Freescale Semiconductor, Inc.
User's Guide

Document Number: MKWSWPUG
Rev. 0, 01/2016

Software Porting From MKW40Z to MKW30Z and MKW20Z Wireless MCUs

1. Introduction

This user's guide describes porting of software from the MKW40Z-based platforms to MKW30Z- and MKW20Z-based platforms. The starting point is the KW40Z Connectivity Software package, available on the Freescale website www.freescale.com.

This guide consists of two major chapters. Section 2, "Software porting from MKW40Z- based platforms to MKW30Z- based platforms" describes porting of software applications from the MKW40Z-based platforms to the MKW30Z-based platforms. Section 3, "Software porting from MKW40Z based platforms to MKW20Z based platforms" describes porting of software applications from the MKW40Z-based platforms to the MKW20-based platforms.

This document is intended for software engineers, software testers, software integrators, and customers designing their own hardware.

Contents

2. Acronyms and abbreviations

This table lists the acronyms and abbreviations used in this document:

Table 1. Acronyms and abbreviations

Term/abbreviation	Description
BLE	Bluetooth Low-Energy
DC	Direct Current
FRDM	Freescale Freedom development platform
GPIO	General-Purpose Input Output
I^2C	Inter-Integrated Circuit
KSDK	Kinetis SDK
LED	Light-Emitting Diode
LPUART	Low-Power UART
MCU	Microcontroller Unit
RTC	Real-Time Clock
UART	Universal Asynchronous Receiver Transmitter
XCVR	Transceiver

3. Software porting from MKW40Z-based platforms to MKW30Z-based platforms

The MKW30Z wireless MCU is a Bluetooth Low-Energy protocol chip that has the same memory footprint as the MKW40Z, but has a different pinout (32-pin MAPLGA) and a slightly different chip configuration. In this table, the pinout of the MKW30Z is compared to that of MKW40Z:

Table 2. Pinout comparison

Pin #	MKW40Z pinout	MKW30Z pinout
1	PTA0	PTC19
2	PTA1	PTA0
3	PTA2	PTA1
4	PTA16	PTA2
5	PTA17	PSWITCH
6	PTA18	DCDC_CFG
7	PTA19	VDCDC_IN
8	PSWITCH	DCDC_GND
9	DCDC_CFG	DCDC_LP
10	VDCDC_IN	DCD_LN
11	DCDC_LP	VDD_1P8OUT
12	DCDC_LN	VDD_1P45OUT_PMCIN
13	DCDC_GND	PTB3
14	VDD_1P8OUT	VDD_0
15	VDD_1P45OUT_PMCIN	PTB16
16	PTB0	PTB17
17	PTB1	PTB18
18	PTB2	VDDA
19	PTB3	EXTAL_32M
20	VDD_0	XTAL_32M
21	PTB16	VDD_RF2
22	PTB17	RF_N
23	PTB18	RF_P
24	ADC0_DP0	VDD_RF1

Table 2. Pinout comparison

Pin #	MKW40Z pinout	MKW30Z pinout
25	ADC0_DM0	PTC0
26	VSSA	PTC1
27	VREFH	PTC2
28	VDDA	PTC3
29	EXTAL_32M	VDD_1
30	XTAL_32M	PTC16
31	VDD_XTAL	PTC17
32	VDD_RF2	PTC18
33	RF_N	—
34	RF_P	—
35	VDD_RF1	—
36	PTC0	—
37	PTC1	—
38	PTC2	—
39	PTC3	—
40	PTC4	—
41	PTC5	—
42	PTC6	—
43	PTC7	—
44	VDD_1	—
45	PTC16	—
46	PTC17	—
47	PTC18	—
48	PTC19	—

KW30Z has a different pin assignment than KW40Z. KW30Z has subsets of KW40Z ports, and it has no differential ADC inputs. VREFH is internally connected to VDDA, and VSSA is internally connected to VSS.

3.1. Changes needed in IDE project options and settings

The starting point is to port an existing BLE example application project that is available in the current KW40Z Connectivity Software package. The temperature sensor does not require too many external components and it is suitable to be ported as an example. The board used in the example is Freescale Freedom FRDM-KW40Z development board.

To start, open the project in IAR Embedded Workbench for ARM IDE. The workspace is located at this path:

..\ConnSw\examples\bluetooth\temperature_sensor\frdmkw40z\FreeRTOS\build\iar\temperature_sensor.eww

The workspace contains two projects: the FreeRTOS library project and the BLE application project:

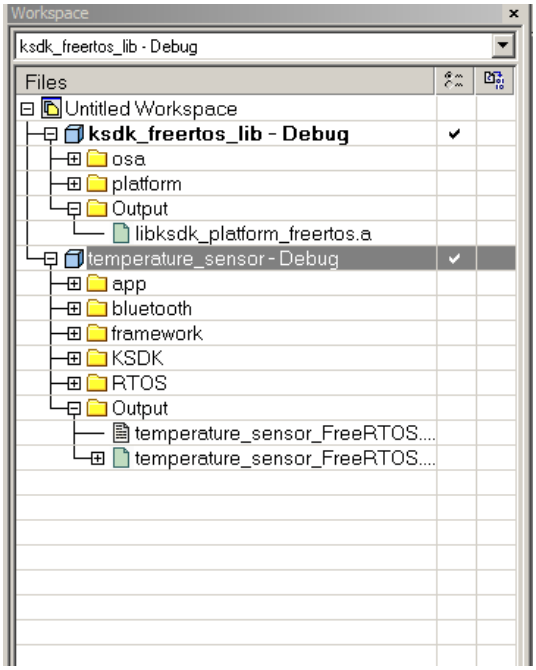


Figure 1. IAR workspace with two projects

3.1.1. Modifying and updating application project

1. Set *temperature_sensor* as the active project:

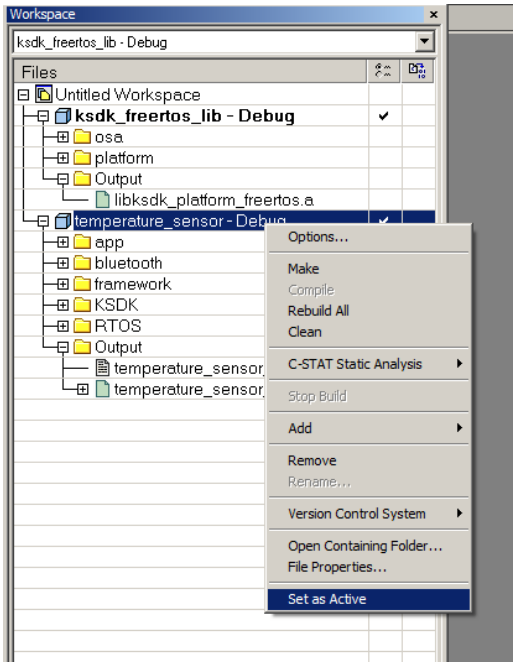


Figure 2. Setting application project as active project

2. Update the target processor:
   - Right-click the “lib” project and select “*Options->General Options->Target*”.
   - Change the device from “Freescale MKW40Z160xxx4” to “Freescale MKW30Z160xxx4”.

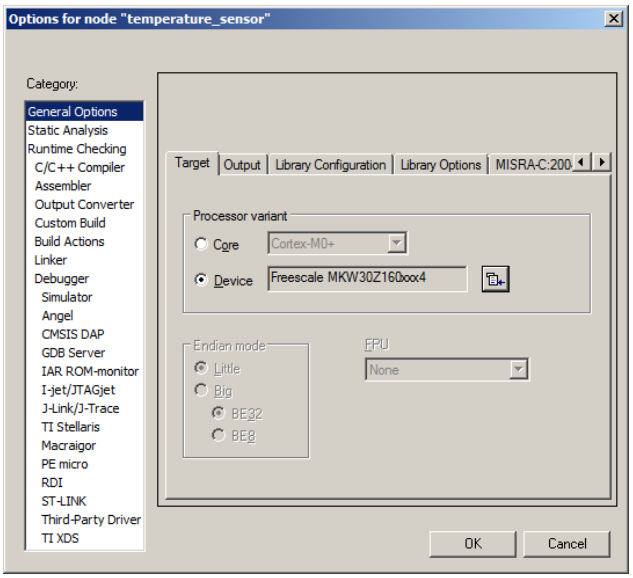


Figure 3. Changing target MCU

3. Create a new folder with a new board name (e.g., myBLECustomBoard) at this path:
   .. ConnSw\boards\myBLECustomBoard
4. Copy the board files from one of the existing board folders (the recommendation is to select the board that has the most in common with the custom board):

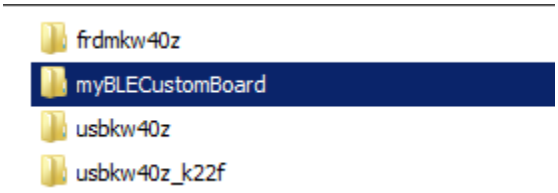


Figure 4. Creating custom board folder

5. Change the existing board files of the project to the new files from the custom board folder. The files to replace are highlighted in this figure:

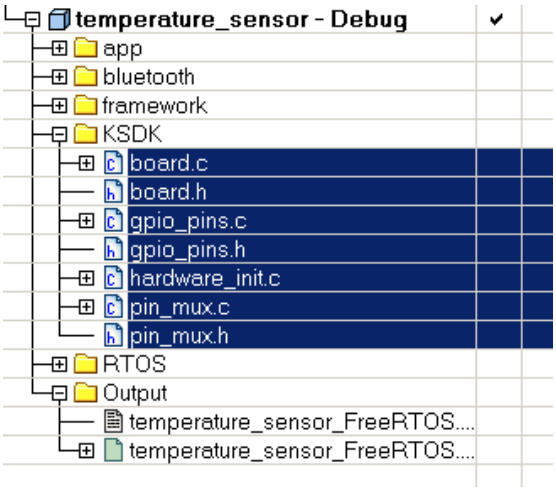


Figure 5. Replacing board files

6. If the project uses a pre-include file, open the *app_preinclude.h* header file (under the *app\temperature_sensor* group):

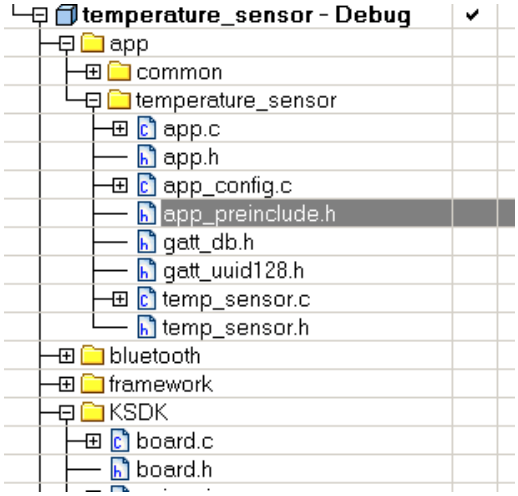


Figure 6. *app_preinclude.h* header file

This file contains important configuration information, such as the number of switches and LEDs the board uses, the number of application and stack timers, low-power settings, DC to DC operating mode, and so on. Examine this file and update it accordingly.

7. Edit the project paths and other project settings:
   - Right-click the project name and select "*Options...*".
   - Select "*C/C++ Compiler*" and then select the "*Preprocessor*" tab.
   - Change the path to board files from *$PROJ_DIR$\..\..\..\..\..\..\..\..\..\..\boards\frdmkw40z* to *$PROJ_DIR$\..\..\..\..\..\..\..\..\..\..\boards\myBLECustomBoard*.

- Change the following paths to the *MKW40Z4* folders to paths to the *MKW30Z4* folders by replacing the words (MKW40Z4 with MKW30Z4 for all existing paths). The result looks like this:

  $_KSDK_1_3_0_PATH_$\platform\devices\MKW30Z4\include

  $_KSDK_1_3_0_PATH_$\platform\devices\MKW30Z4\startup

  $_KSDK_1_3_0_PATH_$\platform\system\src\clock\MKW30Z4

  $_KSDK_1_3_0_PATH_$\platform\devices\MKW30Z4\startup

  $_KSDK_1_3_0_PATH_$\rtos\FreeRTOS\config\KW30Z4\iar

  NOTE

  KSDK_1_3_0_PATH is a Windows® OS system variable and its name is subject to change. This variable is set automatically by the installer of the Connectivity Software package and points to the Kinetis SDK location. Do not change the name of this variable in any way for the purpose of the porting process described in this guide. Check the variable set by the version of the KW40Z Connectivity Software package you have installed (in the release notes of the respective package).

- Change the CPU type from CPU_MKW40Z160VHT4 to CPU_MKW30Z160VHM4.
- Select "*Assembler*" and then select the "*Preprocessor*" tab.
- Change the existing search path to the new one:
  $_KSDK_1_3_0_PATH_$\rtos\FreeRTOS\config\KW30Z4\iar

8. Modify the XCVR driver:
   - Create a new folder called *XCVR\MKW30Z4* and copy the entire content from the *XCVR\MKW40Z4* folder into it (relative path is *\ConnSw\framework\XCVR\MKW40Z4*).

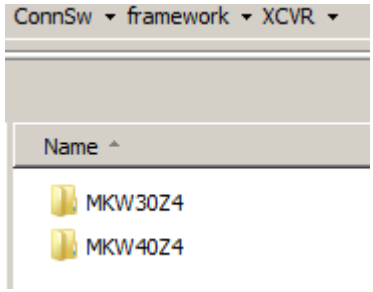


Figure 7. Creating new folder for XCVR driver files

– Rename the files in the new folder as shown in this figure:

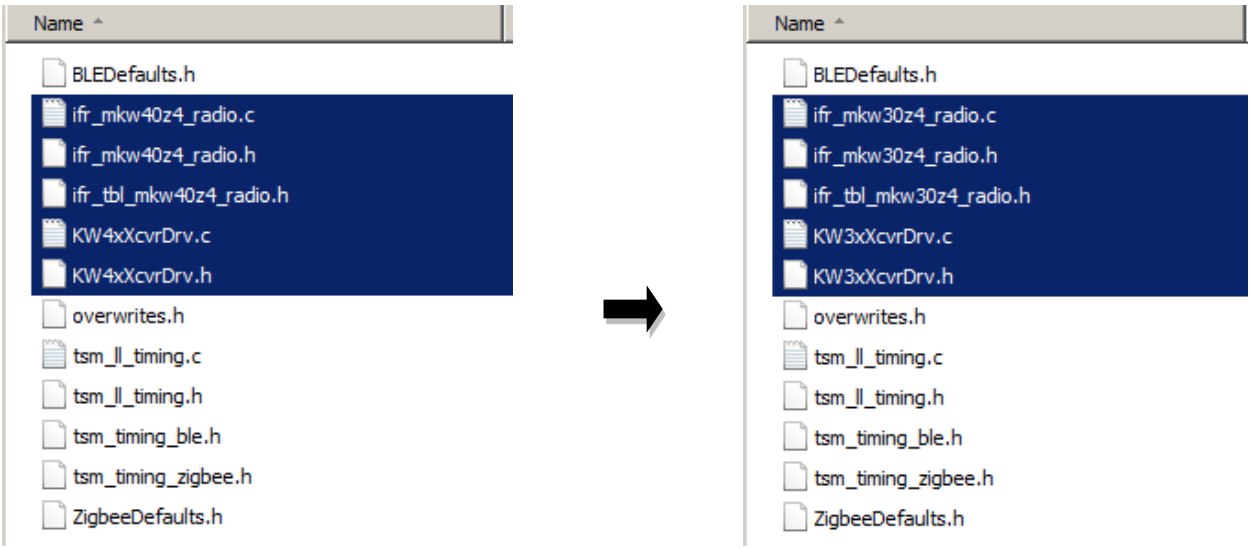


Figure 8. Renaming XCVR files

– Delete the *MKW40Z* folder under *framework\XCVR* group, create a new group called *MKW30Z*, and add all files from the previously created folder *\ConnSw\framework\XCVR\MKW30Z4* into it.

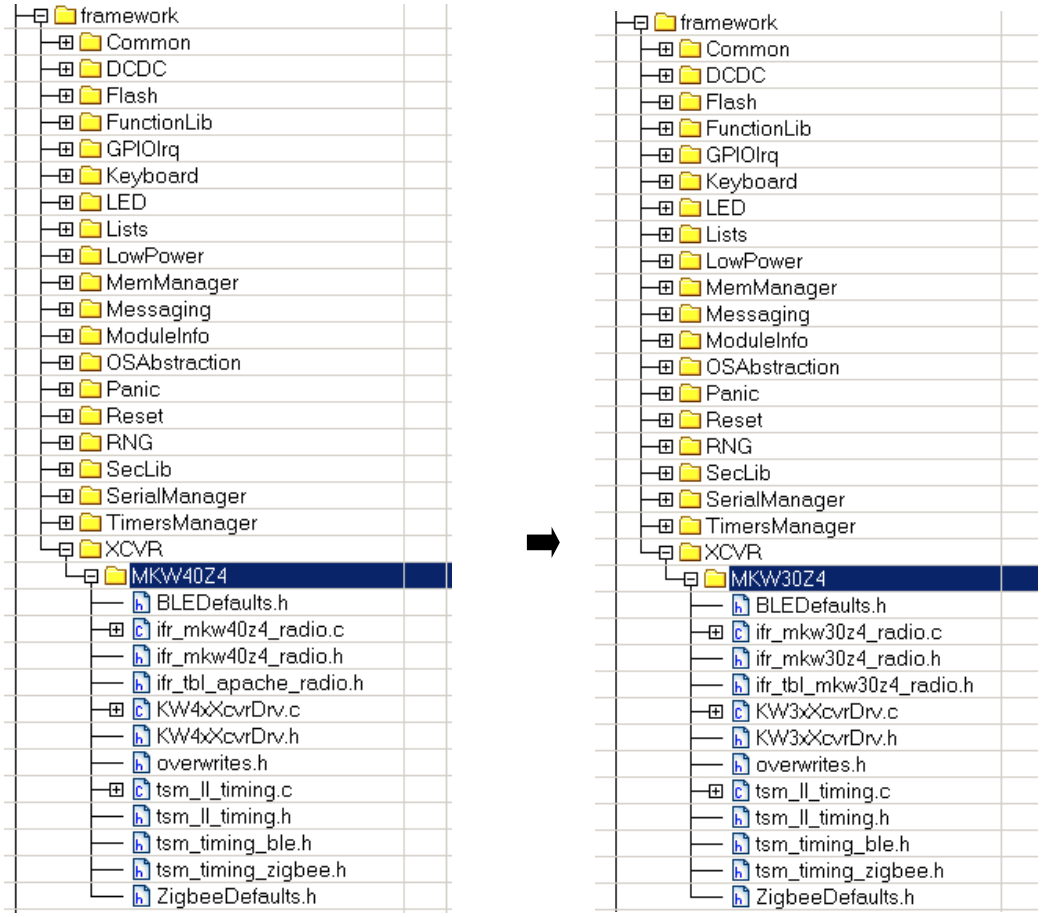


Figure 9. Replacing XCVR files in IAR workspace

- Edit the *ifr_mkw30z4_radio.c* file by replacing the includes below:

```
#include "ifr_mkw40z4_radio.h"
#include "MKW40Z4.h"
#include "KW4xXcvrDrv.h"
```

→

```
#include "ifr_mkw30z4_radio.h"
#include "MKW30Z4.h"
#include "KW3xXcvrDrv.h"
```

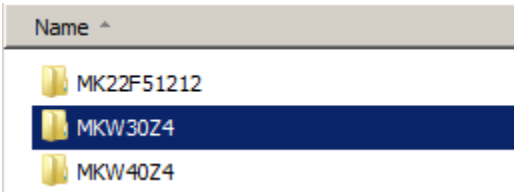
- Edit the *KW3xXcvrDrv.c* file by replacing the includes below:

```
#include "BLEDefaults.h"
#include "ZigbeeDefaults.h"
#include "KW4xXcvrDrv.h"
#include "fsl_os_abstraction.h"
#include "fsl_device_registers.h"
#include "fsl_sim_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_port_hal.h"

#include "tsm_ll_timing.h"
#include "ifr_mkw40z4_radio.h"
```

→

```
#include "BLEDefaults.h"
#include "ZigbeeDefaults.h"
#include "KW3xXcvrDrv.h"
#include "fsl_os_abstraction.h"
#include "fsl_device_registers.h"
#include "fsl_sim_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_port_hal.h"

#include "tsm_ll_timing.h"
#include "ifr_mkw30z4_radio.h"
```

9. Create a new linker configuration file:
   - Create a new folder called *MKW30Z4* at this path: *KW40Z_Connectivity_Software_1.0.0\ConnSw\platform\devices*.



   - Copy the content from the *MKW40Z* folder to the new *MKW30Z4* folder.
   - Rename the linker configuration file within the *MKW30Z4* folder from *MKW40Z160xxx4_connectivity.icf* to *MKW30Z160xxx4_connectivity.icf*.
   - Reopen the project options by right-clicking the project name and selecting "*Options*...".
   - Select "*Linker*" section and change *MKW40Z4/linker/iar/MKW40Z160xxx4_connectivity.icf* to *MKW30Z4/linker/iar/MKW30Z160xxx4_connectivity.icf*. Selecting the linker configuration file by browsing adds the absolute file path (not recommended).
10. Replace the startup files for MKW40Z with files for MKW30Z. The files are located at this path: *\KSDK_1.3.0\platform\devices\MKW30Z4\startup*.

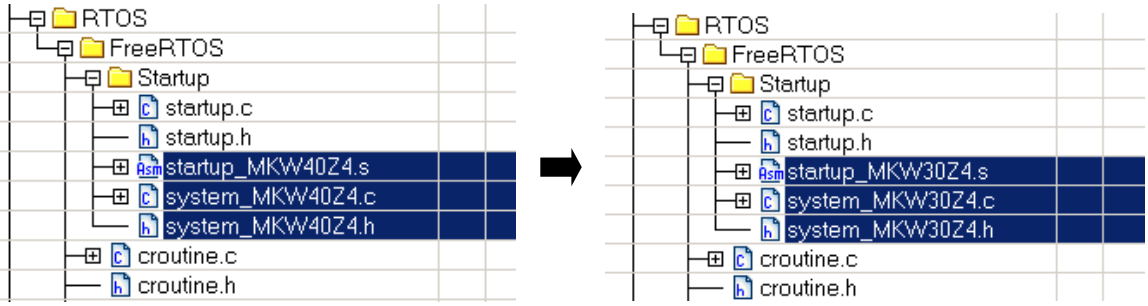


Figure 10. Replacing startup files

3.2. Updating board files

After updating the project and performing the steps described in the previous section, update the board files to make the project work with the new BLE custom board. The board files are part of the KSDK software package.

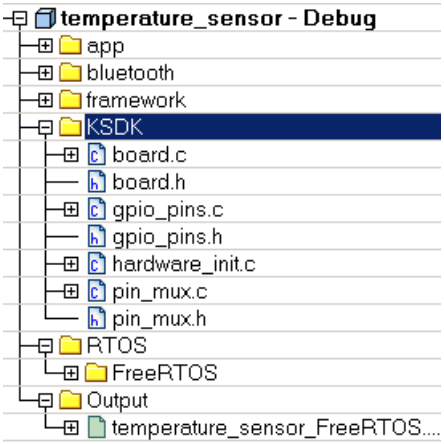


Figure 11. KSDK board files

NOTE

The MKW30Z MCUs have only subsets of port's pins available to its package, as shown in this table:

Table 3. MKW30Z available GPIO pins

PORTA	PORTB	PORTC
PTA0	PTB3	PTC0
PTA1	PTB16	PTC1
PTA2	PTB17	PTC2
—	PTB18	PTC3
—	—	PTC16
—	—	PTC17
—	—	PTC18
—	—	PTC19

3.2.1. *board.c* source file

This file contains the implementation of various functions used to initialize and set up the wireless MCU and peripherals.

- The *BOARD_InitOsc0* function is used to initialize the OSC0 oscillator. You don't have to change its current implementation.
- The *BOARD_InitRtcOsc* function is used to initialize the RTC peripheral. You don't have to change its current implementation.
- The *BOARD_InitAdc* function initializes the ADC peripheral. You don't have to change its current implementation, but it may be necessary to change settings such as ADC channel,

resolution, mode (differential or single-ended), and so on.

- The *BOARD_ClockInit* function sets the XTAL port and pins, the pin mux mode, and then calls the specific clock-initialization functions. You don't have to change its current implementation.
- The *BOARD_DCDCInit* function initializes the DC to DC converter. You don't have to change its current implementation, but you can choose a different operation mode according to the board design: buck, boost, or bypass. See the *mDcdcDefaultConfig* variable definition and the *APP_DCDC_MODE* macro.
- The *dbg_uart_init* function initializes the debug UART interface. Check and update the *BOARD_DEBUG_UART_INSTANCE* and *CLOCK_INIT_CONFIG* macros in the *board.h* header file.

3.2.2. *board.h* header file (contains important macro definitions)

- BOARD_NAME—the name of the board.
- CLOCK_INIT_CONFIG—clock-initialization configuration.
- BOARD_USE_LPUART—set this macro to 1 if using low-power UART.
- APP_SERIAL_INTERFACE_INSTANCE—represents the physical instance ID of the serial interface (e.g., if UART is the serial interface and APP_SERIAL_INTERFACE_INSTANCE is set to 0, then it is called the UART0 peripheral.
- APP_DCDC_MODE—this macro sets the DC to DC operation mode (buck, boost, or bypass).
- BOARD_DEBUG_UART_INSTANCE—instance ID of the UART peripheral used to print debug messages. The value of this macro represents the physical instance ID of the UART (e.g., if set to 3 => UART3).
- BOARD_DEBUG_UART_BASEADDR—represents the debug UART base address (see the UART peripheral section in the reference manual).
- BOARD_DEBUG_UART_BAUD—sets the debug UART baud rate (set to 115200 bps by default).
- BOARD_UART_CONFIG—selects the UART interface to use (debug UART, FRDM UART header, or alternate FRDM UART header).
- The other macros defined in the file are not used by the Connectivity Software and you can ignore them.

3.2.3. *gpio_pins.c* source file

This file is important because it defines the GPIO pins that are used by switches, LEDs, and other I/O peripherals. Here you can add, remove, or modify the existing pin definitions. In the below example, the pins are grouped into properties structures. Each pin has a name, has or has not the pull enabled, has or has not the passive filter enabled, and so on. See the reference manual for more details about the GPIO ports and pins.

```
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntFallingEdge,
    },
    {
        .pinName = kGpioSW2,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntFallingEdge
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

3.2.4. *gpio_pins.h* header file

This file contains the enumeration of the GPIO pins and makes a link between the given name of the pin and the GPIO port name and pin number. Let's take a look at the below example: *kGpioLED1* is assigned using the *GPIO_MAKE_PIN* macro and points to PTC1 (GPIOC, PORTC, pin 1).

```
enum _gpio_pins
{
    kGpioLED1        = GPIO_MAKE_PIN(GPIOC_IDX, 1),
    kGpioLED2        = GPIO_MAKE_PIN(GPIOC_IDX, 0),
    kGpioSW1         = GPIO_MAKE_PIN(GPIOB_IDX, 16),
    kGpioSW2         = GPIO_MAKE_PIN(GPIOB_IDX, 17),
    kGpioI2cDAP      = GPIO_MAKE_PIN(GPIOB_IDX, 1),
    kGpioSpiDAP      = GPIO_MAKE_PIN(GPIOB_IDX, 1),
};
```

Link the given name of the GPIO and the GPIO port and pin correctly. Make sure the ports and pins defined in the code are those from Table 2.

3.2.5. *hardware_init.c* source file

This file contains a single function (*hardware_init*), which is usually called by the application only once at the system initialization. Initialize the GPIO ports' clock gating here. If you don't want to use some of the ports, disable them by commenting the corresponding line of code.

3.2.6. *pin_mux.c* source file

This file contains the pin-multiplexing functions that you can use to configure various MCU peripherals for further usage. The functions are:

void pin_mux_GPIO(uint32_t instance);

void pin_mux_I2C(uint32_t instance);

void pin_mux_TPM(uint32_t instance);

void pin_mux_RTC(uint32_t instance);

void pin_mux_SPI(uint32_t instance);

void pin_mux_LPUART(uint32_t instance);

Depending on the function called, *instance* has this meaning:

- For GPIO: 0 = PORTA, 1 = PORTB, and 2 = PORTC.
- For I^2C: 0 = I^2C0 on PTB0(SCL)/PTB1(SDA) and 1 = I^2C0 on PTC2(SCL)/PTC3(SDA).
- For RTC: the *instance* parameter is ignored.
- For LPUART: if *instance* = 0, then the LPUART0 hardware module is used as follows:
  - If the board UART configuration is the debug one, then RX = PTC6 and TX = PTC7.
  - If the board UART configuration is the FRDM header, then RX = PTC17 and TX = PTC18.
  - If the board UART configuration is the alternate FRDM header, then RX = PTC2, TX = PTC3, CTS = PTC0, and RTS = PTC1.

NOTE

For any other values of *instance*, there is no implementation within the *pin_mux_LPUART* function.

The *pin_mux.h* header file contains only prototypes of functions defined in the *pin_mux.c* source file. Now all is done, and you can clean, rebuild, and run the project.

4. Software porting from MKW40Z-based platforms to MKW20Z-based platforms

The MKW20Z wireless MCU is an IEEE 802.15.4-based protocol chip and has an important feature that makes porting of software seamless: it is pin-to-pin compatible with the MKW40Z MCU and has the same package type (48-pin MAPLGA). Therefore, the IEEE 802.15.4 or Zigbee applications designed and compiled for MKW40Z-based platforms work on MKW20Z-based platforms without modifications.

For consistency and scalability reasons, follow the steps described below when porting IEEE 802.15.4 or Zigbee software application to MKW20Z-based platforms.

4.1. Changes needed in IDE project options and settings

Because using an existing example is the best way to show how to port an application, the IEEE 802.15.4 MAC example application MyWirelessApp from the KW40Z Connectivity Software package is used. The board used with the example is Freescale Freedom FRDM-KW40Z.

The project has a number of sub-projects, but the porting rules are valid for all of them. The demo example requires a coordinator and an end device for running and testing, but only the coordinator project is taken into account. The same porting steps are valid for the end device. To start, open the project in IAR Embedded Workbench for ARM IDE. The project is located at this path: *\ConnSw\examples\ieee_802_15_4\MyWirelessApp\Coordinator\frdmkw40z\FreeRTOS\build\iar\MyWirelessAppCoordinator.eww*

After opening, the workspace contains two projects: a library project for FreeRTOS (FreeRTOS is integrated in KSDK) and the wireless application project:

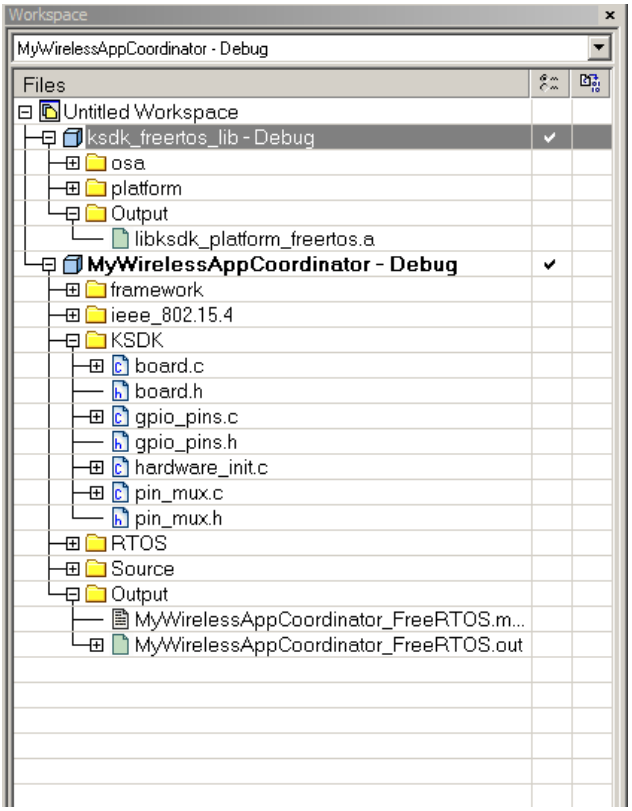


Figure 12. IAR workspace with two projects

Because the MyWirelessApp uses the FreeRTOS library as well as the platform-dependent drivers and OS abstraction layer, the first thing to do is to update and rebuild the library project. The steps are described in the following sections.

4.1.1. Modifying and updating FreeRTOS library project

This is an optional stage. As a general KSDK rule, the FreeRTOS libraries always have the biggest MCU in the family as a platform, and they are applicable to all family members.

1. Update the target processor:
   - Right-click the “lib” project and select “*Options*”.
   - In “*General Options*”, click the “*Target*” tab.
   - Change the device from “Freescale MKW40Z160xxx4” to “Freescale MKW20Z160xxx4”.
2. Modify the compiler options:
   - In the “*Options*” window, select “*C/C++ Compiler*”.
   - Select “*Preprocessor*” tab.

- Modify the existing include directories paths as follows:
  $PROJ_DIR$/../../../../rtos/FreeRTOS/port/iar
  $PROJ_DIR$/../../../../rtos/FreeRTOS/config/KW20Z4/iar
  $PROJ_DIR$/../../../../rtos/FreeRTOS/include
  $PROJ_DIR$/../../../../rtos/FreeRTOS/src
  $PROJ_DIR$/../../../../platform/CMSIS/Include
  $PROJ_DIR$/../../../../platform/devices
  $PROJ_DIR$/../../../../platform/devices/MKW20Z4/include
  $PROJ_DIR$/../../../../platform/devices/MKW20Z4/startup
  $PROJ_DIR$/../../../../platform/utilities/inc
  $PROJ_DIR$/../../../../platform/hal/inc
  $PROJ_DIR$/../../../../platform/drivers/inc
  $PROJ_DIR$/../../../../platform/system/inc
  $PROJ_DIR$/../../../../platform/osa/inc

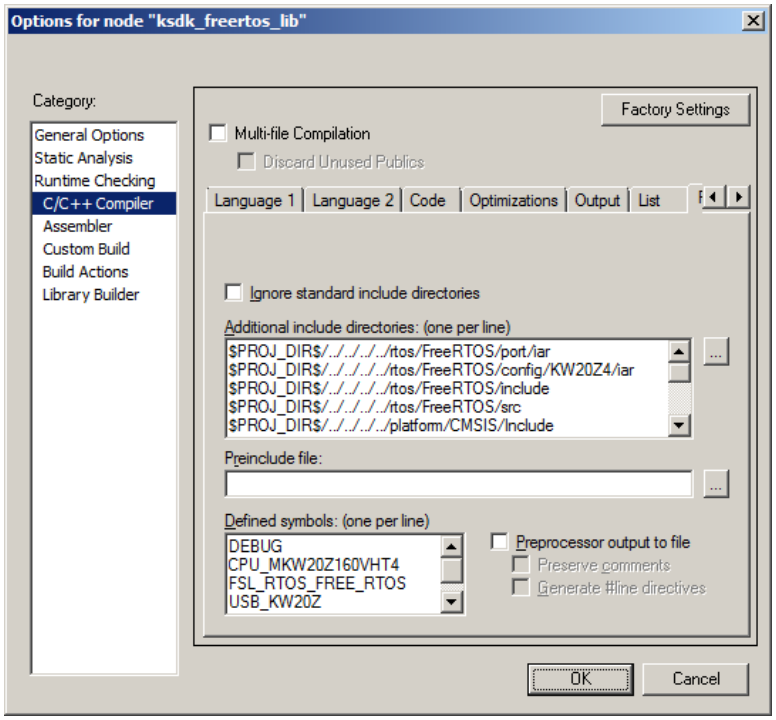


Figure 13. Modifying include paths and defined symbols

- Change the defined symbols as follows:
  CPU_MKW40Z160VHT4 to *CPU_MKW20Z160VHT4*
  USB_KW40Z to *USB_KW20Z*
- Close the "*Options*" window and save the workspace.

3. Clean and make the library project.

At this point, the KSDK FreeRTOS library is rebuilt. The next step is to modify the application project (MyWirelessApp).

4.1.2. Modifying and updating application project

1. Set MyWirelessApp as the active project:

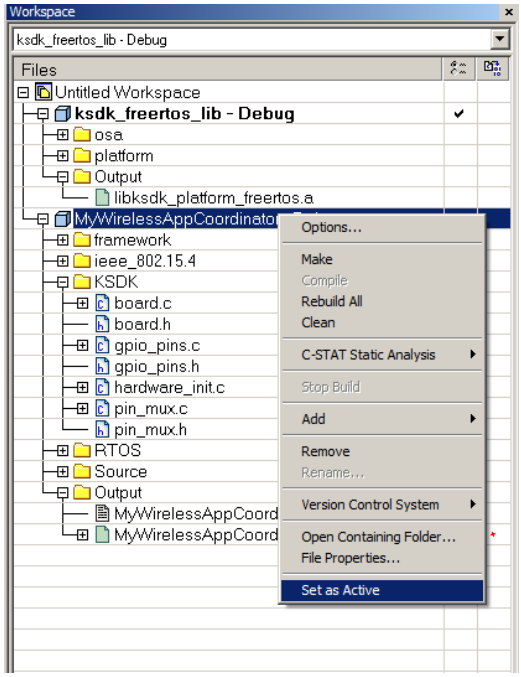


Figure 14. Setting MyWirelessApp project as active project

2. Update the target processor:
   - Right-click the “lib” project and select “*Options*”.
   - In “*General Options*”, click the “*Target*” tab.
   - Change the device from “Freescale MKW40Z160xxx4” to “Freescale MKW20Z160xxx4”.

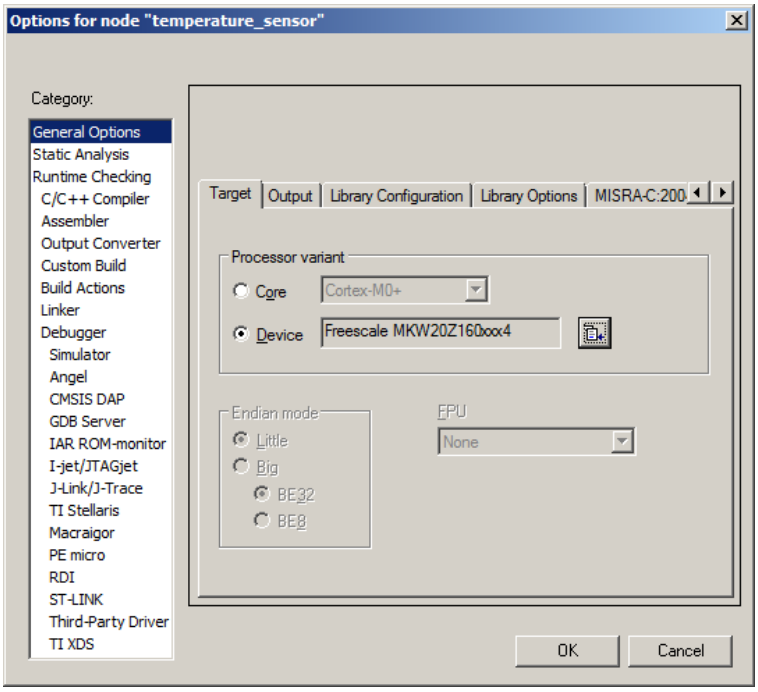


Figure 15. Changing target MCU

3. Create a new folder with a new board name (e.g., myCustomBoard) at this path: *..\KW40Z_Connectivity_Software_1.0.0\ConnSw\boards\myCustomBoard*
4. Copy the board files from one of the existing board folders (select the board that has the most in common with the custom board).

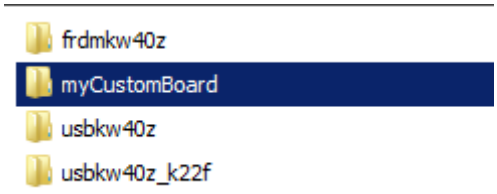


Figure 16. Creating custom board folder

5. Replace the existing board files with the new files from the custom board folder. The files to replace are highlighted in this figure:

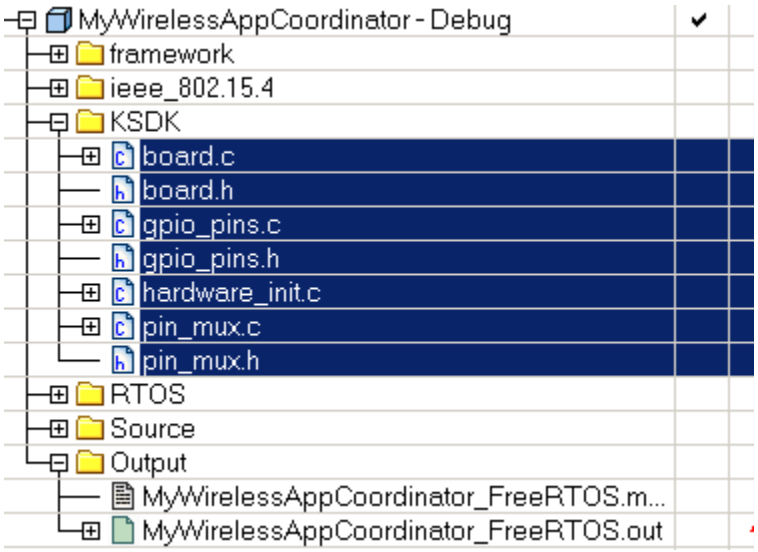


Figure 17. Replacing board files

6. If the project is using a pre-include file, open the *app_preinclude.h* header file (under "*Source*" group) and replace the definition of the MCU type:

```
#define CPU_MKW40Z160VHT4                1 ➔ #define CPU_MKW20Z160VHT4                1
```

7. Edit the project paths and other project settings:
   - Right-click the project name and select "*Options…*".
   - Select "*C/C++ Compiler*" and then select the "*Preprocessor*" tab.
   - Change the path to board files from *$PROJ_DIR$\..\..\..\..\..\..\..\..\..\..\boards\frdmkw40z* to *$PROJ_DIR$/../../../../../../../../../boards/myCustomBoard*.
   - Change the paths to *MKW40Z4* folders to paths to *MKW20Z4* folders by changing MKW40Z4 to MKW20Z4 in all paths. The result looks like this:

     $_KSDK_1_3_0_PATH_$/platform/devices/MKW20Z4/include

     $_KSDK_1_3_0_PATH_$/platform/devices/MKW20Z4/startup

     $_KSDK_1_3_0_PATH_$/platform/system/src/clock/MKW20Z4

     $_KSDK_1_3_0_PATH_$/platform/devices/MKW20Z4/startup

     $_KSDK_1_3_0_PATH_$/rtos/FreeRTOS/config/KW20Z4/iar

NOTE

The *KSDK_1_3_0_PATH* is a Windows OS system variable and its name is subject to change. This variable is automatically set by the installer of the Connectivity Software package and points to the Kinetis SDK location. Do not change the name of this variable in any way for the purpose of the porting process described in this guide. Check the variable set by the version of the KW40Z Connectivity Software package you have installed (in the release notes of the respective package).

- Change the CPU type from CPU_MKW40Z160VHT4 to CPU_MKW20Z160VHT4 (the pre-include file overrides these macros).
- Select "*Assembler*" and then select the "*Preprocessor*" tab.
- Change the existing search path to the new one: *$_KSDK_1_3_0_PATH_$\rtos\FreeRTOS\config\KW20Z4\iar*

8. Modify the XCVR driver:
   - Create a new folder called *XCVR\MKW20Z4* and copy the entire content of the *XCVR\MKW40Z4* folder (the relative path is *\ConnSw\framework\XCVR\MKW20Z4*).
   - Rename the files in the new folder as shown in this figure:

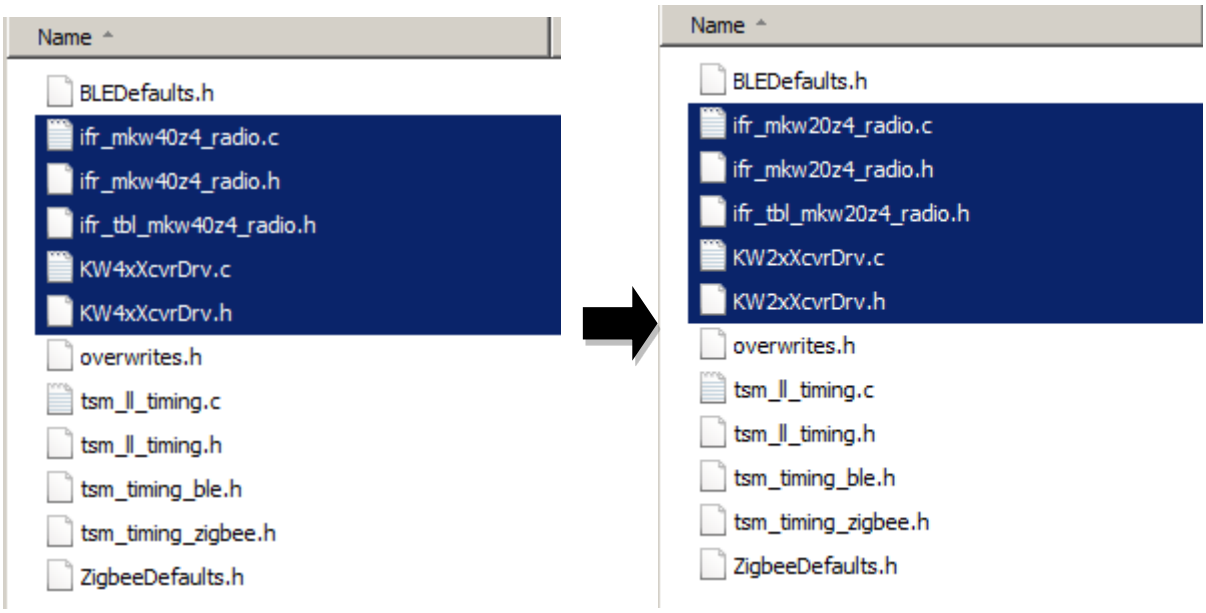


Figure 18. Renaming XCVR files

– Delete the *MKW40Z* folder in the *framework/XCVR* group, create a new group called *MKW20Z*, and add all files from the *\ConnSw\framework\XCVR\MKW20Z4* folder:

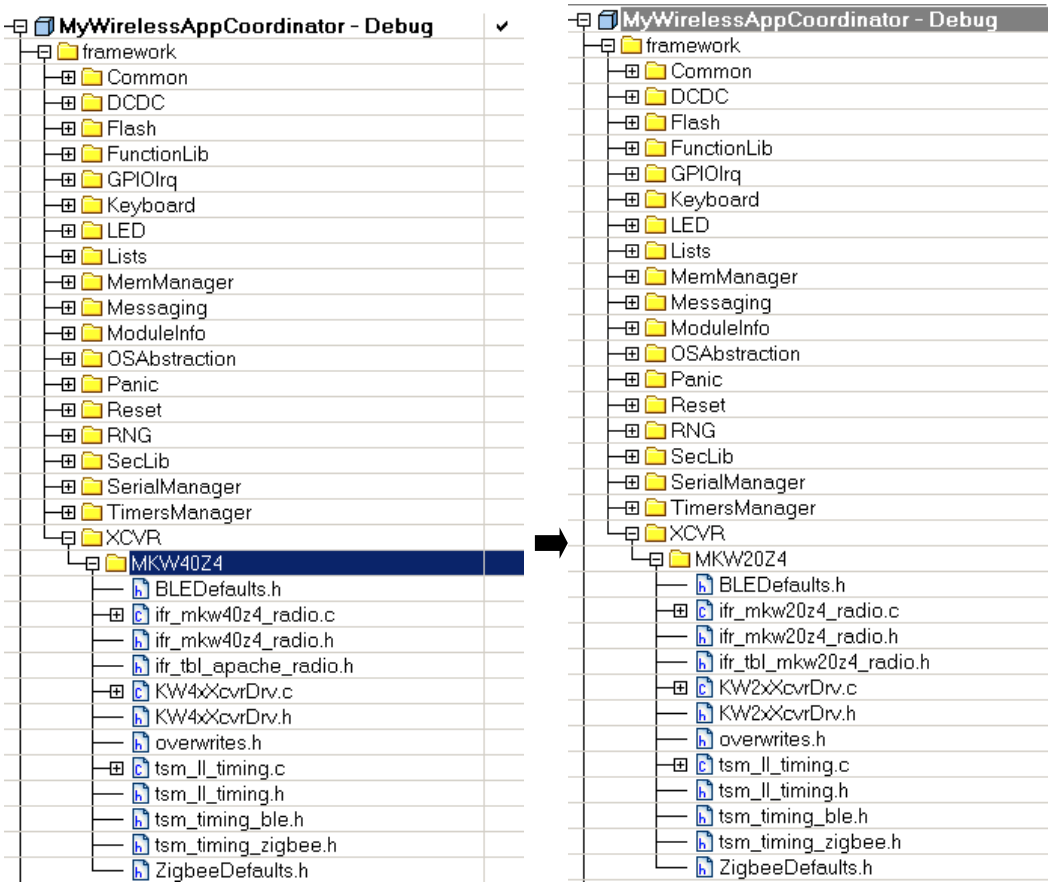


Figure 19. Replacing XCVR files in IAR workspace

– Edit the *ifr_mkw20z4_radio.c* file by changing the includes as follows:

```
#include "ifr_mkw40z4_radio.h"
#include "MKW40Z4.h"
#include "KW4xXcvrDrv.h"
```

```
#include "ifr_mkw20z4_radio.h"
#include "MKW20Z4.h"
#include "KW2xXcvrDrv.h"
```

– Edit the *KW2xXcvrDrv.c* file by changing the includes as follows:

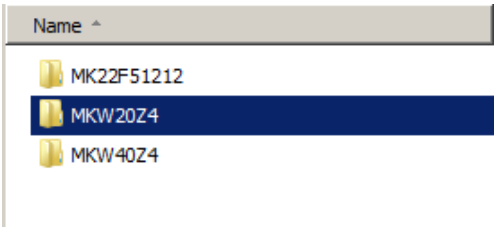
```
#include "KW4xXcvrDrv.h"
#include "fsl_os_abstraction.h"
#include "fsl_device_registers.h"
#include "fsl_sim_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_port_hal.h"

#include "tsm_ll_timing.h"
#include "ifr_mkw40z4_radio.h"
```

```
#include "KW2xXcvrDrv.h"
#include "fsl_os_abstraction.h"
#include "fsl_device_registers.h"
#include "fsl_sim_hal.h"
#include "fsl_gpio_hal.h"
#include "fsl_port_hal.h"

#include "tsm_ll_timing.h"
#include "ifr_mkw20z4_radio.h"
```

– Edit the *PhyISR.c* file by changing *#include "KW4xXcvrDrv.h"* to *#include "KW2xXcvrDrv.h"*.

– Edit the *PhyPacketProcessor.c* file by changing *#include "KW4xXcvrDrv.h"* to *#include "KW2xXcvrDrv.h"* and *#include "ifr_mkw40z4_radio.h"* to *#include "ifr_mkw20z4_radio.h"*.

9. Create a new linker configuration file:
   - Create a new folder called *MKW20Z4* at this path: *KW40Z_Connectivity_Software_1.0.0\ConnSw\platform\devices*



   - Copy the content of the *MKW40Z* folder to the *MKW20Z4* folder.
   - Rename the linker configuration file in the *MKW20Z4* folder from *MKW40Z160xxx4_connectivity.icf* to *MKW20Z160xxx4_connectivity.icf*.
   - Reopen the project options by right-clicking the project name and selecting "*Options...*".
   - Select the "*Linker*" section and replace *MKW40Z4/linker/iar/MKW40Z160xxx4_connectivity.icf* with *MKW20Z4/linker/iar/MKW20Z160xxx4_connectivity.icf*. Selecting the linker configuration file by browsing adds the absolute file path (not recommended).
10. Change the startup files for MKW40Z to those for MKW20Z. The files are located at this path: *KW40Z_Connectivity_Software_1.0.0\KSDK_1.3.0\platform\devices\MKW20Z4\startup*

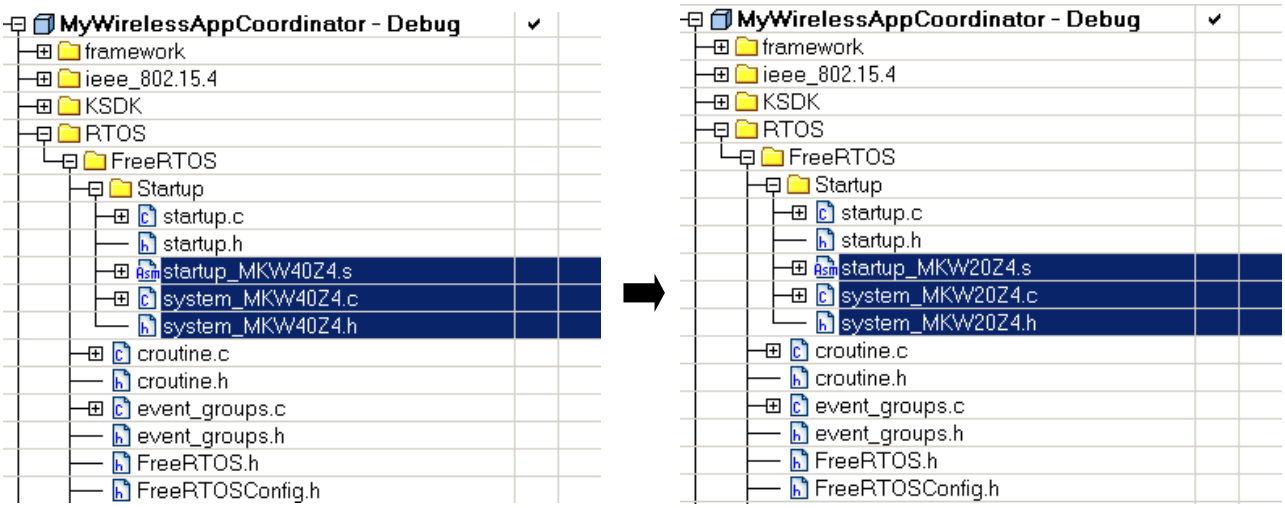


Figure 20. Replacing startup files

4.2. Updating board files

The next step after updating the project and performing all steps from previous sections is to update the board files to make the project work with the new custom board. The board files are a part of the KSDK software package.

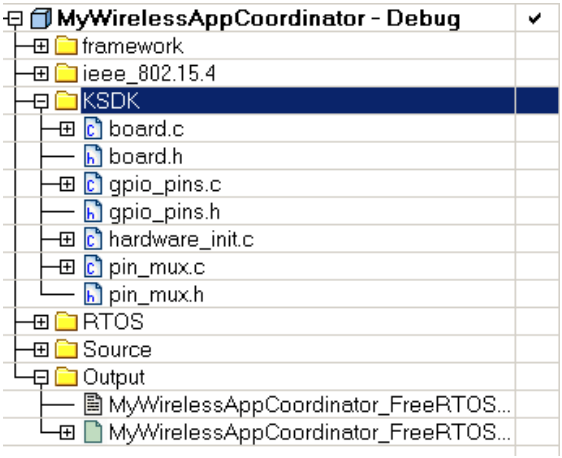


Figure 21. KSDK board files

4.2.1. *board.c* source file

This file contains the implementation of various functions that are used to initialize and set up the wireless MCU and peripherals.

- The *BOARD_InitOsc0* function initializes the OSC0 oscillator. You don't have to change its current implementation.
- The *BOARD_InitRtcOsc* function initializes the RTC peripheral. You don't have to change its current implementation.
- The *BOARD_InitAdc* function initializes the ADC peripheral. You don't have to change its current implementation, but it may be necessary to change settings such as ADC channel, resolution, mode (differential or single-ended), and other.
- The *BOARD_ClockInit* function sets the XTAL port, pins, pin mux mode, and then it calls the specific clock-initialization functions. You don't have to change its current implementation.
- The *BOARD_DCDCInit* function initializes the DC to DC converter. You don't have to change its current implementation, but you can choose a different operation mode according to the board design: buck, boost, or bypass. See the *mDcdcDefaultConfig* variable definition and the *APP_DCDC_MODE* macro.
- The *dbg_uart_init* function initializes the debug UART interface. Check and update the *BOARD_DEBUG_UART_INSTANCE* and *CLOCK_INIT_CONFIG* macros within the "*board.h*" header file (if necessary).

4.2.2. *board.h* header file

This file contains these important macro definitions:

- BOARD_NAME—the name of the board.
- CLOCK_INIT_CONFIG—clock-initialization configuration.
- BOARD_USE_LPUART—set this macro to 1 if using the low-power UART.
- APP_SERIAL_INTERFACE_INSTANCE—represents the physical instance ID of the serial interface (for example, if UART is the serial interface and APP_SERIAL_INTERFACE_INSTANCE is set to 0, then it is called the UART0 peripheral.
- APP_DCDC_MODE—this macro sets the DC to DC operation mode (buck, boost, or bypass).
- BOARD_DEBUG_UART_INSTANCE—the instance ID of the UART peripheral used to print debug messages. The value of this macro represents the physical instance ID of the UART (if set to 3, then it is UART3).
- BOARD_DEBUG_UART_BASEADDR—represents the debug UART base address (see the UART peripheral section in the reference manual).
- BOARD_DEBUG_UART_BAUD—sets the debug UART baud rate (it is 115200 bps by default).
- BOARD_UART_CONFIG—selects the UART interface to use (debug UART, FRDM UART header, alternate FRDM UART header).
- The other macros defined in the file are not used by the Connectivity Software and you can ignore them.

4.2.3. *gpio_pins.c* source file

This file is important because it defines the GPIO pins used by switches, LEDs, and other I/O types. Add, remove, or modify the existing pin definitions here. In the example below, the pins are grouped into properties structures. Each pin has a name, has or has not the pull enabled, has or has not the passive filter enabled, and so on. See the reference manual for more details about GPIO ports and pins.

```
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntFallingEdge,
    },
    {
        .pinName = kGpioSW2,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntFallingEdge
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

4.2.4. *gpio_pins.h* header file

This file contains an enumeration of the GPIO pins, and links the given name of the pin and the GPIO port name and pin number. In this example, *kGpioLED1* is assigned using the *GPIO_MAKE_PIN* macro and points to PTC5 (GPIOC, PORTC, pin 5):

```
enum _gpio_pins
{
    kGpioLED1        = GPIO_MAKE_PIN(GPIOC_IDX, 5),
    kGpioLED2        = GPIO_MAKE_PIN(GPIOC_IDX, 4),
    kGpioLED3        = GPIO_MAKE_PIN(GPIOC_IDX, 1),
    kGpioLED4        = GPIO_MAKE_PIN(GPIOC_IDX, 0),
    kGpioSW1         = GPIO_MAKE_PIN(GPIOA_IDX, 18),
    kGpioSW2         = GPIO_MAKE_PIN(GPIOA_IDX, 19),
    kGpioI2cDAP      = GPIO_MAKE_PIN(GPIOB_IDX, 1),
    kGpioSpiDAP      = GPIO_MAKE_PIN(GPIOB_IDX, 1),
};
```

Link the given name of the GPIO and the GPIO port and pin correctly.

4.2.5. *hardware_init.c* source file

This file contains the *hardware_init* function, which is usually called by the application only once during the system initialization. The GPIO ports' clock gating is initialized here, so if you don't use some of the ports, disable them here by commenting the corresponding line of code.

4.2.6. *pin_mux.c* source file

This file contains pin-multiplexing functions that you can use to configure various MCU peripherals for further usage. These functions are:

void pin_mux_GPIO(uint32_t instance);

void pin_mux_I2C(uint32_t instance);

void pin_mux_TPM(uint32_t instance);

void pin_mux_RTC(uint32_t instance);

void pin_mux_SPI(uint32_t instance);

void pin_mux_LPUART(uint32_t instance);

Depending on the function called, *instance* has this meaning:

- For GPIO: 0 = PORTA, 1 = PORTB, and 2 = PORTC.
- For I^2C: 0 = I^2C0 on PTB0(SCL)/PTB1(SDA) and 1 = I^2C0 on PTC2(SCL)/PTC3(SDA).
- For RTC: the *instance* parameter is ignored.
- For LPUART: if *instance* = 0, then the LPUART0 hardware module is used as follows:
  - If the board UART configuration is the debug one, then RX = PTC6 and TX = PTC7.
  - If the board UART configuration is the FRDM header, then RX = PTC17 and TX = PTC18.
  - If the board UART configuration is the alternate FRDM header, then RX = PTC2, TX = PTC3, CTS = PTC0 and RTS = PTC1.

NOTE

For any other values of *instance*, there is no implementation within the *pin_mux_LPUART* function.

4.2.7. *pin_mux.h* header file

This file contains only prototypes of functions defined in the *pin_mux.c* source file.

Now everything is done. Clean and rebuild the project.

5. Revision history

This table lists the changes done to this document since the initial release:

Table 4. Revision history

Revision number	Date	Substantive changes
0	01/2016	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MKWSWPUG
Rev. 0
01/2016