nRF52840 Errata Attachment

Anomaly 172 Addendum

Bluetooth LE long range co-channel performance

January, 2019

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Life support applications

Nordic Semiconductor's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest dealer, please see www.nordicsemi.com

Main office:

Otto Nielsens veg 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.com

RoHS statement

Nordic Semiconductor's products meet the requirements of Directive 2002/95/EC of the European Parliament and of the Council on the Restriction of Hazardous Substances (RoHS). Complete hazardous substance reports as well as material composition reports for all active Nordic Semiconductor products can be found on our web site www.nordicsemi.com.

Revision history

Date	Version	Description
January 2019	1.0	First version

Contents

Introduction

This document is an attachment to the nRF52840 Errata, ID 172. It explains the product anomaly and the proposed workaround.

Anomaly description

The device fails the co-channel interference test (RF-PHY/RCV/BV-29-C), specifically for the Ble_LR125Kbit and Ble_LR500Kbit modes.

A software workaround is required to pass the test. The workaround is implemented in the S140 SoftDevice v6.1.1 and the DTM example in nRF5 SDK v15.3.0.

Workaround

All C code references (variables, defines, and functions) that are used in the description/pseudo code are available in the Code examples section at the end of this chapter.

Precheck: Verify if a fix for Errata 172 is needed or not

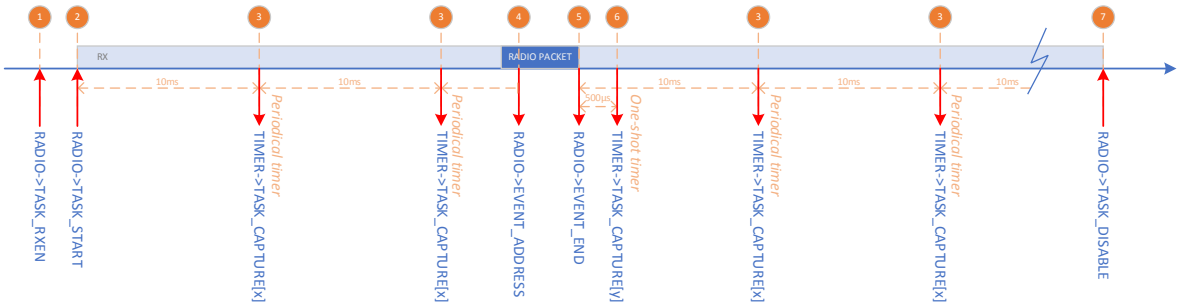
To check if the workaround is needed for a specific chip, call the following function:

```
bool enable_errata_172 (void)
```

If the function returns true, the workaround is needed.

Timing diagram

The following timing diagram illustrates the flow for Bluetooth LE radio mode Ble_LR125Kbit or Ble_LR500Kbit running RX. The orange numbers refer to the pseudo code below.



In the pseudo code, we use one timer with two capture registers. One capture register is used for the 10 ms periodical timer, while the other capture register is used for the 500 µs one-shot timer.

Pseudo code

(1) On RADIO->TASK_RXEN

When enabling Radio in RX mode:

```
set_strict_mode(true);
```

(2) On RADIO->TASK_START

When starting the radio to listen for incoming packets:

```
rssi_val = read_rssi();
strict_mode_on = check_rssi_value(rssi_val);
if (strict_mode_on == false)
   set_strict_mode(false);
clear_start_internal_cntr();

// Configure .CC[x] to give 10 ms periodic time-out
TIMER->CC[x] = <10 ms period>
// Start Timer
TIMER->TASK_CLEAR = 1;
TIMER->EVENT_COMPARE[x] = 0;
TIMER->TASK_START = 1;
```

(3) On TIMER->EVENTS_COMPARE[x]

On periodical timer time-out events:

```
rssi_val = read_rssi();
strict_mode_on = check_rssi_value(rssi_val);
if (m_strict_mode)
{
   if (strict_mode_on == false)
   {
      set_strict_mode(false);
   }
}
else
{
   too_many_detects = check_counter_values();
   if (too_many_detects || strict_mode_on)
   {
      set_strict_mode(true);
   }
}
clear_start_internal_cntr();
// Clear timer for next 10 ms period
TIMER->EVENT_COMPARE[x] = 0;
TIMER->TASK_CLEAR = 1;
```

(4) On RADIO->EVENT_ADDRESS:

Radio event on address match (address received):

```
// Stop Timer (10ms periodical timer)
TIMER->TASK_STOP = 1;
```

(5) On RADIO->EVENT_END

Radio event on packet received:

```
// Configure .CC[x] to give 10 ms periodic time-out
TIMER->CC[x] = <10 ms period>
// Configure .CC[y] to give 500 µs one-shot time-out
TIMER->CC[y] = <500 µs period>
// Start Timer
TIMER->EVENT_COMPARE[x] = 0;
TIMER->EVENT_COMPARE[y] = 0;
TIMER->TASK_CLEAR = 1;
TIMER->TASK_START = 1;
```

(6) On TIMER->EVENTS_COMPARE[y]

On one-shot timer event:

```
rssi_val = read_rssi();
strict_mode_on = check_rssi_value(rssi_val);
if (m_strict_mode == true)
{
   if (strict_mode_on == false)
   {
      set_strict_mode(false);
   }
}
else
{
   if (strict_mode_on == true)
   {
      set_strict_mode(true);
   }
}
clear_start_internal_cntr();
// Disable more events from the one-shot timer
TIMER->EVENT_COMPARE[y] = 0;
TIMER->CC[y] = 0;
```

(7) On RADIO->TASKS_DISABLE

Disabling the radio:

```
set_strict_mode(false);
```

Code examples

Here is a list of variables, defines, and functions used in the pseudo code above:

```
// ----------------------------------------------------------------------
// Global variable reflecting Strict Mode status
bool   m_strict_mode;


// ----------------------------------------------------------------------
// Check if workaround for nRF52840 Anomaly 172 should be enabled or not.
bool   enable_anomaly_172 (void)
{
   bool   retval = false;
   if ((*(volatile uint32_t *)0x40001788) == 0)
   {
      retval = true;
   }
   return(retval);
}

// ----------------------------------------------------------------------
// Strict mode setting will be used only by devices affected by nRF52840 Anomaly
172
void set_strict_mode (bool enable)
{
   uint8_t   dbcCorrTh;
   uint8_t   dsssMinPeakCount;

   if (enable == true)
   {
      dbcCorrTh = 0x7d;
      dsssMinPeakCount = 6;
      *(volatile uint32_t *) 0x4000173c = ((*((volatile uint32_t *) 0x4000173c))
& 0x7FFFFF00) | 0x80000000 | (((uint32_t)(dbcCorrTh)) << 0);
      *(volatile uint32_t *) 0x4000177c = ((*((volatile uint32_t *) 0x4000177c))
& 0x7FFFFF8F) | 0x80000000 | ((((uint32_t)dsssMinPeakCount) & 0x00000007) << 4);
   }
   else
   {
      *(volatile uint32_t *) 0x4000173c = 0x40003034;
      *(volatile uint32_t *) 0x4000177c = ((*((volatile uint32_t *) 0x4000177c))
& 0x7FFFFFFF);
   }
   m_strict_mode = enable;
}

// ----------------------------------------------------------------------
// nRF52840 Anomaly 172 - Check RSSI value against threshold value
bool check_rssi_value (uint8_t rssi_value)
{
   bool   retval = false;
```

```
   if (rssi_value < 95)
   {
      retval = true
   }
   return(retval);
}


// ----------------------------------------------------------------------
// Clear and restart internal counters
#define clear_start_internal_cntr() \
   do{  \
      *(volatile uint32_t *) 0x40001040 = 1;  \
      *(volatile uint32_t *) 0x40001038 = 1;  \
   }
   while (0)


// ----------------------------------------------------------------------
// Initiate and read RSSI value
uint8_t read_rssi(void)
{
   uint8_t   rssi_value;

   NRF_RADIO->EVENTS_RSSIEND = 0;
   NRF_RADIO->TASKS_RSSISTART = 1;
   while (NRF_RADIO->EVENTS_RSSIEND == 0);
   rssi_value = NRF_RADIO->RSSISAMPLE;
   return(rssi_value);
}


// ----------------------------------------------------------------------
// nRF52840 Anomaly 172 - read out and verify internal counter values
bool check_counter_values (void)
{
   uint32_t   counter_values;
   uint16_t   detect_cnt;
   uint16_t   addr_cnt;
   bool     retval = false;

   counter_values = *(volatile uint32_t *) 0x40001574;
   detect_cnt = counter_values & 0xffff;
   addr_cnt = (counter_values >> 16) & 0xffff;
   if ( (detect_cnt > 15) && (addr_cnt < 2))
   {
      retval = true;
   }
   return(retval);
}
```