	STMicroelectronics
X-CUBE PRODUCT USER MANUAL	Date: Author:

Porting X-CUBE-BLE1, Expansion for STM32Cube Across Different STM32 Series

Table of Contents

1 References

[1] TBD

2 Acronyms and Abbreviations

Acronym	Description

3 Introduction

This document describes how to migrate the X-CUBE-BLE1 expansion software to different STM32 series. The instructions provided here refer to the specific example of porting the code to a NUCLEO-L152RE board (based on STM32L152RE) using STM32Cube with IAR EWARM. Porting to different series of STM32 MCUs is very similar and only slight changes would be required.

4 What is STM32Cube?

4.1 STM32Cube Overview

STMCube™ initiative was originated by STMicroelectronics to ease developers' life by reducing development efforts, time and cost. STM32Cube covers STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF4 for STM32F4 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

Information about STM32Cube are available on st.com at:
http://www.st.com/stm32cube

4.2 STM32Cube Architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with each other's as described in the figure below:

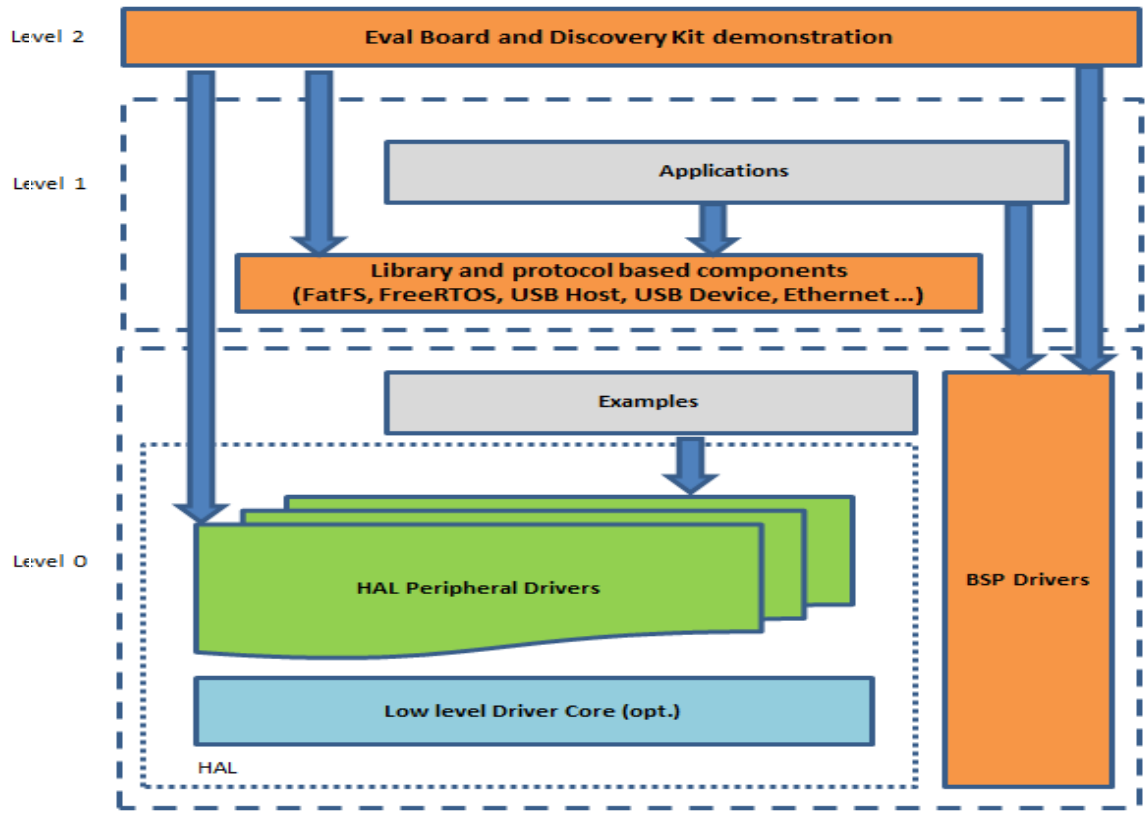


Figure 1 Firmware Architecture

Level 0: This level is divided into three sub-layers:

- **Board Support Package (BSP):** this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…) and composed of two parts:
  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provide specific APIs to the BSP driver external components and could be portable on any other board.
  - BSP driver: it permits to link the component driver to a specific board and provides a set of friendly used APIs. The APIs naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On()

  It's based on modular architecture allowing to port it easily on any hardware by just implementing the low level routines.

- **Hardware Abstraction Layer (*HAL*):** this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides a generic, multi instance and functionalities oriented APIs which permit

to offload the user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART…) it provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication. The HAL Drivers APIs are split in two categories, generic APIs which provides common and generic functions to all the STM32 series and extension APIs which provides specific and customized functions for a specific family or a specific part number.

- **Basic peripheral usage examples**: this layer encloses the examples build over the STM32 peripheral using only the HAL and BSP resources.

Level 1: This level is divided into two sub-layers:

- **Middleware components**: set of Libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interactions between the components of this layer is done directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in the library system call interface. As example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

- **Examples based on the Middleware components**: each Middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several Middleware components are provided as well.

Level 2: This level is composed of a single layer which is global real-time and graphical demonstration based on the Middleware service layer, the low level abstraction layer and the basic peripheral usage applications for board based functionalities.

5 Porting X-CUBE-BLE1 to different STM32 series

5.1 Preliminary Steps

Download the X-CUBE-BLE1 software package and extract its content. From now on, we'll refer to the base directory of the X-CUBE-BLE1 package as "$BLE1_DIR".

We'll provide instructions to port the software to different STM32 series. In this specific example, we'll use another STM32 Nucleo board as the target for our porting. In particular we'll migrate the code to a NUCLEO-L152RE board (based on STM32L152RE) using IAR EWARM.

Since we're interested in STM32L1, search for STM32CubeL1 package on www.st.com and unzip it to a local directory. From now on we'll refer to the base directory of STM32CubeL1 as "$CUBE_DIR".

Start from the SensorDemo project located in

"$BLE1_DIR\Projects\Multi\Applications\SensorDemo" for NUCLEO-F401RE and port it to NUCLEO-L152RE.

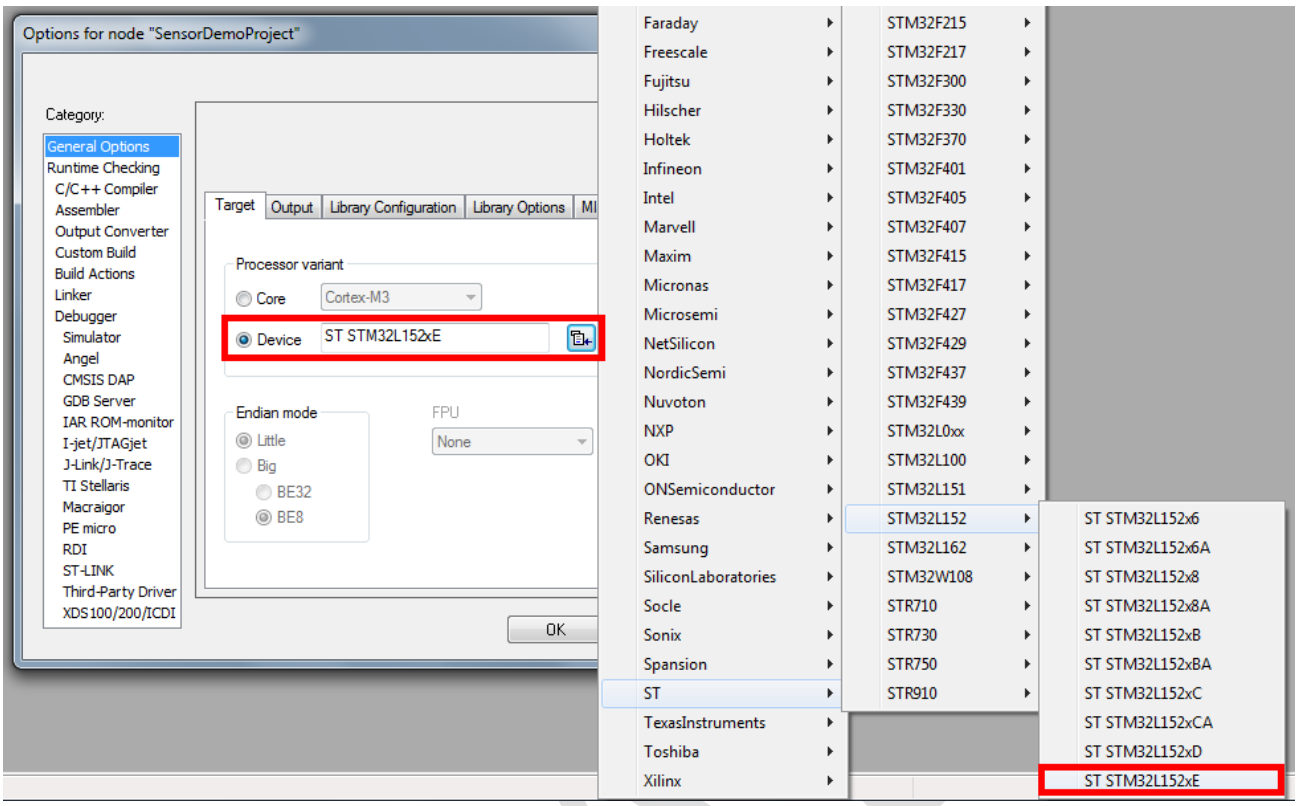
Copy the whole

"$BLE1_DIR\Projects\Multi\Applications\SensorDemo\EWARM\STM32F401RE-Nucleo" directory into "$BLE1_DIR\Projects\Multi\Applications\SensorDemo\EWARM" and within Explorer rename it to STM32L152RE-Nucleo.

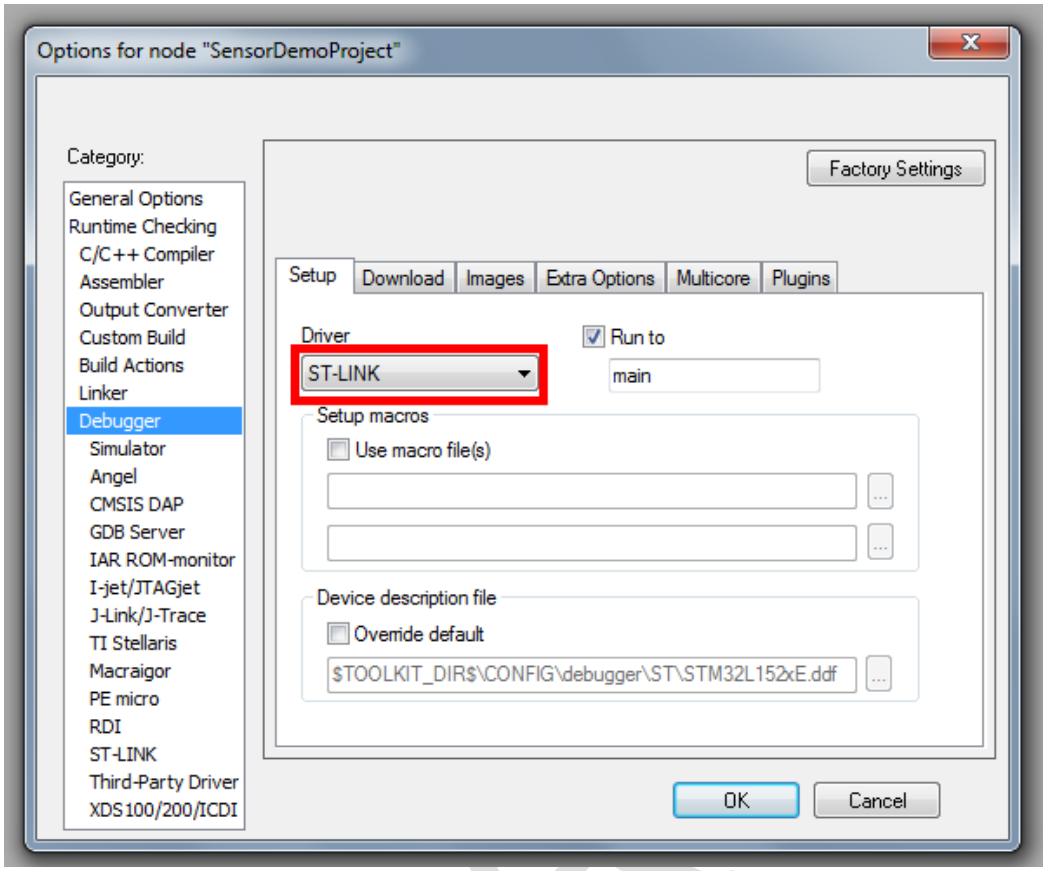
Now go to the

"$BLE1_DIR\Projects\Multi\Applications\SensorDemo\EWARM\STM32L152RE-Nucleo" directory and open the "SensorDemoProject.eww" file using IAR.

As a first step, within IAR select "Project -> Options..." and from the "Target" tab, change the "device" in the "Processor variant" section. In our case, we're going to migrate to the STM32L152RE chip. From the device selector, choose "ST STM32L152xE".



Verify the debugger settings and check that ST-Link is being used.

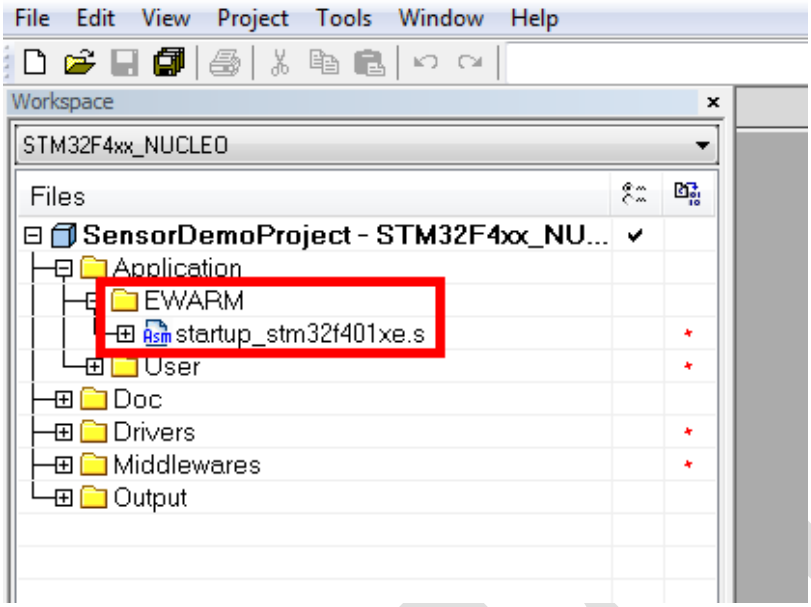


Press "OK" to accept the changes.

Now we are going to replace the specific library and product files.

5.2 Replace the startup file

Now we have to replace the "startup_stm32f401xe.s" startup file for STM32F401RE with one suitable for STM32L152RE. We can find an appropriate startup file in the STM32CubeL1 package.



Copy the

“$CUBE_DIR\Projects\STM32L152RE-Nucleo\Examples\SPI\SPI_FullDuplex_ComIT\

\EWARM\startup_stm32l152xe.s” file into this directory:

“$BLE1_DIR\Projects\Multi\Applications\SensorDemo\EWARM\STM32F401RE-Nucleo”

To add the newly copied file to the project, within IAR right click on the "EWARM" directory and select "Add -> Add Files..." and select the

“$BLE1_DIR\Projects\Multi\Applications\SensorDemo\EWARM\STM32F401RE-Nucleo\

\startup_stm32l152xe.s" file.

Now, within IAR, remove the old startup file for STM32F4 by right clicking on "startup_stm32f401xe.s" and selecting "Remove".

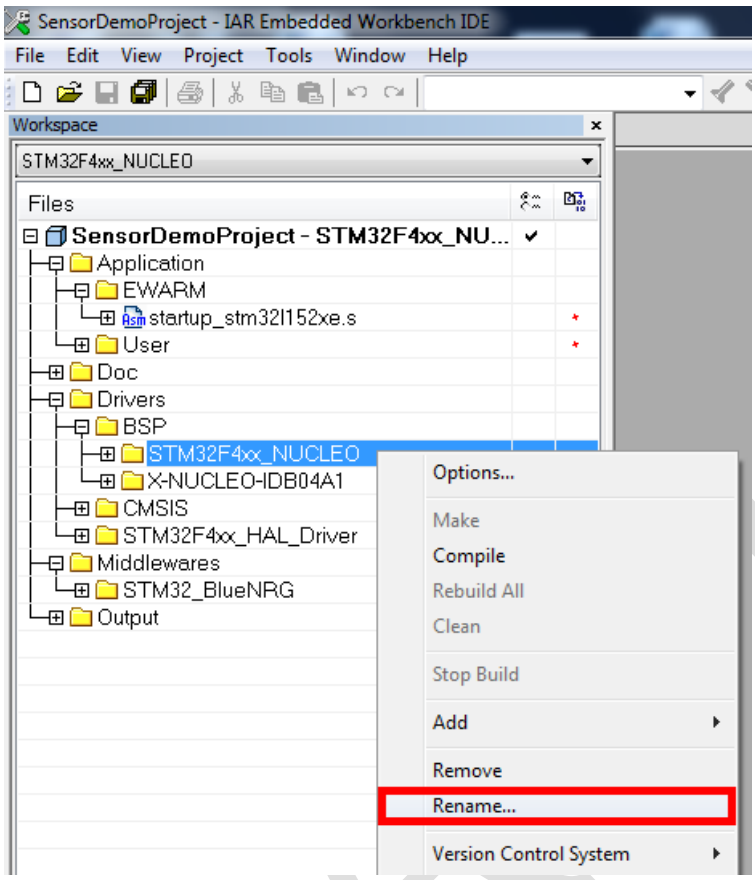
5.3 Replace the STM32 Nucleo board BSP driver

Replace the "stm32f4xx_nucleo.c" BSP driver file for NUCLEO-F401RE with the one specific for NUCLEO-L152RE.

Copy the "$CUBE_DIR\Drivers\BSP\STM32L1xx_Nucleo" directory to

“$BLE1_DIR\Drivers\BSP”.

Now, as shown in the picture, within IAR, right click on the "STM32F4xx_NUCLEO" folder name and select "Rename...". Choose "STM32L1xx_NUCLEO" as the new name.



Right click on the newly renamed "STM32L1xx_NUCLEO" folder and select "Add -> Add Files..." and add the "stm32l1xx_nucleo.c" file (located in

"$BLE1_DIR\Drivers\BSP\STM32L1xx_Nucleo") to the project.

Remove the old driver file for NUCLEO-F401RE by right clicking on "stm32f4xx_nucleo.c" and selecting "Remove".

Now copy the "$BLE1_DIR\Drivers\BSP\STM32F4xx-Nucleo\stm32f4xx_nucleo_bluenrg.h" file into "$BLE1_DIR\Drivers\STM32L1xx_Nucleo" and within Explorer rename the newly copied file to "stm32l1xx_nucleo_bluenrg.h". Now edit the new file and replace the following lines:

```
#include "stm32f4xx_hal.h"
#include "stm32f4xx_nucleo.h"
```

with:

```
#include "stm32l1xx_hal.h"
#include "stm32l1xx_nucleo.h"
```

NOTE: The “stm32l1xx_nucleo_bluenrg.h” file allows you to configure the driver for the BlueNRG chip according to your HW setup. In this case we’re migrating from STM32F401RE to STM32L152RE and there are no changes required: in both cases we’re using an STM32 Nucleo board and the pin configuration is identical for both microcontrollers. When migrating across other STM32 series, however, you may need to update the configuration of the SPI alternate function and the EXTI line for the BlueNRG IRQ.

In particular, you may need to update the definition of “BNRG_SPI_EXTI_IRQn” and “BNRG_SPI_EXTI_IRQHandler” according to the documentation of your specific STM32 chip. To get an idea of the changes that may be needed across STM32 series, please compare the contents of “$BLE1_DIR\Drivers\BSP\STM32F4xx-Nucleo\stm32f4xx_nucleo_bluenrg.h” and “$BLE1_DIR\Drivers\BSP\STM32L0xx-Nucleo\stm32l0xx_nucleo_bluenrg.h”.

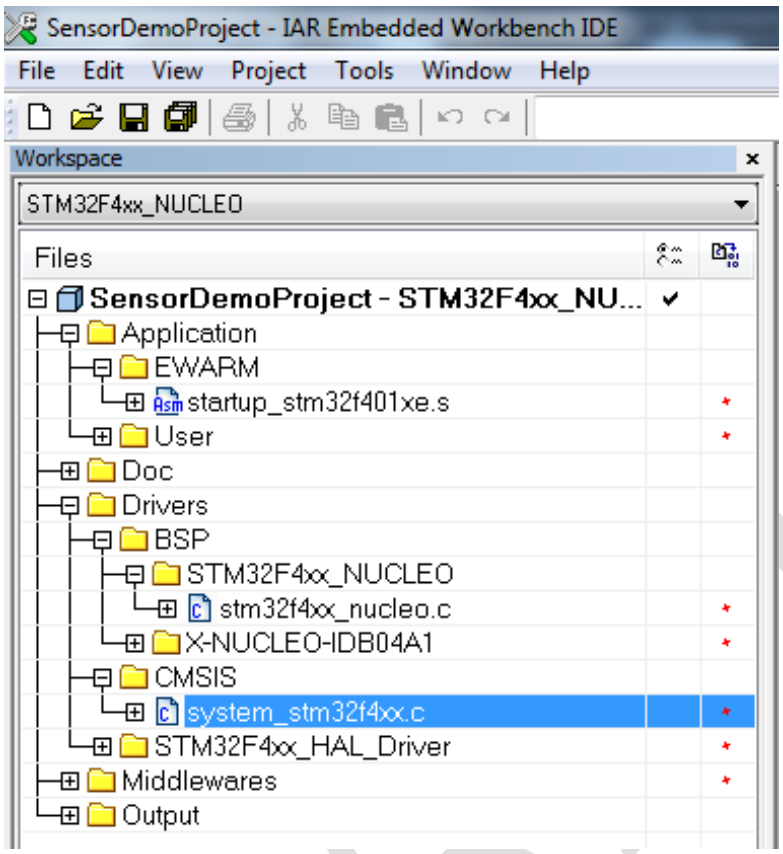
For example, with the STM32L0xx you have to use GPIO_AF0_SPI1 as the alternate function for SPI1 while with STM32F4xx you have to use GPIO_AF5_SPI1.

As for the EXTI line handling, STM32L0xx and STM32F4xx are different because in the first case, EXTI lines 0 and 1 are handled by a single handler, while in the second case, each line is handled individually.

If you need to port your software on a custom board based on STM32, of course, you will have to change the pin configuration according to your schematics.

5.4 Replace the device CMSIS driver

Now we have to replace the device CMSIS driver for STM32F4 with one specific for STM32L1.



Copy the whole

"$CUBE_DIR\Drivers\CMSIS\Device\ST\STM32L1xx" directory to

"$BLE1_DIR\Drivers\CMSIS\Device\ST".

From this directory:

“$CUBE_DIR\Projects\STM32L152RE-Nucleo\Examples\SPI\SPI_FullDuplex_ComIT\Src”

copy file "system_stm32l1xx.c" to:

“$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Src”

To add the newly copied file to the project, within IAR right click on the "CMSIS" directory and select "Add -> Add Files..." and add the "system_stm32l1xx.c" file (located in

"$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Src") to the project.

Now, within IAR, remove the old CMSIS driver file for STM32F4 by right clicking on "system_stm32f4xx.c" and selecting "Remove".

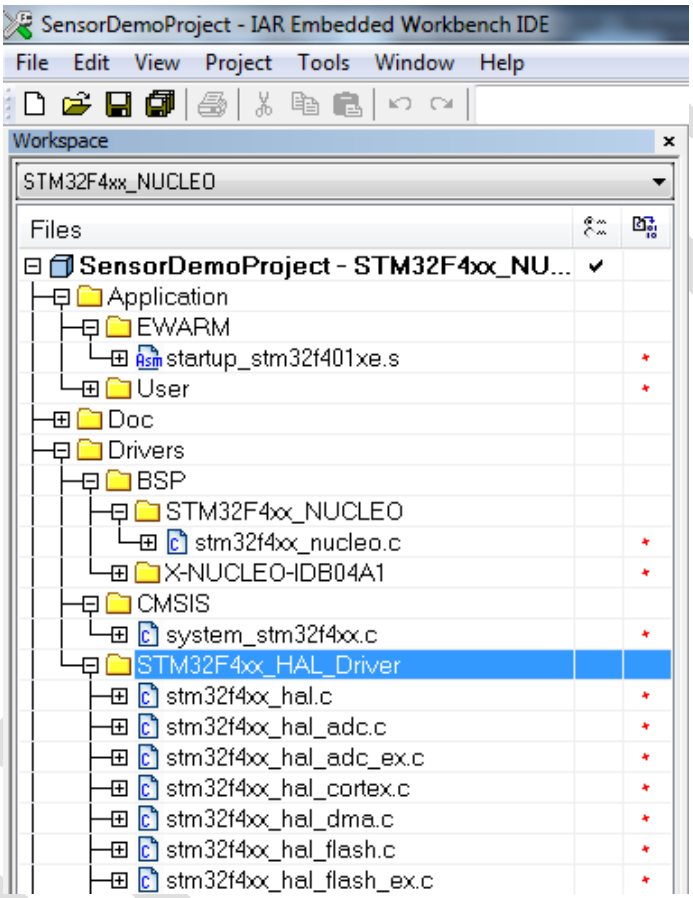
5.5 Replace the device Hardware Abstraction Layer (HAL) files

Replace the device Hardware Abstraction Layer (HAL) files for STM32F4 with the ones for STM32L1.

Copy the "$CUBE_DIR\Drivers\STM32L1xx_HAL_Driver" directory to:

“$BLE1_DIR\Drivers”

Now, as shown in the picture, within IAR, right click on the "STM32F4xx_HAL_Driver" folder name and select "Rename...". Choose "STM32L1xx_HAL_Driver" as the new name.



Within IAR, right click on the newly renamed "STM32L1xx_HAL_Driver" folder and select "Add -> Add Files..." and add all the files located in "$BLE1_DIR\Drivers\STM32L1xx_HAL_Driver\Src" to the project.

NOTE: make sure to include all the relevant files ending with "_ex.c". It is important that you add the "stm32l1xx_hal_spi_ex.c" which implements the HAL_SPI_Init() function needed by the BlueNRG driver.

Now, within IAR, remove the old HAL files for STM32F4 by right clicking on the files with the "stm32f4xx_hal" prefix and selecting "Remove".

Copy file “$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Src\cube_hal_f4.c” into

“$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Src\cube_hal_l1.c”.

To add the newly copied file to the project, within IAR right click on the "User" directory and select "Add -> Add Files..." and add the "cube_hal_l1.c" file to the project.

Within IAR, right click on "cube_hal_f4.c" and select "Remove" to remove it from the project.

Now open "cube_hal_l1.c" and update the "SystemClock_Config" function using a one suitable for STM32L1.

For example, you can copy the "SystemClock_Config" function that you can find in the “$CUBE_DIR\Projects\

\STM32L152RE-Nucleo\Examples\UART\UART_TwoBoards_ComIT\Src\main.c” file.

Edit "$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Inc\cube_hal.h" file and add a section like the following for L1:

```
#ifdef USE_STM32L1XX_NUCLEO
  #include "stm32l1xx_hal.h"
  #include "stm32l1xx_nucleo.h"
  #include "stm32l1xx_nucleo_bluenrg.h"
  #include "stm32l1xx_hal_conf.h"
  /* The following defines are needed because the BSP for Nucleo L1
   * uses different names compared to the BSP for Nucleo F4 and Nucleo L0
   */
  #define BUTTON_KEY BUTTON_USER
  #define KEY_BUTTON_PIN USER_BUTTON_PIN
#endif
```

Copy file "$CUBE_DIR\Projects\

\STM32L152RE-Nucleo\Examples\SPI\SPI_FullDuplex_ComIT\Inc\stm32l1xx_hal_conf.h" into "$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Inc" directory.

Now edit the newly copied

"$BLE1_DIR\Projects\Multi\Applications\SensorDemo\Inc\stm32l1xx_hal_conf.h" file and make sure that at the top of the file you decomment the same defines that were defined in the "stm32f4xx_hal_conf.h", i.e.:

```
#define HAL_MODULE_ENABLED
#define HAL_ADC_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
```

```
#define HAL_CORTEX_MODULE_ENABLED
```

Edit “$BLE1_DIR\Drivers\BSP\X-NUCLEO-IDB04A1\stm32_bluenrg_ble.h” file and add a section like the following for STM32L1:

```
#ifdef USE_STM32L1XX_NUCLEO
  #include "stm32l1xx_hal.h"
  #include "stm32l1xx_nucleo.h"
  #include "stm32l1xx_nucleo_bluenrg.h"
  #define SYSCLK_FREQ 32000000
#endif
```

NOTE: SYSCLK_FREQ should be set accordingly to the frequency set by the SystemClock_Config() function described earlier.

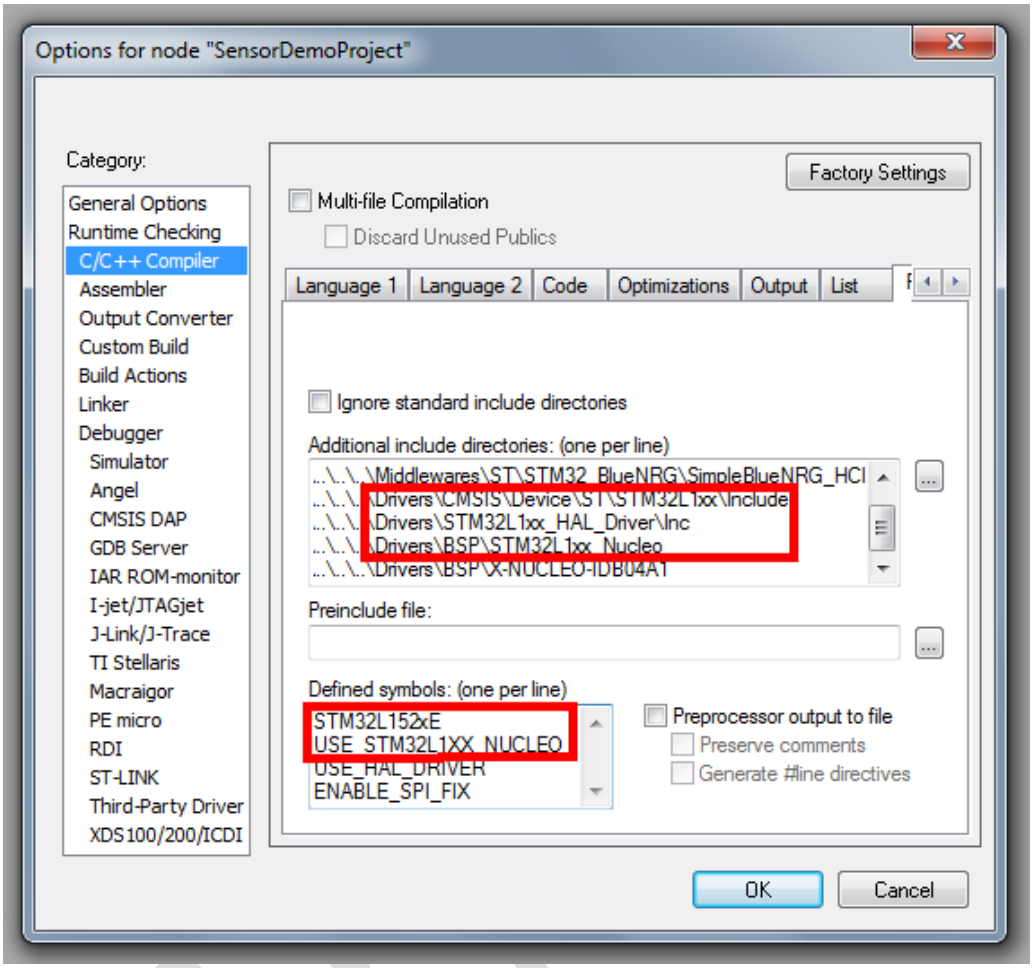
5.6 Update the project settings and rebuild

Within IAR select “Project -> Options...” and from the "C/C++ Compiler" category, select the "Preprocessor" tab.

In the "Additional include directories" section, change all occurrences of F4 to L1, in order to correctly include the L1 directories created in the previous step.

NOTE: there is a slight difference in "$BLE1_DIR\Drivers\BSP" directory between "STM32F4xx-Nucleo" and "STM32L1xx_Nucleo" because in one case there is a hyphen and in the other an underscore.

In the "Defined symbols" section, change "STM32F401xE" to "STM32L152xE" and "USE_STM32F4XX_NUCLEO" to "USE_STM32L1XX_NUCLEO".



Press "OK" to accept the changes.

Save the IAR EWARM project and then rebuild all by selecting "Project -> Rebuild All" from IAR menu.

There you go with your SensorDemo for NUCLEO-L152RE…