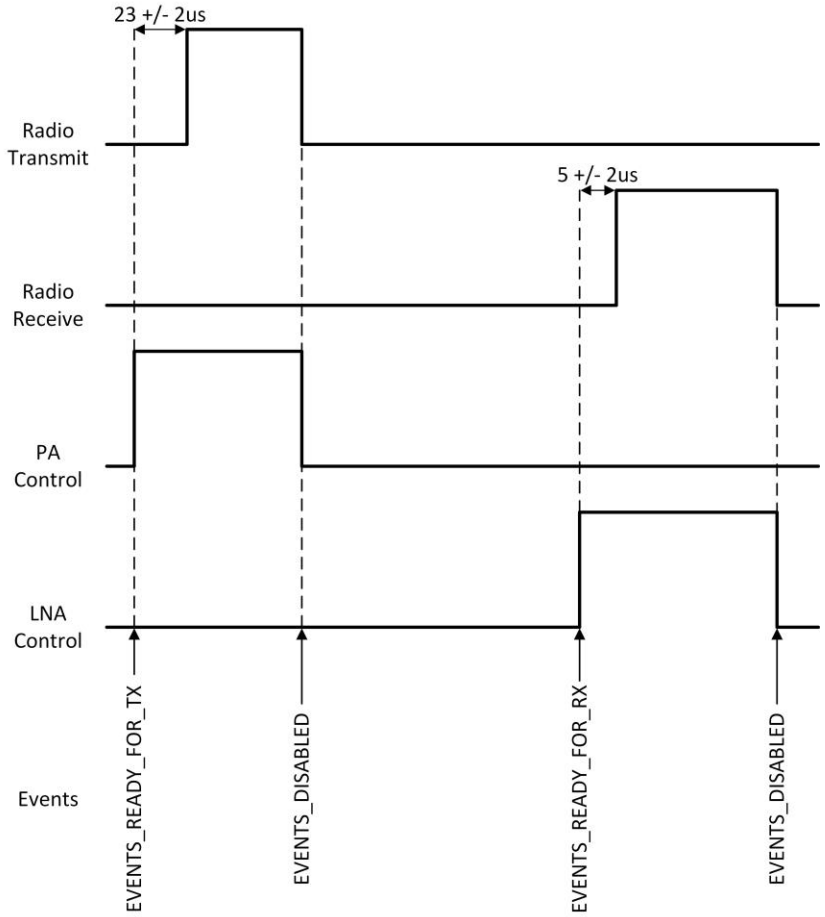


***Figure 2: SKY66112 control interface timing and associated events***



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

## 6.1 Front-End Module SDK Types

The following unions and structures were transcribed from Nordic SDK revision 16.0.

Front-end module support is implemented in the Nordic SDK via the ***ble_opt_t*** union. This is defined as follows:

```
typedef union
{
  ble_common_opt_t common_opt ;
  ble_gap_opt_t gap_opt ;
} ble_opt_t ;
```

PA/LNA related parameters reside within the ***common_opt*** instance of the union type ***ble_common_opt_t***.
***Ble_common_opt_t*** is defined as shown below.

```
typedef union
{
  ble_common_opt_conn_bw_t conn_bw ;
  ble_common_opt_pa_lna_t pa_lna ;
  ble_common_opt_conn_evt_ext_t conn_evt_ext ;
} ble_common_opt_t ;
```

Front-end module related items reside within the ***pa_lna*** instance of ***ble_common_opt_pa_lna_t***.

The ***ble_common_opt_pa_lna_t*** structure is defined as shown below.

```
typedef struct
{
   ble_pa_lna_cfg_t pa_cfg ;
   ble_pa_lna_cfg_t lna_cfg ;
   uint8_t ppi_ch_id_set ;
   uint8_t ppi_ch_id_clr ;
   uint8_t gpiote_ch_id ;
} ble_common_opt_pa_lna_t ;
```

Its members can be described as follows:

- ***pa_cfg*** – This is an instance of the ***ble_pa_lna_cfg_t*** type used to describe the pin that controls the PA of the front-end module.
- ***lna_cfg*** – This is an instance of the ***ble_pa_lna_cfg_t*** type used to describe the pin that controls the LNA of the front-end module.
- ***ppi_ch_id_set*** – An 8-bit value defining the PPI channel triggered when radio operations start.
- ***ppi_ch_id_clr*** – An 8-bit value defining the PPI channel triggered when radio operations end.
- ***gpiote_ch_id*** – An 8-bit value defining the GPIOTE channel used to toggle the GPIOs associated with radio transmission start and end events.

The ***ble_pa_lna_cfg_t*** struct is defined as follows:

```
typedef struct
{
     uint8_t enable :1 ;
     uint8_t active_high :1 ;
     uint8_t gpio_pin :6 ;
} ble_pa_lna_cfg_t ;
```

The members of the ble_pa_lna_cfg_t struct can be described as follows:

- ***enable*** – Enables control of the associated GPIO pin
- ***active_high*** – Set if the associated output pin is active high
- ***gpio_pin*** – The Nordic SDK formatted identifier of the GPIO used to control the front-end module



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

During initialization, an instance of ***ble_opt_t*** is populated by the user application, then passed to the SDK API function ***sd_ble_opt_set*** to achieve the desired configuration.

## 6.2 Compile Time Definitions

For readability, macros should be defined for the pins used to control the front-end module. This is achieved as follows:

```
#define DRIVER_PALNA_RX_PIN NRF_GPIO_PIN_MAP(1,4)
#define DRIVER_PALNA_TX_PIN NRF_GPIO_PIN_MAP(1,2)
```

The Nordic SDK API macro ***NRF_GPIO_PIN_MAP*** is used to translate port and pin information into a format intelligible by successive GPIO manipulation API calls.

These definitions should be used hereon when referring to the pins.

As described in section 5, pin **DRIVER_PALNA_RX_PIN** is set to a high logic level when receiving data, and low otherwise, and pin **DRIVER_PALNA_TX_PIN** is set to a high logic level when transmitting data and low otherwise.

---

**Note:** Precautions must be taken to ensure the PA and LNA are not enabled at the same time. Doing so and transmitting data may cause irreparable damage to the front-end module.

---

## 6.3 Variables Required

The following variables are required for storing details of PPI and GPIOTE channel allocations.

```
// This is the channel triggered during radio enable events
nrf_ppi_channel_t ppi_set_ch ;
// This is the channel triggered during radio disable events
nrf_ppi_channel_t ppi_clr_ch ;
// This the GPIOTE channel associated with the above
int16_t nGpioTeChannelNum ;
```

## 6.4 Initialization Code

The pins used to control the SKY66112 must be set to a low logic level and configured as outputs during the initialization phase of the user application. Note this should be performed as quickly as possible following reset of the user application to minimize current consumption.

This is achieved with the Nordic SDK API calls ***nrf_gpio_pin_write*** and ***nrf_gpio_cfg_output***, as follows.

```
nrf_gpio_pin_write(DRIVER_PALNA_TX_PIN, 0) ;
nrf_gpio_pin_write(DRIVER_PALNA_RX_PIN, 0) ;
nrf_gpio_cfg_output(DRIVER_PALNA_TX_PIN) ;
nrf_gpio_cfg_output(DRIVER_PALNA_RX_PIN) ;
```

Upon configuration of the pins as outputs, code of the following form needs to be executed to configure the PPI and GPIOTE modules, and to associate the allocated channels and pins with predefined radio events.

```
uint32_t nrfErr ;
ble_opt_t opt = {0} ;

// Initialize PPI module – note this is only necessary if not performed prior to this step.
// An error code will be returned if this is the case
nrfErr = nrf_drv_ppi_init() ;
assert(nrfErr == NRF_SUCCESS) ;

// Allocate the next free PPI channel available
nrfErr = nrf_drv_ppi_channel_alloc(&ppi_set_ch) ;
assert(nrfErr == NRF_SUCCESS) ;
```



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

```
// Allocate the next free PPI channel available
nrfErr = nrf_drv_ppi_channel_alloc(&ppi_clr_ch) ;
assert(nrfErr == NRF_SUCCESS) ;

// GPIOTE channel used for radio pin toggling
opt.common_opt.pa_lna.gpiote_ch_id = nGpioTeChannelNum ;
// PPI channel used for radio disable events
opt.common_opt.pa_lna.ppi_ch_id_clr = ppi_clr_ch ;
// PPI channel used for radio enable events
opt.common_opt.pa_lna.ppi_ch_id_set = ppi_set_ch ;
// Set the pin to be active high
opt.common_opt.pa_lna.lna_cfg.active_high = 1 ;
// Enable toggling for this amplifier
opt.common_opt.pa_lna.lna_cfg.enable = 1 ;
// The GPIO pin to toggle for this amplifier
opt.common_opt.pa_lna.lna_cfg.gpio_pin = DRIVER_PALNA_RX_PIN ;
// Set the pin to be active high
opt.common_opt.pa_lna.pa_cfg.active_high = 1 ;
// Enable toggling for this amplifier
opt.common_opt.pa_lna.pa_cfg.enable = 1 ;
// The GPIO pin to toggle for this amplifier
opt.common_opt.pa_lna.pa_cfg.gpio_pin = DRIVER_PALNA_TX_PIN ;
// This stack function commits the allocations to memory
// accessible by the SoftDevice
nrfErr = sd_ble_opt_set(BLE_COMMON_OPT_PA_LNA, &opt) ;
assert(nrfErr == NRF_SUCCESS) ;
```



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

# 7 LIMITATION OF TRANSMIT POWER

The transmit power configured by the end-user application must consider both the characteristics of the SKY66112 power amplifier and the regulatory requirements associated with the physical layer selected. These are summarized in Table 3. Customer has control of nRF52840 TX RF power setting **only** as per Table 3 and the resulting SKY66112 TX power is decided, i.e. the SKY66112 can be thought of as a RF gain block.

***Table 3: Transmit power limitations for baudrate, physical layer and SKY66112 power amplifier characteristics***

| NRF52840 TX Power (dBm) | SKY66112 TX Power (dBm) | PHY | | |
|---|---|---|---|---|
| | | 1 Mbps | 2 Mbps | Coded PHY 125 kbps |
| 8 | 30 | Exceeds SKY66112 output power limit | | |
| 7 | 29 | Exceeds SKY66112 output power limit | | |
| 6 | 28 | Exceeds SKY66112 output power limit | | |
| 5 | 27 | Exceeds SKY66112 output power limit | | |
| 4 | 26 | Exceeds SKY66112 output power limit | | |
| 3 | 25 | Exceeds SKY66112 output power limit | | |
| 2 | 24 | Exceeds SKY66112 output power limit | | |
| 0 | 22 | Exceeds SKY66112 output power limit | | |
| -4 | 18 | Allowed | Allowed | Not Allowed |
| -8 | 14 | Allowed | Allowed | Allowed |
| -12 | 6 | Allowed | Allowed | Allowed |
| -16 | 0 | Allowed | Allowed | Allowed |
| -20 | -6 | Allowed | Allowed | Allowed |
| -40 | -26 | Allowed | Allowed | Allowed |

**Notes:** For 1 Mbps and 2 Mbps BLE, the BL654PA is certified for RF TX output power of 18 dBm maximum.

For Coded PHY 125 kbps BLE, the BL654PA is certified for reduced RF TX output power of 14 dBm maximum. This ensures compliance with FCC, IC, AUS, and NZ power spectral density requirements.

## 7.1 Configuration of Transmit Power Via the Nordic SDK

The Nordic SDK API call ***sd_ble_gap_tx_power_set*** is used to configure the radio transmit power. Power levels can be individually set for advertising, scanning/initiating, and each active connection. The end-user application must ensure appropriate power levels are set in each case.

## 7.2 Transmit Power Validation

The Nordic SDK does not take into account the presence of front-end module amplification when setting transmit power levels. The end-user application must instead perform validation checks to ensure a valid transmit power configuration is set.

Before setting transmit power levels via SDK API calls, code of the following form should be used to check the intended transmit power level. The former validates transmit power levels for 1 Mbps and 2 Mbps PHYs, and the latter for Coded PHY at 125 Kbps.

```
#define TX_POWER_DEFAULT ((int8_t) RADIO_TXPOWER_TXPOWER_Neg40dBm)

uint32_t CheckTransmitPower1Mbps2Mbps(int8_t transmitPower, int8_t *pActualTransmitPower)
{
     uint32_t nrfErr ;
     switch(transmitPower)
     {
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg4dBm):
```



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

```
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg8dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg40dBm):
               *pActualTransmitPower = transmitPower ;
               nrfErr = NRF_SUCCESS ;
          break ;
          default:
               *pActualTransmitPower = TX_POWER_DEFAULT ;
               nrfErr = NRF_ERROR_FORBIDDEN ;
     }
     return(nrfErr) ;
}

uint32_t CheckTransmitPowerCodedPHY125kbps(int8_t transmitPower, int8_t
*pActualTransmitPower)
{
     uint32_t nrfErr ;

     switch(transmitPower)
     {
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg8dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm):
          case((int8_t) RADIO_TXPOWER_TXPOWER_Neg40dBm):
               *pActualTransmitPower = transmitPower ;
               nrfErr = NRF_SUCCESS ;
          break ;
          default:
               *pActualTransmitPower = TX_POWER_DEFAULT ;
               nrfErr = NRF_ERROR_FORBIDDEN ;
     }
     return(nrfErr) ;
}
```

The following are suggested as a means of deriving a valid transmit power level from any input value.

```
int8_t GetEquivalentSdTxPower1Mbps2Mbps (int8_t transmitPower)
{
     int8_t actualTransmitPower ;

     switch(transmitPower)
     {
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg4dBm):
          {
               actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg4dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg8Bm):
          {
              actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg8dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm):
          {
              actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm):
          {
```



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

```
                actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm ;
           }
           break ;
           case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm):
           {
                actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm ;
           }
           break ;
           default:
                actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg40dBm ;
           break ;
     }
     return(actualTransmitPower) ;
}
```

```
int8_t GetEquivalentSdTxPowerCodedPHY125kbps (int8_t transmitPower)
{
     int8_t actualTransmitPower ;

     switch(transmitPower)
     {
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg8Bm):
          {
              actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg8dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm):
          {
              actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg12dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm):
          {
               actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg16dBm ;
          }
          break ;
          case(transmitPower >= (int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm):
          {
               actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg20dBm ;
          }
          break ;
          default:
               actualTransmitPower = (int8_t) RADIO_TXPOWER_TXPOWER_Neg40dBm ;
          break ;
     }
     return(actualTransmitPower) ;
}
```



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

## 7.3 Initialization

Following initialization, transmit power level is set to 0 dBm by the Softdevice. The end-user application must ensure this is set to an allowed value at start-up. Code of the following form can be used to achieve this.

```
// Set the tx power for the advertiser role - connection handle discarded
nrfErr = sd_ble_gap_tx_power_set(BLE_GAP_TX_POWER_ROLE_ADV, 0, transmitPower) ;
if (NRF_SUCCESS == nrfErr)
{
     // Set the tx power for the scanner/initiator - connection handle discarded
     nrfErr = sd_ble_gap_tx_power_set(BLE_GAP_TX_POWER_ROLE_SCAN_INIT, 0, transmitPower) ;
}
```

**Note:** No connections are active following initialization therefore no transmit power must be configured.

## 7.4 Runtime

Changes to transmit power during end-user application runtime must be validated via checks described in 7.2 prior to calls to the ***sd_ble_gap_tx_power_set*** API function.

# 8 Limitations of Duty-Cycle And PDU Length

Depending upon the physical layer in use, the PDU length must be restricted, and, for the 1 Mbps and 2 Mbps PHYs, the duty cycle must be restricted. These constraints are described as follows.

## 8.1 1 Mbps and 2 Mbps

For the LE 1M PHY and 2M LE PHY, the time spent by the end-user application transmitting data must be limited to comply with regulatory requirements. Over a 100ms observation period, no more than 24 milliseconds can be spent transmitting data on a single data channel. This is further dependent upon the transmit power of the radio and the widths of data packets sent.

The constraints necessary for implementation are shown in Table 4.

***Table 4: Duty cycle limitation for 1 Mbps and 2 Mbps***

| NRF52840 TX Power (dBm) | SKY66112 TX Power (dBm) | PHY | | Declared Maximum BLE Protocol TX RF Duty Cycle in 100 ms sweep |
|---|---|---|---|---|
| | | 1 Mbps | 2 Mbps | |
| | | Maximum PDU Length (bytes) | | |
| 8 | 30 | Not Allowed | Not Allowed | Not Allowed |
| 7 | 29 | Not Allowed | Not Allowed | Not Allowed |
| 6 | 28 | Not Allowed | Not Allowed | Not Allowed |
| 5 | 27 | Not Allowed | Not Allowed | Not Allowed |
| 4 | 26 | Not Allowed | Not Allowed | Not Allowed |
| 3 | 25 | Not Allowed | Not Allowed | Not Allowed |
| 2 | 24 | Not Allowed | Not Allowed | Not Allowed |
| 0 | 22 | Not Allowed | Not Allowed | Not Allowed |
| -4 | 18 | 200 | 200 | 24% (24 ms) plus margin |
| -8 | 14 | 251 | 251 | 24% (24 ms) plus margin |
| -12 | 6 | 251 | 251 | 24% (24 ms) plus margin |
| -16 | 0 | 251 | 251 | 24% (24 ms) plus margin |
| -20 | -6 | 251 | 251 | 24% (24 ms) plus margin |
| -40 | -26 | 251 | 251 | 24% (24 ms) plus margin |



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

## 8.2 125 kbps Coded PHY

No duty cycle restriction is necessary when the 125 kbps Coded PHY is in use. However, the PDU length must not be allowed to exceed 251 bytes in length.

**Note:** 125 kbps Coded PHY transmit power is limited to 14 dBm to pass FCC TX Power spectral Density. This passes FCC Band Edge emissions, so duty cycle demonstration or declaration is not required.

## 8.3 Configuration of Duty-Cycle via the Nordic SDK

For worst-case conditions, each Connection Event will be fully consumed by communications traffic (with associated inter-frame spaces). To ensure duty-cycle restrictions are complied with, the Connection Interval and the Event Length are adjusted, depending upon the number of connections required by the end-user application, to enforce the 24 milliseconds per 100 milliseconds per channel restriction.

Event Length is set using the Nordic SDK API call ***sd_ble_cfg_set***, with this being passed an instance of a ***ble_cgf_t*** structure that contains the necessary configuration parameters.

This is defined as follows.

```
typedef union
{
	ble_conn_cfg_t conn_cfg ;
	ble_common_cfg_t common_cfg ;
	ble_gap_cfg_t gap_cfg ;
	ble_gatts_cfg_t gatts_cfg ;
}ble_cfg_t ;
```

Parameters related to the connection interval reside in the ***conn_cfg*** instance of the ***ble_conn_cfg_t*** structure. This is defined as follows.

```
typedef struct
{
	uint8_t conn_cfg_tag ;
	union
	{
		ble_gap_conn_cfg_t gap_conn_cfg ;
		ble_gattc_conn_cfg_t gattc_conn_cfg ;
		ble_gatts_conn_cfg_t gatts_conn_cfg ;
		ble_gatt_conn_cfg_t gatt_conn_cfg ;
		ble_l2cap_conn_cfg_t l2cap_conn_cfg ;
	}params ;
}ble_conn_cfg_t ;
```

The ***gap_conn_cfg*** instance of the ***ble_gap_conn_cfg_t*** structure is then used to configure the connection parameters. This is defined as follows.

```
typedef struct
{
	uint8_t conn_count ;
	uint16_t event_length ;
} ble_gap_conn_cfg_t ;
```

The ***conn_count*** member determines how many connections the end-user application can support.

The ***event_length*** member determines how much time is allocated to each Connection during each Connection Interval. This is specified in multiples of 1.25 milliseconds.



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

**Note:** A value of 4 written to the ***event_length*** parameter configures an Event Length of 4 * 1.25 ms = 5 ms. This dedicates 5 milliseconds of each connection interval to each connection.

The connection interval is defined initially at compile time as shown below.

```
#define MIN_CONN_INTERVAL                MSEC_TO_UNITS(2000, UNIT_1_25_MS)
#define MAX_CONN_INTERVAL                MSEC_TO_UNITS(3000, UNIT_1_25_MS)
```

These values are passed to a call to the Nordic SDK API function ***gap_params_init*** during initialisation of the end-user application. The Connection Interval can be adjusted at run-time with successive calls to the SDK function ***sd_ble_gap_ppcp_set***, - this is called from within the ***gap_params_init*** function during start-up.

## 8.4 Calculation of Event Length for Connection Interval

For a given Connection Interval, Event Length, and number of Connections, the maximum time available for each connection can be calculated with the following.

Maximum Event Length = Connection Interval / Connection Count

Connection Events of up to this length then occur after each Connection Interval.

A Connection Interval of 100 ms for five Connections with an Event Length of 20 milliseconds is shown in Figure 3.

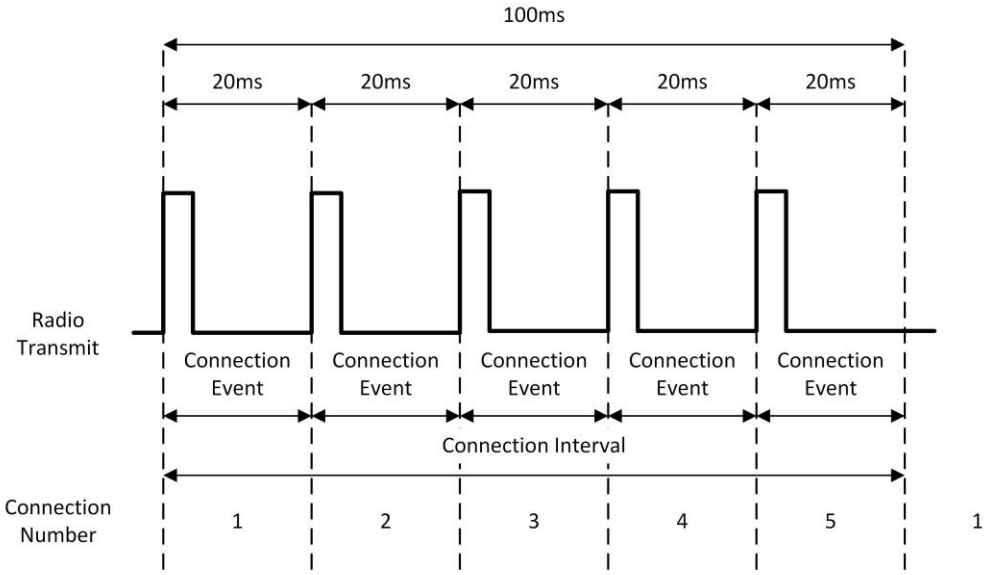


***Figure 3: Relationship of connection interval, event length, and number of connections***

Each connection event utilizes a different, randomly-selected data channel.

The maximum event length usable by the end-user application must be limited to 24 milliseconds. Even with a connection interval of 24 milliseconds and a single connection, a different data channel is selected such that the minimum time the same channel is used is (24ms * 38) 912 milliseconds.



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

## 8.5 Channel Map Configuration

All 37 channels must be enabled at all times to comply with regulatory requirements. Upon start-up, applications have their channel map set to enable all 37 channels by default. This must not be changed.

The channel map can be read and written using the ***ble_opt_get*** and ***ble_opt_set*** API calls.

Channel map parameters are read and written via an instance of the ***ble_gap_opt_ch_map_t*** structure, the elements of which can be defined as follows.

```
typedef struct
{
  // Connection handle – only applicable when reading the channel map
  uint16_t conn_handle;
  // The 37-bit channel map
  uint8_t ch_map[5];
} ble_gap_opt_ch_map_t;
```

An instance of ***ble_gap_opt_ch_map_t*** is housed within the ***ble_opt_t*** union, as described in section 6.1.

The channel map is read and written as follows.

```
#define CHANNEL_MAP_BYTE_CNT 5

uint32_t nrfErr ;
ble_opt_t opt = {0} ;
uint8_t channelMapDefault[CHANNEL_MAP_BYTE_CNT] = {0xFF,0xFF,0xFF,0xFF,0x1F} ;

// Connection handles are issued during connect events
opt.gap_opt.ch_map.conn_handle = conn_handle ;
// Read back the channel map
nrfErr = Ble_opt_get(BLE_GAP_OPT_CH_MAP, &opt) ;
assert(nrfErr==NRF_SUCCESS) ;
// Ensure all channels are enabled
if (memcmp(opt.gap_opt.ch_map.ch_map, channelMapDefault, CHANNEL_MAP_BYTE_CNT))
{
       // And enable all if not
       memcpy(opt.gap_opt.ch_map.ch_map,
              channelMapDefault,
              CHANNEL_MAP_BYTE_CNT) ;
       nrfErr = Ble_opt_set(BLE_GAP_OPT_CH_MAP, &opt) ;
assert(nrfErr==NRF_SUCCESS) ;
}
```

**Note:** Only central devices can request changes of the channel map. Peripheral devices must implement boundary checking code to reject changes of the channel map where all 37 channels are not enabled.



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

# 9 Verification Steps

**Note:** The end-user must validate their application to ensure the constraints described throughout are adhered to. A single data channel cannot be used to transmit data for more than 24 milliseconds over a 100 milliseconds observation time.

## 9.1 125 kbps Coded PHY

As described in the *Limitations of Duty-Cycle and PDU Length* section, there is no limitation of PDU size for the 125 kbps Coded PHY, other than that from the specification. All transmitted messages must be limited to a transmit power of 14 dBm, or lower.

## 9.2 1 Mbps and 2 Mbps LE PHY

Duty-cycle, transmit power, and PDU length must be restricted for the 1 Mbps and 2 Mbps LE physical layers.

A spectrum analyzer (in zero span mode) should be used to verify that in all cases, with a 100-millisecond sweep time, the total duration of all packets sent does not exceed 24 milliseconds.

**Note:** The 24-milliseond restriction applies to each data channel. We suggest that you observe more more than one channel during development to ensure the restriction is complied with across all data channels.

The worst-case duty-cycle conditions can be simulated with the following configuration.

- The longest packet length for the data rate being verified as defined in Table 4.
- The worst-case connection interval that produces the maximum number of packets per connection interval

For transmit power, all roles (advertising, scanning/initiating, *etc.*) should be verified to ensure the limits defined in Table 3 are never exceeded.

# 10 Definitions, Abbreviations, and Acronyms

| Term | Definition |
|---|---|
| API | Application Programming Interface |
| AUS | Australia |
| BLE | Bluetooth Low Energy |
| FCC | Federal Communications Commission |
| GPIO | General Purpose Input/Output |
| IC | Industry Canada |
| LNA | Low Noise Amplifier |
| NZ | New Zealand |
| PA | Power Amplifier |
| PDU | Protocol Data Unit |
| PHY | Physical Layer |
| RF | Radio Frequency |
| SDK | Software Development Kit |
| Softdevice | Bluetooth protocol stack supplied in precompiled binary format by Nordic |
| TX | Transmit |



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610

# 11 Revision History

| Version | Date | Notes | Contributor(s) | Approver |
|---|---|---|---|---|
| 1.0 | 02 December 2019 | Initial Release | Greg Leach | Jonathan Kaye |
| 1.1 | 09 April 2020 | Added details of restrictions to Channel Map.<br>Added further details to Limitation of Transmit Power section.<br>Expanded transmit power example functions section. | Greg Leach | Jonathan Kaye |



**Americas**: +1-800-492-2320
**Europe**: +44-1628-858-940
**Hong Kong**: +852 2923 0610