Migrating from the nRF51 Series to the nRF52 Series

v1.0.0

2015-07-09

Contents

Chapter 1
Migrating from the nRF51 Series to the nRF52 Series

To make the migration process from the nRF51 Series to the nRF52 Series as straightforward as possible we have documented here important differences between the series that you will need to consider during the migration, including highlighting particular areas to consider when migrating code from the nRF51 Series to the nRF52 Series.

Chapter 2
Functional changes

This section lists new and changed features of the Cortex®-M4F processor.

The Cortex®-M0 processor in the nRF51 Series has been replaced with Cortex®-M4F in the nRF52 Series. The differences between the Cortex®-M4F and Cortex®-M0 processors are not in the scope of this document and thus not described in detail.

While Cortex®-M4F is capable of running code compiled for Cortex®-M0, there are some functional differences in the nRF52 Series to pay attention to when porting code from the nRF51 Series. These are described below:

Interrupt event clearing

Clearing an interrupt event may not take effect immediately.

As Cortex®-M4F has a write buffer on the system bus, write accesses to peripherals/SRAM may be delayed. In addition, there can be other delays in the bus infrastructure, such as bridges, that may result in writes occurring after the store instruction has been completed.

To make sure the interrupt line is cleared before exiting an ISR, read back the given event register before returning. Any read on the system bus will require the write buffer to be emptied first.

Existing code:

```
void TIMER0_IRQHandler(void)
{
  NRF_TIMER0->EVENTS_COMPARE[0] = 0;
}
```

Required code:

```
void TIMER0_IRQHandler(void)
{
  NRF_TIMER0->EVENTS_COMPARE[0] = 0;
  (void)NRF_TIMER0->EVENTS_COMPARE[0];
}
```

Interrupt wakeup time

Waking up from an interrupt takes additional cycles compared to the nRF51 Series. This is due to increased flash startup time and increased startup time for the regulators and oscillators. See $t_{IDLE2CPU}$ in the nRF52 Product Specification for the wakeup time.

If the wakeup time from sleep is important, the interrupt vector table can be moved into RAM. This is done by copying the vector table to a suitable RAM location and pointing to it in Cortex®-M4F's Vector Table Offset Register (VTOR). The start of the ISR should also be implemented in RAM to avoid flash latency. The flash will power up in the background when the CPU is running from RAM, so a partial RAM/Flash ISR implementation can improve the interrupt latency without using much RAM.

> **Important:** This feature may not be available if you are using a SoftDevice. Check the SoftDevice Specification.

Unaligned read/write

Cortex®-M4F allows unaligned reads/writes, but on Cortex®-M0 this leads to a HardFault.

In practice Cortex®-M4F splits the read or write into multiple bus accesses. This means that 32-bit and 16-bit reads or writes will not be atomic bus accesses when reading or writing to 32-bit and 16-bit unaligned addresses, respectively. Unaligned accesses are useful to speed up accesses to packed data structures.

Cortex®-M4F can generate a UsageFault on unaligned accesses by setting the `UNALIGN_TRP` bit in the Configuration Control Register (CCR) register. Compilers typically allow you to enable or disable the generation of unaligned memory access instructions.

Interrupt call stack alignment

In Cortex®-M4F the default stack alignment is 4 bytes, while in Cortex®-M0 it is 8 bytes. The ARM Application Binary Interface (ABI) requires 8-byte alignment, so when executing an interrupt, the stack pointer may not be ABI compliant.

The stack alignment can be forced to be 8 bytes by setting the `STKALIGN` bit of the Configuration Control Register (CCR).

For more information, see section Handling an exception in the ARM Compiler - Software Development Guide.

Call stack with FPU

Cortex®-M4F in the nRF52 Series includes the Floating Point Unit (FPU). Some of the registers introduced by the FPU are preserved on the call stack when executing an interrupts service routine. As more registers are stacked when the FPU is enabled, the interrupt latency and stack size increase.

In order to minimize the stack usage, the Lazy Stacking option can be enabled. This will optimize the call stack usage when the FPU is enabled. More information can be found in the ARM application note Cortex-M4(F) Lazy Stacking and Context Switching.

Code RAM region in the memory map

The memory map for the nRF52 Series introduces a Code RAM region located in Cortex®-M4F's Code segment at address 0x00800000. The Code RAM region is a direct mapping of the Data RAM region, which is located in the SRAM segment of Cortex®-M4F. This is done because running code from the Code segment is more effective than from the SRAM segment. It is up to the programmer to make sure that the physical program code and RAM sections do not overlap.

System vector table size

Cortex®-M4F allows up to 240 interrupts defined by implementation, and 16 predefined interrupts or reserved slots. These 256 potential interrupts each have a 4-byte vector table entry, which means that the total vector table size of Cortex®-M4F can be up to 1024 bytes (address 0x400). This size will vary depending on the number of implemented interrupts, but it is good practice to avoid having any static data, such as FW information, before address 0x400.

Additional fault exceptions

In Cortex®-M0, there is only one fault interrupt vector (HardFault), while in Cortex®-M4F there are three extra fault handlers, MemManage, BusFault, and UsageFault. In addition, these handlers have status registers that provide additional information about what caused the fault. This means that what used to cause a HardFault in the nRF51 Series may cause a different type of exception in the nRF52 Series. For more information about the additional exceptions, see the Cortex™-M4 Devices Generic User Guide.

Flash page size

Flash page size has changed from 1 kB in the nRF51 Series to 4 kB in the nRF52 Series. This has implications when erasing flash content, as you can only delete on a page by page basis. It is recommended to use the FICR CODEPAGESIZE and CODESIZE registers to make portable code.

Protect all mechanism

In the nRF52 Series, the UICR RBPCONF register used for read-back protection has been removed. It has been replaced with the PALL field in UICR's APPROTECT; this can disable all read/write accesses to all CPU registers and memory mapped addresses (all accesses through the AHB-AP debug component).

RAM power and retention

In the nRF52 Series, the RAMON and RAMONB registers are deprecated. The RAM[x] registers in the POWER peripheral replace this functionality. Individual RAM sections can still be switched on or off in System-On mode, and retention can be switched on or off in System-Off mode. In addition, the FICR's NUMRAMBLOCK and SIZERAMBLOCKS registers are removed. RAM configuration can be derived from the Cortex®-M4F ROM table part number, as it is static for a given part number.

nRF51 Series Memory Protection Unit replaced

The two-region MPU functionality in the nRF51 Series has been removed from the nRF52 Series. This should not be confused with the Cortex®-M4Fs MPU. In the nRF52 Series the same functionality can be achieved with the memory watch unit (MWU), which is a more generic implementation. In addition, the block protect mechanism in the nRF51 Series (PROTEN registers) has been renamed as BPROT in the nRF52 Series.

NVM timing

The time it takes to do a page erase, mass erase and to write to non-volatile memory (NVM) has changed in the nRF52 Series. Please look up the new values in the electrical parameters section of the NVMC chapter in the nRF52 Product Specification.

Analog to Digital Converter (ADC) redesign

The ADC peripheral has been redesigned and is not compatible with the nRF51 Series ADC.

Chapter 3
New features

The following table provides a brief overview of the new features of the nRF52 Series devices. For more detailed information about each feature, please see the nRF52832 Product Specification.

Component	Change description
Cortex®-M4F	The Cortex®-M4F processor has been implemented with the following options: • Memory Protection Unit (MPU) • Floating Point Unit (FPU) • SysTick Timer • Priority bits: 3 • Endian: Little Endian • Interrupts: One per peripheral ID

Debugging	Cortex®-M4F has been implemented with full debug and trace support:
	• Debug Access Port (DAP)
	• Flash Patch and Breakpoint Unit (FPB) supports:
	  • Two literal/data comparators
	  • Six instruction comparators
	• Data Watchpoint and Trace Unit (DWT)
	  • 4 comparators
	• Instrumentation Trace Macrocell (ITM)
	• Embedded Trace Macrocell (ETM)
	• Trace Port Interface Unit (TPIU)
	  • 5-pin parallel trace
	  • Single pin serial trace (SWO)
	Trace debug functionality is available in the nRF52 Series, which requires additional pins. The register TRACECONFIG of the CLOCK peripheral can be used to set the multiplexing function of the trace pins along with the trace clock speed. For details on how to use the debug and trace capabilities, please read the debug documentation of your IDE.

CLOCK	In the nRF52 Series, the high frequency clock (HFCLK) is sourced from either an internal oscillator, or a crystal oscillator controlled by an external 32 MHz crystal. The CPU runs at 64 MHz, while the peripheral clocks run at lower frequencies; the TIMER, for example, uses a 16 MHz clock as in the nRF51 Series. The crystal oscillator must be used as the HFCLK source when running the radio or calibrating the 32.768 kHz RC oscillator. The crystal oscillator on nRF52 is started the same way as on the nRF51 Series, using the HFCLKSTART task.
Block protect	The Block protect (BPROT) partly replaces the memory protection unit (MPU) in nRF51. The PERR0 and RLENR0 registers that provided write protection of peripherals and RAM are removed in the nRF52 Series in which there is no two-region protection scheme as in the nRF51 Series, but the same functionality is available by configuring the new memory watch unit (MWU) peripheral.
RADIO	New features: • TXPOWER modes configurable from +4 dBm to -40 dBm.

Component	Change description
	• Support for *Bluetooth*® low energy packet length extension. • Fast radio ramp-up time (40 µs).
NFCT	Near Field Communication Tag (NFCT). This is a new peripheral introduced in the nRF52 Series. The NFC Tag supports the ISO/IEC 14443-2 PICC device with communication signal interface type A and 106 kbps bit rate.
UART	An additional UART peripheral (UARTE) is added. This new peripheral has EasyDMA support for CPU offloading.
SPI	There are three peripheral IDs for serial peripheral interfaces, each capable of acting as SPIS, SPIM or SPI. EasyDMA support is added to SPIS/SPIM for CPU offloading.
TWI	There are two peripheral IDs for two-wire interfaces, each capable of acting as TWIS, TWIM or TWI. EasyDMA supports is added to TWIS and TWIM for CPU offloading.
COMP	The General purpose comparator is a new feature in the nRF52 Series, and can work in a differential or single-ended mode. The COMP and LPCOMP peripherals are mutually exclusive and cannot be used together.
ADC	The ADC peripheral is replaced by a differential Successive Approximation Register Analog to Digital Converter (SAADC). The interface is not backwards compatible, but has added features, such as support for up to 4 differential inputs and EasyDMA for CPU offloading.
TIMERn	Additional timer instances are added, TIMER3 and TIMER4. All timers now support all bit modes (8, 16, 24, and 32 bits).
RTC	An additional RTC instance is added, RTC2.
GPIOTE	The GPIO Tasks and Events (GPIOTE) peripheral changes: • Number of tasks/events increased from 4 to 8. • Set and Clear tasks have been added in addition to the existing Out task.
PPI	The number of user-configurable PPI channels has increased from 16 to 20. The number of PPI groups has increased from 4 to 6. A new type of endpoint called FORK can be used to trigger an additional task from a single event endpoint.
PDM	The Pulse Density Modulation (PDM) peripheral is new in the nRF52 Series. This peripheral is designed so that it can filter two PDM signals and output it in PCM format (e.g. stereo sound). It supports EasyDMA for CPU offloading and can handle continuous audio streaming.
I2S	The Inter-IC Sound (I2S) peripheral is a new feature in the nRF52 Series. This peripheral is designed to interface with external audio circuitry, such as a codec or DSP. Most common derived formats are supported, including the original two-channel I2S format. EasyDMA is available for this peripheral to allow controlling the I2S slave from a low priority execution context.
MWU	The Memory Watch Unit (MWU) peripheral is a new feature in the nRF52 Series. This peripheral is designed to generate events when detecting accesses to SRAM or peripherals' memory segments. The events can be configured to trigger regular interrupts or a non-maskable interrupt (NMI). This feature may be used for run time protection of SRAM regions and peripherals, and help detect heap and stack overflows.

Component	Change description
EGU	The Event Generator Unit (EGU) is a new feature in the nRF52 Series, replacing the software interrupts (SWIs). This peripheral can be used to: • Trigger hardware events from firmware. • Trigger hardware events at a different IRQ priority than the originating peripheral (or firmware ISR). • Trigger hardware tasks originating from either hardware or firmware.
PWM	The Pulse Width Modulation (PWM) peripheral is a new feature in the nRF52 Series. There are three instances of the PWM in the nRF52 Series. Each instance features up to four channels and has EasyDMA capabilities for CPU offloading.
NVMC	A 2 kB direct mapped cache is a new feature in the nRF52 Series. The cache is disabled by default, but can be enabled through the NVMC ICACHECNF register. Cache profiling can be enabled in ICACHECNF, which will update cache profiling registers IHIT and IMISS. These registers increment on a cache hit or miss and can help in optimizing code for best performance.
UICR	New UICR configuration options in the nRF52 Series: • PSELRESET - Programmable reset pin, the pin can be multiplexed to either GPIO or RESET. • APPROTECT - Read-back protection mechanism. Removed UICR configuration options in the nRF52 Series: • RBPCONF - Replaced by APPROTECT. • CLENR0 - Because the nRF51 style MPU is removed, this has been removed as well. • XTALFREQ - 32 MHz crystal support only. • FWID - Uses generic NRFFW registers.
FICR	Removed FICR configuration options in the nRF52 Series: • CLENR0 - Because the MPU has been removed, this has been removed as well. • PPFC - Replaced by APPROTECT in UICR. • OVERRIDEEN - Not needed. • NRF_1MBIT[x] - Not needed. • BLE_1MBIT[x] - Not needed.

Chapter 4
Migrating an nRF51 BLE project to nRF52

This section describes the software changes that are required when migrating a BLE application from the nRF51 Series to the nRF52 Series. The document assumes that you already have a BLE application running on the nRF51 Series. If you are starting a new development based on the nRF52 Series, please have a look at the nRF52 Series documentation and start with the development kits, SoftDevice and SDK based on the nRF52 Series directly.

Toolchain upgrades

Upgrade every component of the toolchain to the latest version in order to make the toolchain compatible with the nRF52 Series. The latest versions are also backward compatible with the nRF51 Series. This way you can develop an application in parallel for the nRF51 Series and the nRF52 Series with the latest components in the toolchain. The following toolchain components are required for the nRF52 Series:

- nRF5x Tools 7.5.1 or later
- nRFgo Studio 1.20 or later
- MDK 8.0.3 or later
- Master Control Panel 3.9.0 or later

Migrating to BLE S132 SoftDevice

The new S132 SoftDevice developed for the nRF52 Series is based on the S130 SoftDevice for the nRF51 Series. It supports both peripheral and central role concurrently. The API for S132 v1.0.0-3.alpha is aligned with the APIs for S110 v8.x.x, S120 v2.x.x and S130 v1.0.0. If you have an application running on S110 v8.x.x, S120 v2.x.x or S130 v1.x.x, the application should work with the S132 v1.0.0-3.alpha by adjusting the different memory settings as shown in Figure 1: ROM and RAM settings for the S132 v1.0.0-3.alpha SoftDevice on page 11. Newer S132 SoftDevices may have different memory settings and/or unaligned API compared with S132 v1.0.0-3.alpha. Refer to the release notes, migration document and SoftDevice Specification for details on any newer version of the S132 SoftDevice version. It is important that you recompile your application with the header files released with the S132 SoftDevice you are using.

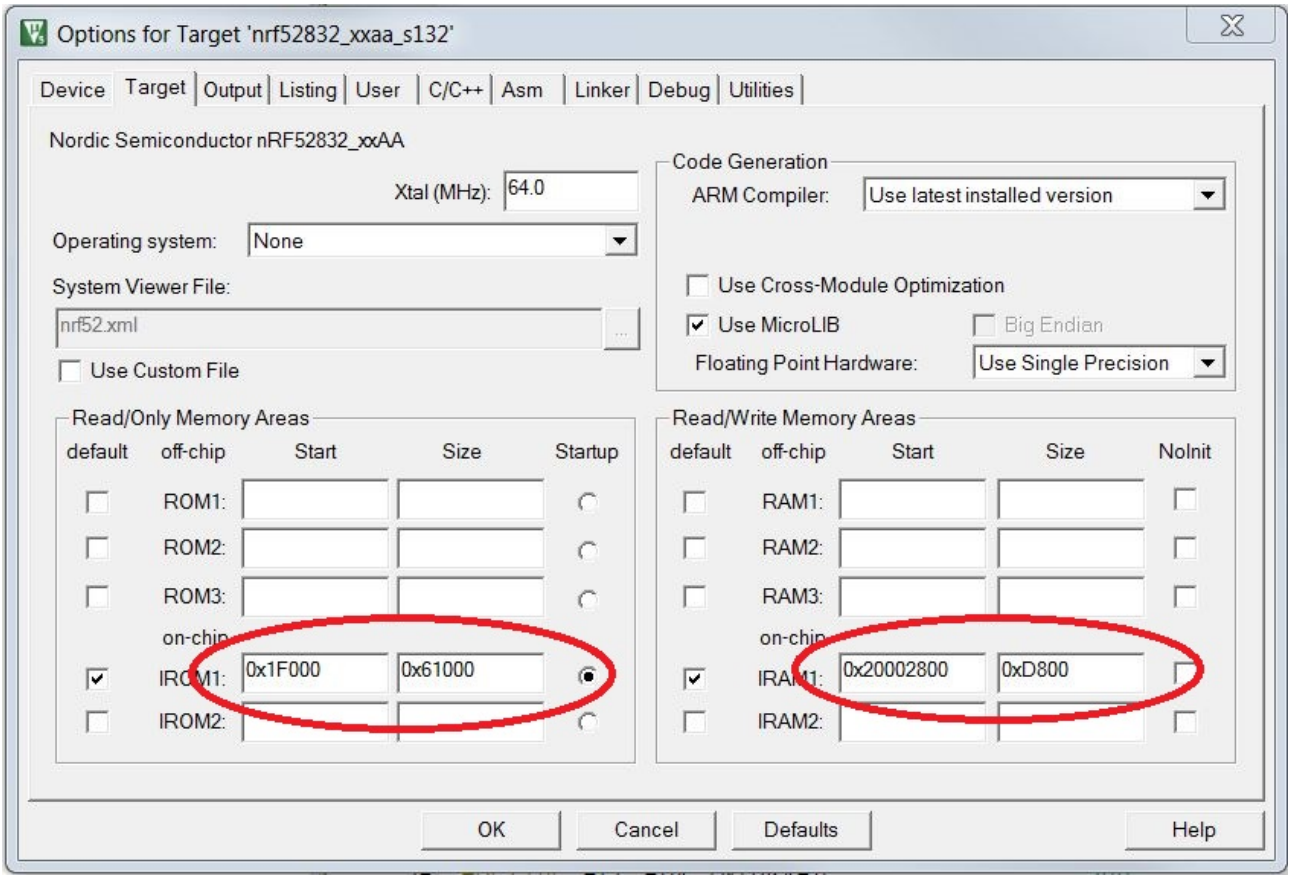

Figure 1: ROM and RAM settings for the S132 v1.0.0-3.alpha SoftDevice

> **Important:** For ROM and RAM settings of other S132 versions than 1.0.0-3.alpha, look at the SoftDevice Specification for that specific S132 version.

Porting a BLE example application from nRF51 SDK the nRF52 Series in Keil

This section describes the steps needed to port the `ble_app_hrs` example application in nRF51 SDK 8.1.0 to run on nRF52. A customer with a custom application that currently compiles and runs with nRF51 SDK 8.1.0 may find the steps helpful in porting their application to the nRF52 Series. A custom application may, however, need to follow only some of the steps mentioned in this example, or require additional steps, depending on what libraries and drivers are used. The following steps are tailored for Keil IDE but may also apply for developers using other toolchains for nRF5x development.

The following steps describe the changes needed to port the `ble_app_hrs` example from nRF51 SDK 8.1.0 to nRF52 SDK 0.9.0. If you are starting a new development based on the nRF52 Series, use the template examples provided by the nRF52 SDK, for example `ble_app_hrs`. This port is chosen here because the nRF52 SDK 0.9.0 is a close successor to the nRF51 SDK 8.1.0, so only minimal changes are required. If you have a more recent version of the nRF52 SDK, you may want to become familiar with the release notes for that specific nRF52 SDK version and add any potential steps to accommodate for changes made from nRF52 SDK 0.9.0. Similarly, if you have an nRF51 SDK version older than 8.1.0, see the nRF51 SDK release notes that represent the changes going from that nRF51 SDK version to nRF51 SDK 8.1.0.

The porting method chosen here is to let the `ble_app_hrs` example reside inside nRF51 SDK 8.1.0 and make changes to it there in order to port it to the nRF52 Series. You should create a separate copy of nRF51 SDK 8.1.0 to use with the nRF52 Series:

1. Install DeviceFamilyPack v8.0.3 or newer, and ARM-CMSIS pack v4.3.0 or newer, as shown in Figure 2: Selecting packs in Keil on page 12.

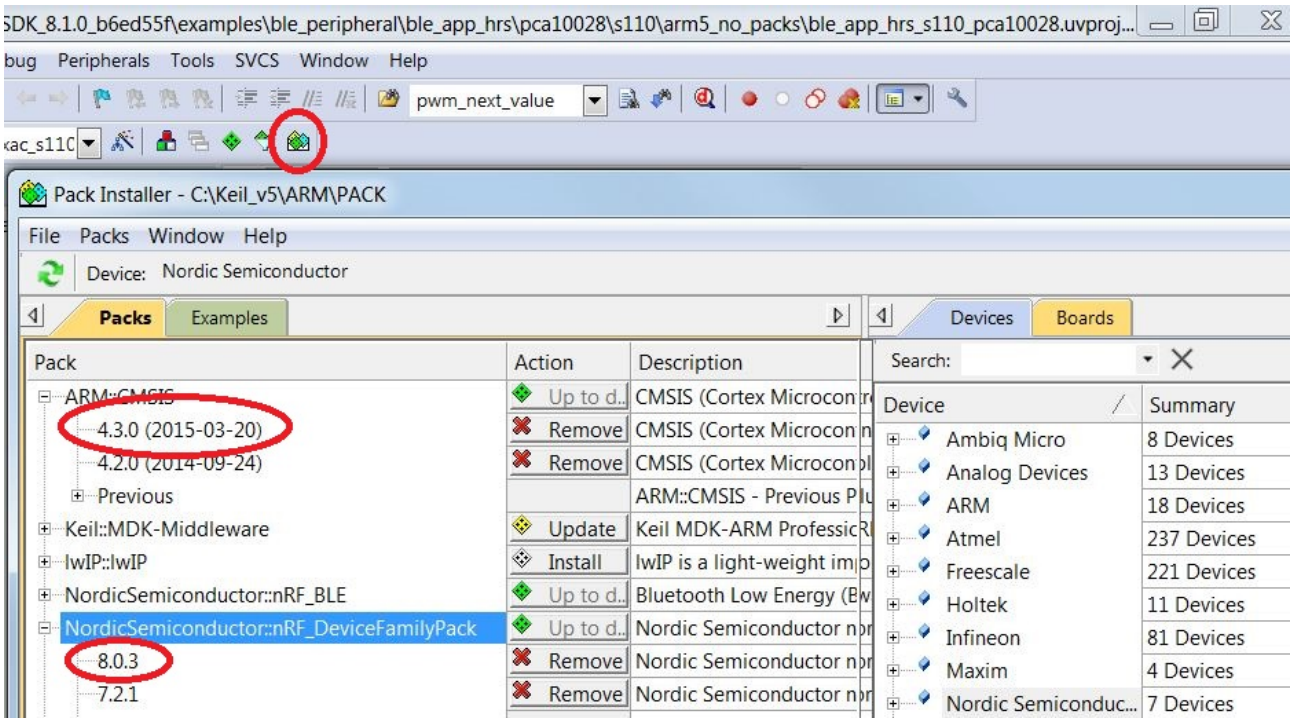

Figure 2: Selecting packs in Keil

2. Open the `\ble_app_hrs\pca10028\S110\arm5_no_packs\ble_app_hrs_s110_pca10028.uvprojx` project, and proceed to step 4.
3. (Optional) Port uVision4 Multi-Project by opening `\ble_app_hrs\ble_app_hrs.uvmpw`. This step is not needed if you use the uVision5 project in step 2.

   a. Remove the nRF51 startup files, that is `arm_startup_nrf51.s` and `system_nrf51.c`.
   b. Copy the nRF52 startup files from nRF52 SDK 0.9.0 into the project. Include the nRF52 startup files `arm_startup_nrf52.s` and `system_nrf52.c` in the project, as shown in Figure 3: Replacing nRF51 Series startup files with nRF52 Series startup files on page 13.

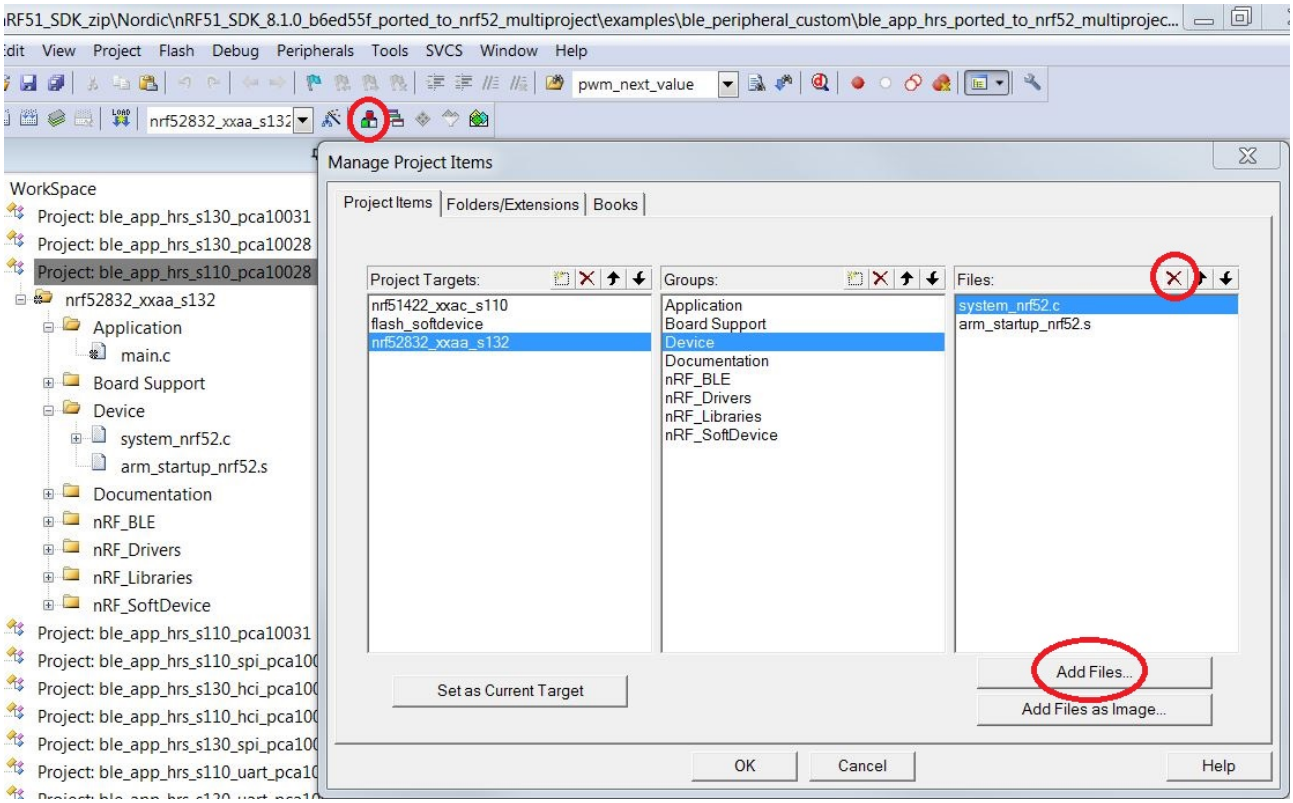

Figure 3: Replacing nRF51 Series startup files with nRF52 Series startup files

c. Select **File > Device Database > Nordic nRF52 Series Devices** to set the floating point unit in the nRF52 Device Database, as shown in Figure 4: Setting the floating point unit on page 13.

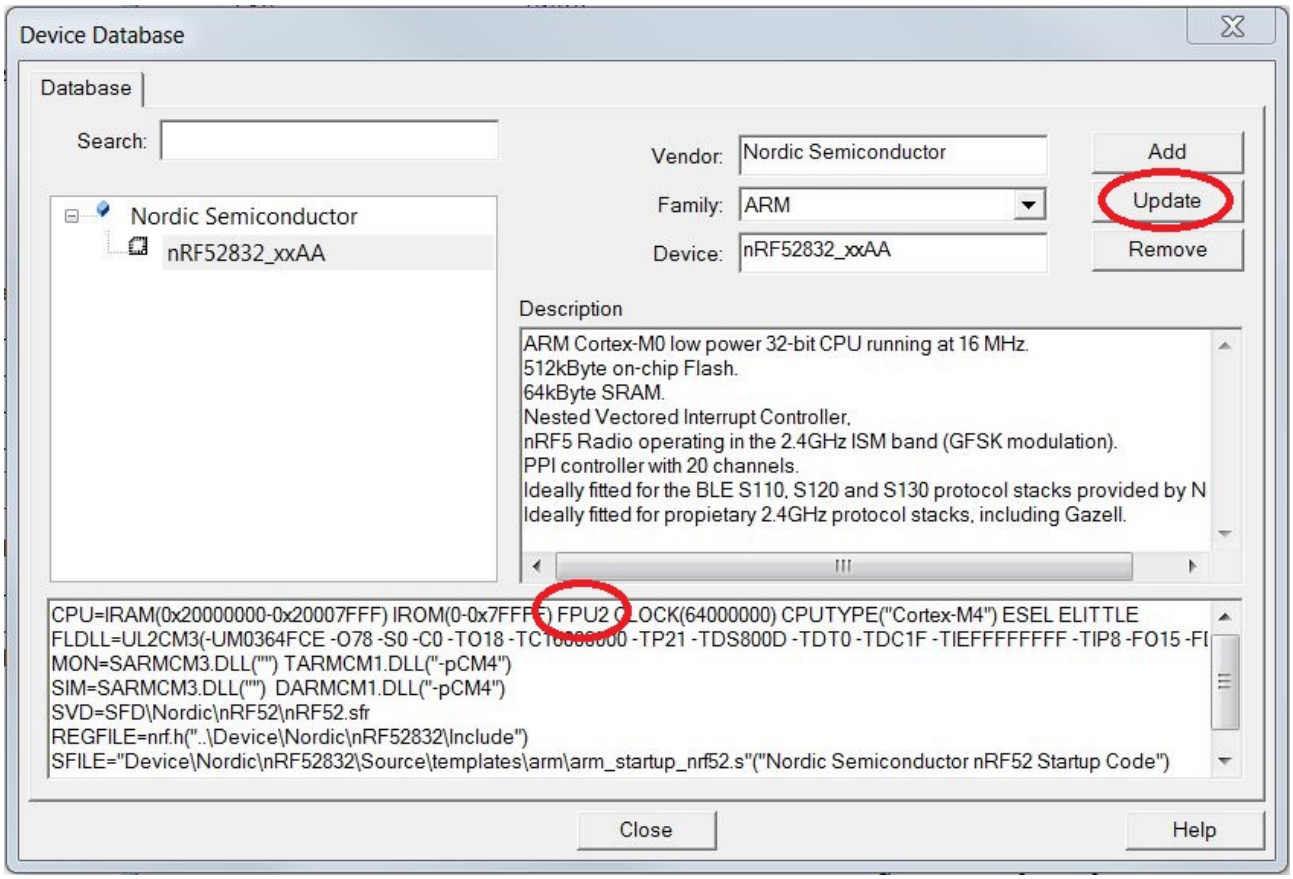

Figure 4: Setting the floating point unit

4. Select **Options for Target** > **Device** tab to set the nRF52832_xxAA device as the target.
5. Configure IROM an IRAM settings for the S132 SoftDevice, as shown in Figure 1: ROM and RAM settings for the S132 v1.0.0-3.alpha SoftDevice on page 11.
6. Overwrite the following components in nRF51 SDK 8.1.0 with components from nRF52 SDK v0.9.0:
   a. `\components\device\`
   b. `\examples\bsp\`
   c. `\components\drivers_nrf\pstorage\`
   d. `\examples\ble_peripheral\ble_app_hrs\config\pstorage_platform.h`
7. Download the S132 SoftDevice and copy the header files into the `\nRF51_SDK_8.1.0_ b6ed55f \components\softdevice` folder.
8. Remove the reference to S110 specific headers in `\components\softdevice\s110\headers`, as shown in Figure 5: Removing the reference to S110 specific headers on page 14.

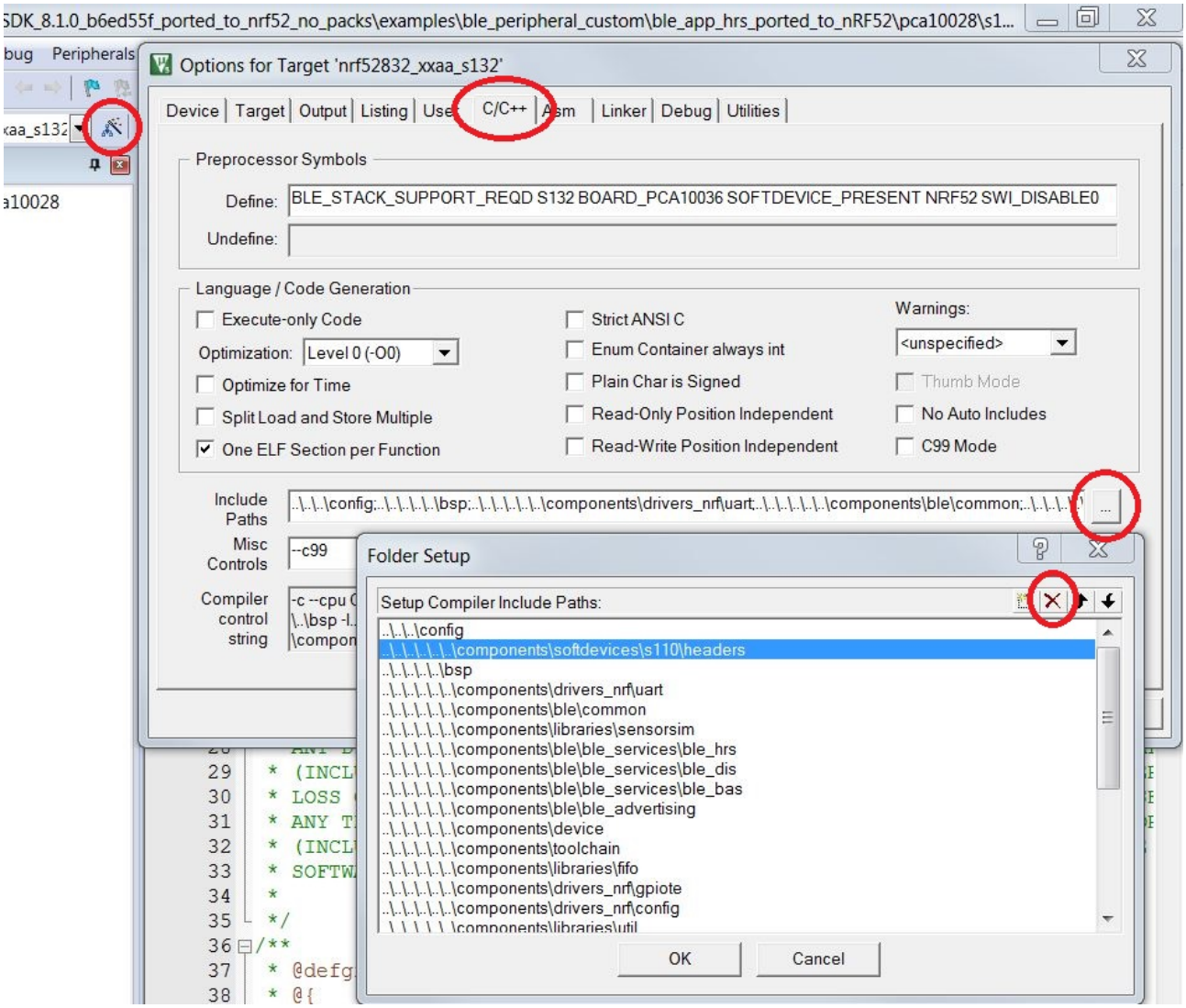

Figure 5: Removing the reference to S110 specific headers

9. Add the reference to S132 headers at `\components\softdevice\s132\include` and `\components\softdevice\s132\include\nrf52`.
10. Update the preprocessor symbols as follows, as shown in Figure 6: Updating the preprocessor symbols on page 15:
    a. NRF51 to NRF52
    b. PCA10028 to PCA10036
    c. S110 to S132

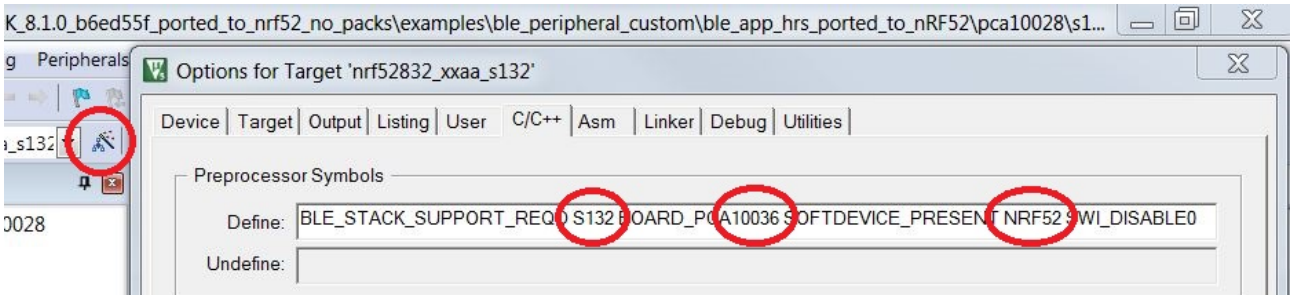

Figure 6: Updating the preprocessor symbols

11. Replace all occurrences of `nRF51.h` and `nrf51_bitfields.h` includes (and any other nRF51 specific includes) with `nrf.h`. You can use **Ctrl+Shift+F** to find the occurrences in all files.
12. Select **Options for Target** > **Debug** tab to set **J-Link / J-Trace Cortex** as the programming/debug interface.
13. Select **Options for Target** > **Utilities** > **Settings** > **Flash Download** tab to specify which **Flashing algorithm** to use. Here you must remove **nRF51xxx** and add both **nRF52xxx** and **nRF51xxx_IUCR**. The last one is needed if you want to upload content to the UICR.
14. In the `main.c` file, in its `ble_stack_init` function, make sure that the SoftDevice is initialized by adding a new test for the new preprocessor symbol S132. This has to take place at the same location as you have for the S130 check.

```
#if defined(S110) || defined(S130) || defined(S132)
    // Enable BLE stack.
    ble_enable_params_t ble_enable_params;
    memset(&ble_enable_params, 0, sizeof(ble_enable_params));
#if defined(S130) || defined(S132)
    ble_enable_params.gatts_enable_params.attr_tab_size =
 BLE_GATTS_ATTR_TAB_SIZE_DEFAULT;
#endif
    ble_enable_params.gatts_enable_params.service_changed =
 IS_SRVC_CHANGED_CHARACT_PRESENT;
    err_code = sd_ble_enable(&ble_enable_params);
    APP_ERROR_CHECK(err_code);
#endif
```

Chapter 5
Performance

On the nRF51 Series, the processor core and the AHB and APB bus-system ran on a 16 MHz clock. On the nRF52 Series, the processor core and AHB system run at 64 MHz, while the APB system runs at 16 MHz. This means that reading from or writing to peripherals on the APB system will take relatively longer time on the nRF52 Series.

However, because Cortex®-M4 uses a write buffer, the core is able to continue executing code that does not access the system bus further. Reordering APB-intensive code to interleave APB accesses with non-memory operations can therefore speed up the code. Read accesses will always stall the processor until the bus transfer is completed, and it will force the write buffer to be emptied before the read is initiated.

Executing code from flash has a wait-state penalty on the nRF52 Series. This is because of the increased processor frequency compared to the nRF51 Series which had zero wait states on executing code from flash. The number of wait-states can be looked up in the product specification, and depends on whether the cache is enabled or not. Wait-states when executing code from flash are mitigated by a direct-mapped cache that can be enabled through the NVMC peripheral. Cortex®-M4F does prefetching, so sequential code executing is capable of running without stalling the pipeline in many cases.