

UM11346

Battery cell controller software driver for MC33771C and MC33772C v2.2

Rev. 3 — 18 December 2020

User manual

1 Introduction

This document describes how to install and use the Battery Cell Controller (BCC) software driver for MC33771C and MC33772C.

The battery cell controller software driver encapsulates the functionality of the MC33771C and MC33772C ICs for Serial Peripheral Interface (SPI) and Isolated Network High-speed Transceiver (TPL) communication modes. The driver acts as an API layer between the microcontroller low-level drivers (e.g. SDK) and your application. This allows you to perform functions such as setting registers and configuring daisy chains. The BCC software driver also includes functions for:

- Measuring voltages, current and temperatures
- Handling faults
- Implementing diagnostics
- Cell balancing
- Accessing fuse mirror and optional EEPROM, etc.

This driver supports and provides a high-level software solution for these analog parts:

- [MC33771C](#): 14-Channel Li-ion Battery Cell Controller IC
- [MC33772C](#): 6-Channel Li-ion Battery Cell Controller IC
- [MC33664](#): Isolated Network High Speed Transceiver

NXP offers the following board solutions based on MC33771C and MC33772C:

- [FRDM33771CSPEVB](#) (MC33771C EVB with SPI communication)
- [RD33771CDSTEVB](#) (MC33771C EVB with TPL communication)
- [FRDM33772CSPEVB](#) (MC33772C EVB with SPI communication)
- [KIT33772CTPLEVB](#) (MC33772C EVB with TPL communication)

The TPL interface requires an Isolated Network High-Speed Transceiver (MC33664).

NXP offers the following board solutions with Isolated Network High-Speed Transceiver:

- [FRDMDUAL33664EVB](#)
- [FRDM33664BEVB](#)

See the related user guides and data sheets listed in [Section 5 "References"](#) for detailed information.

There are two packages with BCC SW driver for MC33771C available: Full version and Lite version. Lite version is available directly from [nxp.com](#) web pages but does not provide an implementation of diagnostic functions. For more details, see [Section 4.2 "Downloading the software driver and example projects"](#).



2 MCU compatibility

2.1 Peripheral requirements

Peripherals and resource requirements critical to the MCU's ability to handle a given part are as follows:

SPI communication mode

- **SPI Module** is required for communication (MOSI, MISO, SCLK, CSB).
- **GPIO** is optionally required for the device RESET pin.
- **Interrupt** pin is optionally required for use with the FAULT pin – interrupt implementation is up to the user.

TPL communication mode

- **SPI Module in slave mode** is required for communication (DATA_RX, SCLK_RX, CSB_RX).
- **SPI Module in master mode** is required for communication (DATA_TX, SCLK_TX, CSB_TX).
- **GPIOs** are required for EN and INTB pins.
- **Interrupt** pin is optionally required for use with the FAULT pin and INTB pins – interrupt implementation is up to the user.

Depending on the user application, other resources might be required. See the example projects provided.

2.2 Supported devices

The battery cell controller software driver v2.2 supports the following NXP device:

- MC33771C: 14-Channel Li-ion Battery Cell Controller IC
- MC33772C: 6-Channel Li-ion Battery Cell Controller IC

2.3 Supported MCUs

Current implementation of the BCC software driver is generic, such that any suitable 32-bit microcontroller with SPI modules and enough number of GPIO pins can be utilized. See [Section 2.1 "Peripheral requirements"](#) for peripheral requirements.

The driver was tested with S32K144 MCU. An S32K1xx SDK 3.0.0 RTM with patch SR 3.0.2 was used for this purpose. The BCC software driver supports the following evaluation board in combination with S32K144 evaluation board:

- FRDM33771CSPEVB (MC33771C EVB with SPI communication)
- RD33771CDSTEVB (MC33771C EVB with TPL communication)
- FRDM33772CSPEVB (MC33772C EVB with SPI communication)
- KIT33772CTPLEVB (MC33772C EVB with TPL communication)

Note: *TPL-based setup requires an evaluation board with isolated network transceiver (MC33664). The S32K144 evaluation board is compatible with FRDM33772CSPEVB.*

2.4 Evaluation board settings

The boards listed in [Section 2.3 "Supported MCUs"](#) are directly compatible with the S32K144 evaluation board (S32K144EVB-Q100) and do not require any further modifications, e.g. any jumper settings.

For details about utilized S32K144 pins, see the user manual of FRDM33771CSPEVB or FRDM33772CSPEVB (SPI communication) or FRDM33772CSPEVB (TPL communication).

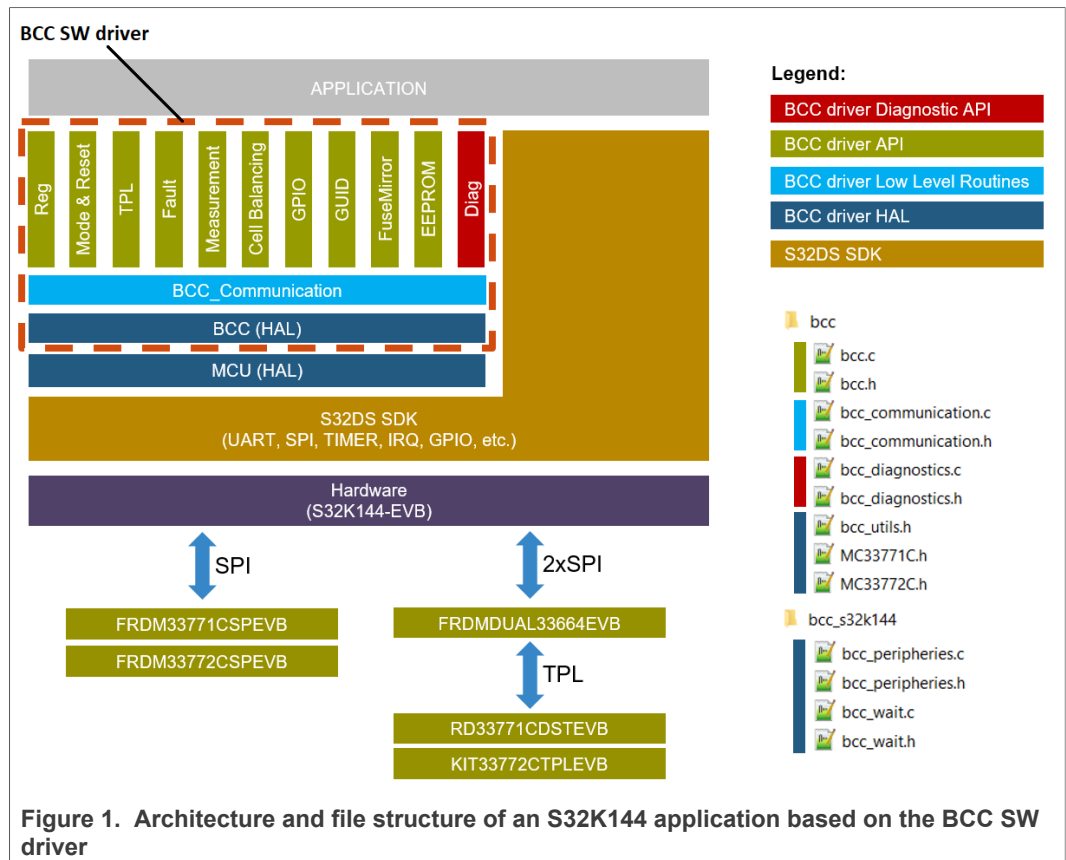
3 BCC software driver

This section provides an overview of the functionality, settings and usage of the driver (configuration and functions). For additional information, see the *API Programmer's Guide* [\[11\]](#) and the comments embedded in the source files.

Architecture of the BCC Software driver v2.2 is depicted in [Figure 1](#). The BCC Software Driver consists of 9 files.

- **bcc.c** and **bcc.h** contain the main functionality
- **bcc_communication.c** and **bcc_communication.h** files serve to support the SPI and TPL communication with the device
- **bcc_diagnostics.c** and **bcc_diagnostics.h** (not available in the Lite version) contain diagnostic functions
- **bcc_utils.c** provides macros for conversions of measurement results, voltage thresholds and other configurations.
- **MC33771C.h** and **MC33772C.h** contain register maps of the supported BCC devices

As the driver is not platform specific, several functions (marked as external in **bcc.h**) need to be implemented by the user. In the example projects, these functions are implemented in **bcc_s32k144** folder - in **bcc_peripherals.c** and **bcc_wait.c** files.



3.1 Configuring the driver

The configuration structures shown in [Figure 2](#) is the user interface available for configuring the driver and its behavior. This structure (`bcc_drv_config_t`) must be filled by appropriate values by the user before calling `BCC_Init` function. The same structure is then passed as a parameter to all API functions of the BCC driver (except `BCC_TPL_*` functions). This structure contains information about the about the communication mode (SPI, TPL), number of connected devices (1 for SPI, up to 63 for TPL), type of connected BCC devices (MC33771C, MC33772C), number of battery cells connected to each BCC device, chain setup (whether the last device in the TPL chain should be initialized for a loop-back mode), etc. Structure `bcc_drv_data_t` (also part of `bcc_drv_config_t` structure) contains internal driver data initialized automatically in the `BCC_Init` function.

As the driver is not platform specific, you need to implement several platform specific functions. See [Section 3.2 "Driver API"](#). `DrvInstance` member in `bcc_drv_config_t` structure is not modified by the driver itself. This member is passed to these user-defined functions in order to differentiate between BCC driver instances, when additional BCC devices are connected to the MCU in parallel (e.g. two separated TPL chains are set up with use of the FRDMDUAL33664EVB board). See the provided example project.

```

/*!
 * @brief Driver configuration.
 *
 * This structure contains all information needed for proper functionality of
 * the driver, such as used communication mode, BCC device(s) configuration or
 * internal driver data.
 */
typedef struct {
    uint8_t drvInstance;           /*!< BCC driver instance. Passed to the external functions
                                   defined by the user. */

    bcc_mode_t commMode;          /*!< BCC communication mode. */
    uint8_t devicesCnt;           /*!< Number of BCC devices. SPI mode allows one device only,
                                   TPL mode allows up to 63 devices. */

    bcc_device_t device[BCC_DEVICE_CNT_MAX]; /*!< BCC device type of
                                   [0] BCC with CID=1, [1] BCC with CID=2, etc. */
    uint16_t cellCnt[BCC_DEVICE_CNT_MAX]; /*!< Number of connected cells to each BCC.
                                   [0] BCC with CID=1, [1] BCC with CID=2, etc. */
    bool loopBack;                /*!< Loop back mode. If False, TPL_TX_Term. (RDTX_OUT) bit
                                   is set for the last BCC device in the TPL chain.
                                   This configuration item is ignored in SPI mode. */

    bcc_drv_data_t drvData;        /*!< Internal driver data. */
} bcc_drv_config_t;
    
```

Figure 2. Driver configuration

The SPI communication mode allows a single device only; no daisy chain is supported. The TPL mode allows up to 63 devices. The device number is based on the device's position in ascending order toward the end of the daisy chain. Each device should have 3 to 6 (MC33772C) or 7 to 14 (MC33771C) battery cells connected according to the data sheet of the specified battery cell controller.

Further configuration of the driver can be done at the top of bcc.h and bcc_diagnostics.h files (in the *User definitions* section). This configuration influences (among others) the byte-order of buffers that are passed to the platform specific functions BCC_MCU_TransferSpi and BCC_MCU_TransferTpl (implemented by the user) in order to perform SPI transfers.

The BCC SW driver can be utilized also to enable/disable a separate MC33664 device only (e.g. the second MC33664 in the loop-back HW setup, see provided example). Functions BCC_TPL_* are provided for this purpose. As soon as they do not communicate with any BCC, no bcc_drv_config_t structure is required in this case.

For a more detailed description of user configuration structure refer to the *API Programmer's Guide* [11].

3.2 Driver API

This BCC software driver provides an API that can be used for dynamic real-time configuration of a device in user code. For a summary of available functions, see [Table 1](#). As the BCC SW driver is platform independent, several functions need to be implemented by the user. These functions are marked as external in the table.

Table 1. Battery cell controller software driver API

Function	Description
BCC_Init	Initializes the battery cell controller device(s), assigns CID and initializes internal driver data.
BCC_SendNop	Sends No Operation command to the BCC device. It can be used to reset the communication timeout of the device without performing any operation.
BCC_Sleep	Sets sleep mode to all battery cell controller devices.
BCC_WakeUp	Sets normal mode to all battery cell controller devices.

Table 1. Battery cell controller software driver API...continued

Function	Description
BCC_SoftwareReset	Resets BCC device using software reset. It enters reset via SPI or TPL interface.
BCC_HardwareReset	Resets BCC device using RESET pin.
BCC_TPL_Enable	Enables MC33664 device (sets a normal mode). Intended for TPL mode only.
BCC_TPL_Disable	Sets MC33664 device into sleep mode. Intended for TPL mode only.
BCC_Reg_Read	Reads a value from addressed register (or desired number of registers) of selected battery cell controller device.
BCC_Reg_Write	Writes a value to addressed register of selected battery cell controller device.
BCC_Reg_WriteGlobal	Writes a value to addressed register of all configured BCC devices. Intended for TPL mode only.
BCC_Reg_Update	Updates content of a selected register; affects bits specified by a bit mask only.
BCC_Meas_StartConversion	Starts ADC conversion. It sets number of samples to be averaged and Start of Conversion bit in ADC_CFG register.
BCC_Meas_StartConversionGlobal	Starts ADC conversion for all devices in TPL chain. Intended for TPL mode only.
BCC_Meas_IsConverting	Checks status of conversion defined by End of Conversion bit in ADC_CFG register.
BCC_Meas_StartAndWait	Starts an on-demand conversion in selected BCC device and waits for completion.
BCC_Meas_GetRawValues	Reads the measurement registers and returns raw values.
BCC_Meas_GetCoulombCounter	Reads the Coulomb counter registers.
BCC_Meas_GetIsenseVoltage	Reads the ISENSE measurement and converts it to [μ V].
BCC_Meas_GetStackVoltage	Reads the stack measurement and converts it to [μ V].
BCC_Meas_GetCellVoltages	Reads the cell measurements and converts them to [μ V].
BCC_Meas_GetCellVoltage	Reads the voltage measurement of a selected cell and converts it to [μ V].
BCC_Meas_GetAnVoltages	Reads the voltage measurement for all ANx and converts them to [μ V]. Intended for ANx configured for absolute measurements only.
BCC_Meas_GetAnVoltage	Reads the voltage measurement of a selected ANx and converts it to [μ V]. Intended for ANx configured for absolute measurements only.
BCC_Meas_GetIcTemperature	Reads the BCC temperature and converts it to the selected unit.
BCC_Fault_GetStatus	Reads the status registers and returns raw values.
BCC_Fault_ClearStatus	Clears selected fault status register.
BCC_GPIO_SetMode	Sets the mode of one BCC GPIOx/ANx pin.
BCC_GPIO_ReadPin	Reads a value of one BCC GPIO pin.
BCC_GPIO_SetOutput	Sets output value of a BCC GPIO pin.
BCC_CB_Enable	Enables or disables the cell balancing via SYS_CFG1[CB_DRVEN] bit.
BCC_CB_SetIndividual	Enables or disables cell balancing for a specified cell and sets its timer.
BCC_CB_Pause	Can be used to manual pause cell balancing before on-demand conversion.
BCC_FuseMirror_Read	Reads a fuse mirror register.
BCC_FuseMirror_Write	Writes a fuse mirror register.
BCC_GUID_Read	Reads an unique serial number of the BCC device from the content of mirror registers.

Table 1. Battery cell controller software driver API...continued

Function	Description
BCC_EEPROM_Read	Reads a byte from specified address of EEPROM memory connected to BCC device via I2C bus.
BCC_EEPROM_Write	Writes a byte to specified address of EEPROM memory connected to BCC device via I2C bus.
Diagnostic functions (not included in Lite version of BCC software driver)	
BCC_Diag_ADC1	Implements the ADC1-A and ADC1-B functional verification.
BCC_Diag_OvUvVer	Implements OV/UV functional verification.
BCC_Diag_OvUvDet	Implements OV and UV detection.
BCC_Diag_CTxOpen	Implements CTx open detection and functional verification.
BCC_Diag_CellVolt	Implements Cell Voltage Channel functional verification.
BCC_Diag_ConnResistance	Detects a connector having an abnormally high contact resistance. It is a part of Cell Terminals and Cell Balancing Terminals leakage diagnostics.
BCC_Diag_CTxLeak	Detects a leakage current. It is a part of Cell Terminals and Cell Balancing Terminals leakage diagnostics.
BCC_Diag_CurrentMeas	Implements diagnostics of BCC internal resources for current measurement.
BCC_Diag_ShuntConn	Verifies whether the shunt resistor is properly connected to the current channel low-pass filter.
BCC_Diag_GPIOxOtUt	Implements GPIOx OT/UT functional verification.
BCC_Diag_GPIOxOpen	Implements GPIOx open terminal diagnostics.
BCC_Diag_CBxOpen	Implements Cell balance open load detection.
External functions (MCU/board specific)	
BCC_MCU_WaitMs	Waits for specified amount of milliseconds.
BCC_MCU_WaitUs	Waits for specified amount of microseconds.
BCC_MCU_StartTimeout	Starts a non-blocking timeout mechanism. After expiration of the time passed as a parameter, function BCCC_MCU_TimeoutExpired should signal with an expired timeout.
BCC_MCU_TimeoutExpired	Returns state of the timeout mechanism started by the function BCC_MCU_StartTimeout.
BCC_MCU_Assert	User implementation of assert.
BCC_MCU_TransferTpl	Sends and receives data to MC33664 via TX and RX SPI buses. This function is called only in the TPL mode.
BCC_MCU_TransferSpi	Sends and receives data via SPI bus. This function is called only in the SPI mode.
BCC_MCU_WriteCsbPin	Writes logic 0 or 1 to the CSB pin (SPI mode) or CSB_TX pin (TPL mode).
BCC_MCU_WriteRstPin	Writes logic 0 or 1 to the RESET pin. This function is called only in the SPI mode.
BCC_MCU_WriteEnPin	Writes logic 0 or 1 to the EN pin of MC33664. This function is called only in the TPL mode.
BCC_MCU_ReadIntbPin	Reads logic value of INTB pin of MC33664. This function is called only in the TPL mode.

For more detailed description of software driver API (function signatures, parameters) refer to the *API Programmer's Guide* [11] or to the comments embedded in the bcc.h and bcc_diagnostics.h files.

3.3 Required driver setup

In order to execute correctly, the BCC software driver requires the following:

- Initialize BCC driver configuration structure (`bcc_drv_config_t`). E.g. see `main.c` in the example projects.
- Check the macros in the section *User definitions* of `bcc/bcc.h` and `bcc/bcc_diagnostics.h`, especially `BCC_MSG_*` macros, which influence the byte order in SPI frames
- Implement the external functions (functions with `BCC_MCU_` prefix) declared in the *Platform specific functions* section of `bcc/bcc.h`

Moreover, the driver requires correctly pin-muxed pins and initialized peripherals (GPIO pins and SPI peripherals), which are handled by the external functions (`BCC_MCU_TransferSpi`, `BCC_MCU_WriteCsbPin`, etc.):

Peripherals configuration in SPI mode:

- **SPI:** Master mode, 48 bits/frame, MSB first, clock polarity: active high, clock phase: capture on the 2nd edge, active low chip select, clock frequency: up to 4 MHz
- **RST (optional):** GPIO output pin

Peripherals configuration in TPL mode:

- **SPI_RX:** Slave mode, 48 bits/frame, MSB first, clock polarity: active high, clock phase: capture on the 2nd edge, active low chip select
- **SPI_TX:** Master mode, 48 bits/frame, MSB first, clock polarity: active high, clock phase: capture on the 2nd edge, active low chip select, clock frequency 2 MHz, and further configuration (falling edge of `CSB_TX` to rising edge `SCLK_TX`, `SCLK_TX` LOW to `CSB_TX` HIGH, etc.) according to the MC33664 datasheet.
- **EN:** GPIO output pin, initial value: low
- **INTB:** GPIO input pin

For more details, refer to the provided example projects.

3.4 Implementation notes

This section describes how the BCC SW driver handles specific features of MC3377xC parts.

The BCC SW driver is based on variables of `bool`, `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t` and `int32_t` types and requires standard `stdbool.h`, `stdint.h` and `stddef.h` libraries.

MC33771C and MC33772C devices are highly similar devices in terms of their registers. Therefore, `MC33771C.h` and `MC33772C.h` files (containing the register maps) are also similar and most of the macros differ just in their prefix (`MC33771C_` vs `MC33772C_`). In order to make the driver code short and efficient, the driver uses macros dedicated to MC33771C for the register access to both devices in all cases where the macros are exactly the same.

3.4.1 Message counter handling

The message counter is a local counter in the MC3377xC devices. It is incremented for each new response transmitted by the BCC device. The BCC SW driver internally stores last received counter value and compares its increment to the newly received value.

Value of the BCC's internal counter doesn't have to be zero at the start-up. Synchronization with the internal data of the SW driver is done in the initialization (BCC_Init function).

4 Installing the software

This section describes installation of S32 Design Studio for ARM and shows how to use this SW driver with S32K144 and S32K1xx SDK for application development. A process of adding a BCC SW driver to an existing project in different IDEs or with use of different MCUs should be analogical. Most likely, the addition of low-level SDK drivers will vary.

4.1 Installing IDE

This procedure explains how to obtain and install S32 Design Studio for ARM (2018.R1 in this guide).

Note: The examples in the driver package are intended for S32 Design Studio for ARM 2018.R1. If the selected IDE is already installed on the system, skip this section.

1. Obtain the latest S32 Design Studio for ARM 2018.R1 installer file from the NXP website here: <http://www.nxp.com/S32DS>
2. Run the executable file and follow the instructions.
3. Download Update 11 (or newer) file for S32 Design Studio for ARM 2018.R1 from the NXP website <http://www.nxp.com/S32DS>. This update contains S32K1xx SDK 3.0.0 RTM with patch SR 3.0.2.
4. In S32 Design Studio, install Update 11 via **Help > Install New Software > Add > Archive**.

4.2 Downloading the software driver and example projects

To download the latest version of the BCC software driver and the example projects:

1. Acquire the desired BCC software driver and example projects
 - a. For the Full version
 - i. Go to NXP DocStore: <https://www.docstore.nxp.com/>
 - ii. Register and request access to Automotive Battery Management
 - iii. Download the SW driver for MC33771C and MC33772C under Automotive Battery Management / MC33771 MC33772 MC33664 / Software Enablement
 - b. For the Lite version
 - i. Go to the Downloads section of the SW driver web page at <http://www.nxp.com/EMBEDDED-SW-MC33771C>
 - ii. Download BCC_SDK_SW.zip
2. Unzip the downloaded file and check to see that the folder contains the files listed in [Table 2](#).

Note: The Lite version does not contain diagnostic functions. See [Table 1](#). Furthermore, BCC_S32K144_Monitoring_Diagnostics example is replaced by BCC_S32K144_Monitoring example, which is highly similar but does not contain diagnostic features.

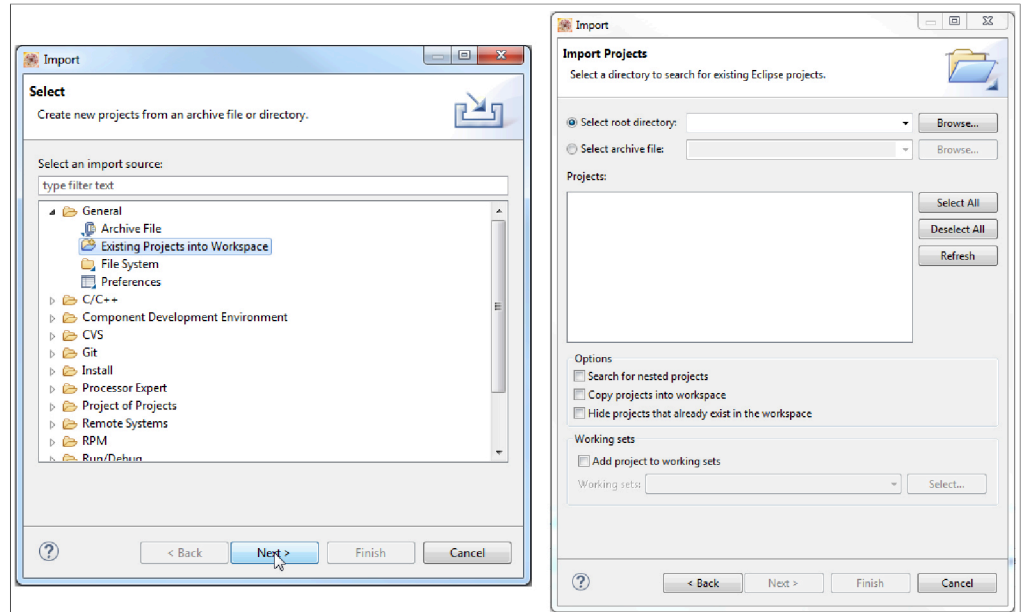
Table 2. Content of the downloaded zip file

Folder Name	Folder Contents
BCC_SW_Driver	This is a battery cell controller SW driver folder.
Programmer_Guide	This folder contains detailed API documentation of this SW driver.
S32DS_Examples	This is the example projects folder for S32 Design Studio for ARM Version 2018.R1 with update 11. All provided examples are determined for S32K144EVB-Q100 MCU board.
BCC_S32K144_DualChain	This project demonstrates how to use the BCC SW driver in order to communicate with two TPL chains connected in parallel. One FRDM33771CSPEVB Isolated Network High Speed Transceiver board and two RD33771CDSTEBV or two KIT33772CTPLEVB BCC boards are required.
BCC_S32K144_FreeMASTER	This project demonstrates how the battery cell controller SW driver can be used in conjunction with the FreeMASTER visualization tool. The FRDM33771CSPEVB board (SPI communication) or FRDM33771CSPEVB and RD33771CDSTEBV board (TPL communication) are required.
BCC_S32K144_LoopBack	This project demonstrates how to use the BCC SW driver in case of the loop-back HW setup. One FRDM33771CSPEVB Isolated Network High Speed Transceiver board and two RD33771CDSTEBV or two KIT33772CTPLEVB BCC boards are required.
BCC_S32K144_Monitoring_Diagnostics	This project demonstrates how to read measurements and fault status information of a BCC device. This project also shows how to use diagnostic functions and evaluate their results using the battery cell controller SW driver. One of the following BCC boards setup is required: <ul style="list-style-type: none"> • FRDM33771CSPEVB (MC33771C with SPI communication) or • FRDM33772CSPEVB (MC33772C with SPI communication) or • FRDM33771CSPEVB + RD33771CDSTEBV (MC33771C with TPL communication) or • FRDM33772CSPEVB + KIT33772CTPLEVB (MC33772C with TPL communication)
CHANGELOG.txt	Change log file
LICENSE.txt	License file

4.3 Import an example project into IDE

The following steps show how to import an example from the downloaded zip file into S32 Design Studio for ARM.

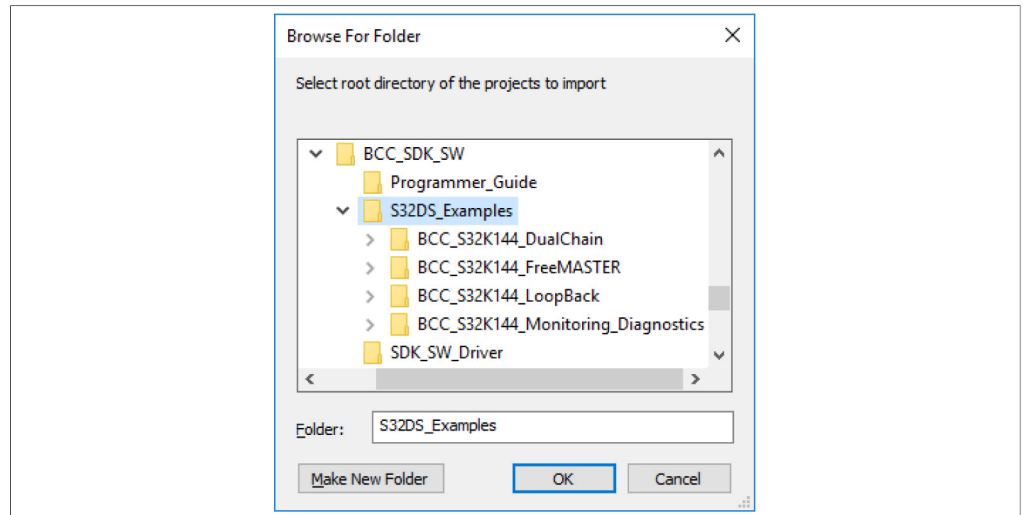
1. In S32 Design Studio menu bar, click **File** → **Import...** In the pop-up window, click **General** → **Existing Projects into Workspace** and click **Next** . See [Figure 3](#).



aaa-031226

Figure 3. Importing an example project (a)

2. Click **Browse** and locate the folder where you unzipped the downloaded example files. Find the folder S32DS_Examples and select a project to import, see [Figure 4](#). Then click **OK**.



aaa-036171

Figure 4. Importing an example project (b)

- With your project now loaded in the **Select root directory** box, click **Copy projects into workspace** checkbox. Then click **Finish**. The project is now in the S32 Design Studio workspace where you can build and run it. See [Figure 5](#).

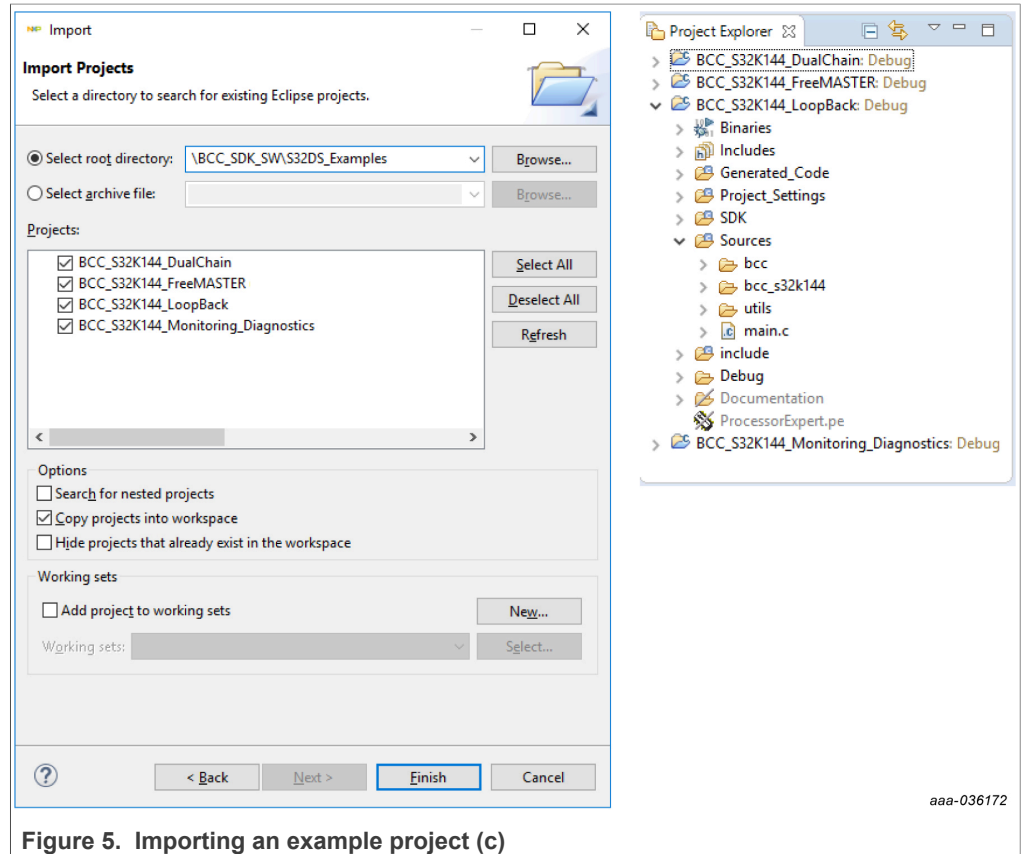


Figure 5. Importing an example project (c)

For additional information about the examples, refer to the PDF documentation inside each example project folder.

4.4 Creating a new project with BCC SW driver

If you choose to not use the example projects, the following instructions describe how to create and set up a new project for S32K144 MCU that uses BCC SW driver.

To create a new project in S32 Design Studio for ARM:

1. In the S32 Design Studio menu bar, click **File > New > S32DS Application Project**. See [Figure 6](#).

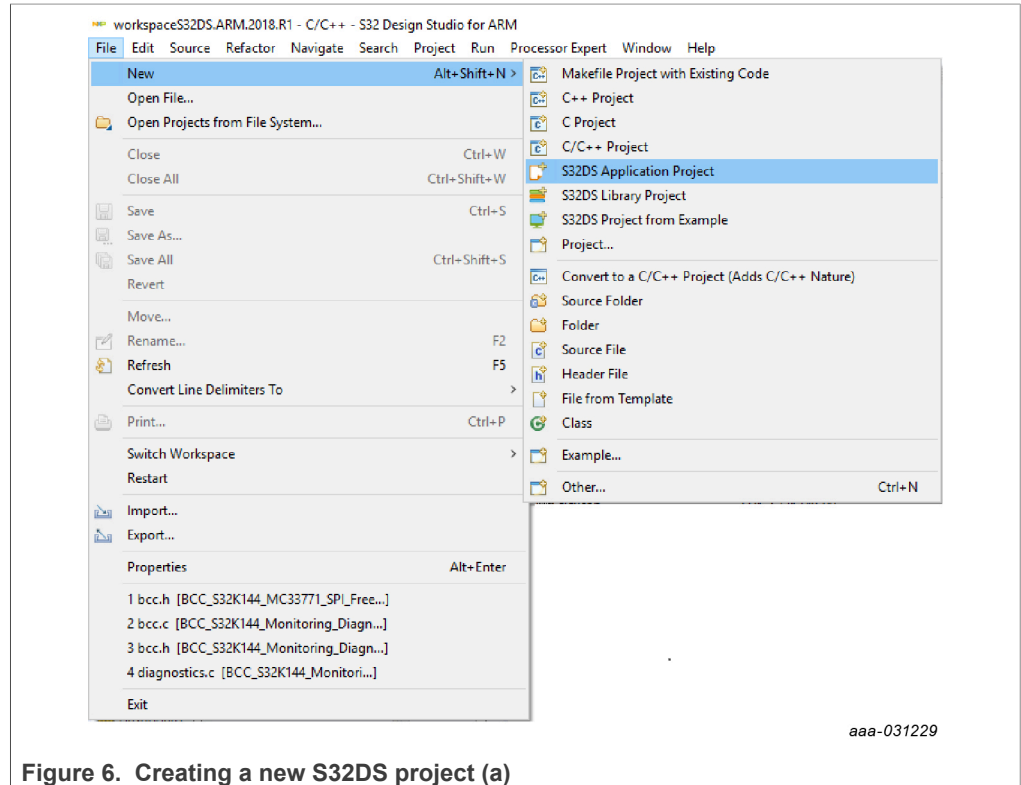


Figure 6. Creating a new S32DS project (a)

Battery cell controller software driver for MC33771C and MC33772C v2.2

- 2. When the **S32DS Application Project** box opens, enter a project name into the text box, choose **S32K144** processor in the **Processors** tab and click **Next**. See [Figure 7](#).

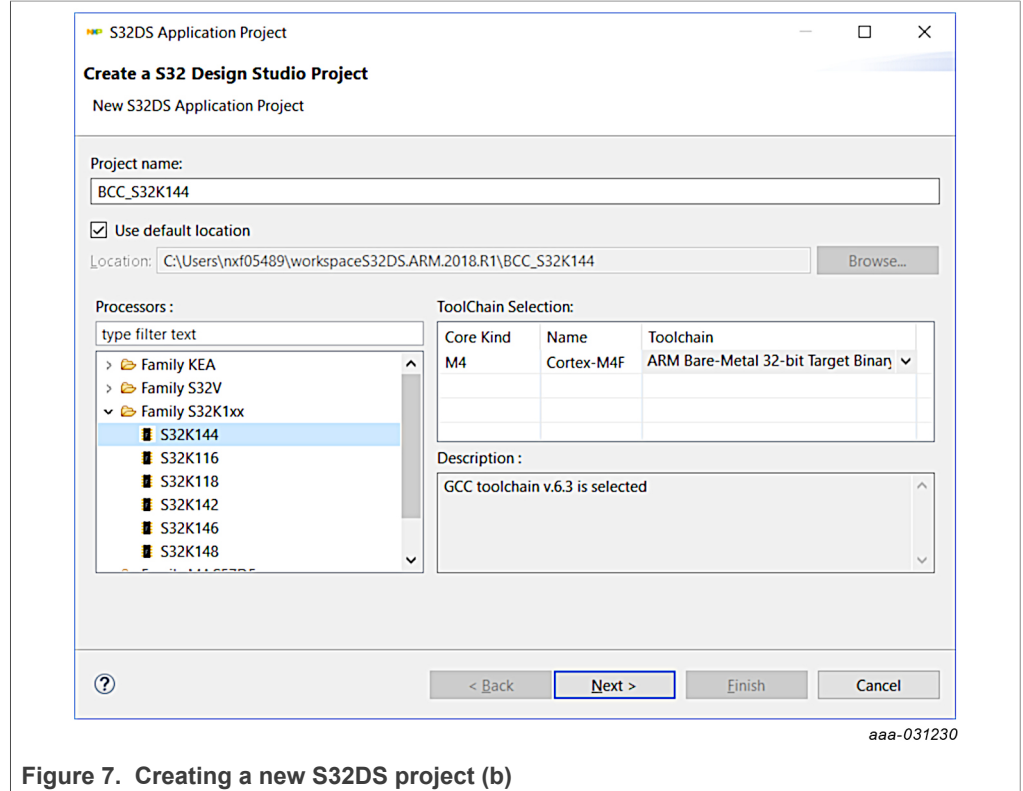


Figure 7. Creating a new S32DS project (b)

- 3. Click **NewLib Nano** library, **S32K144_SDK (version 3.0.0)** SDK and click **Next**. See [Figure 8](#).

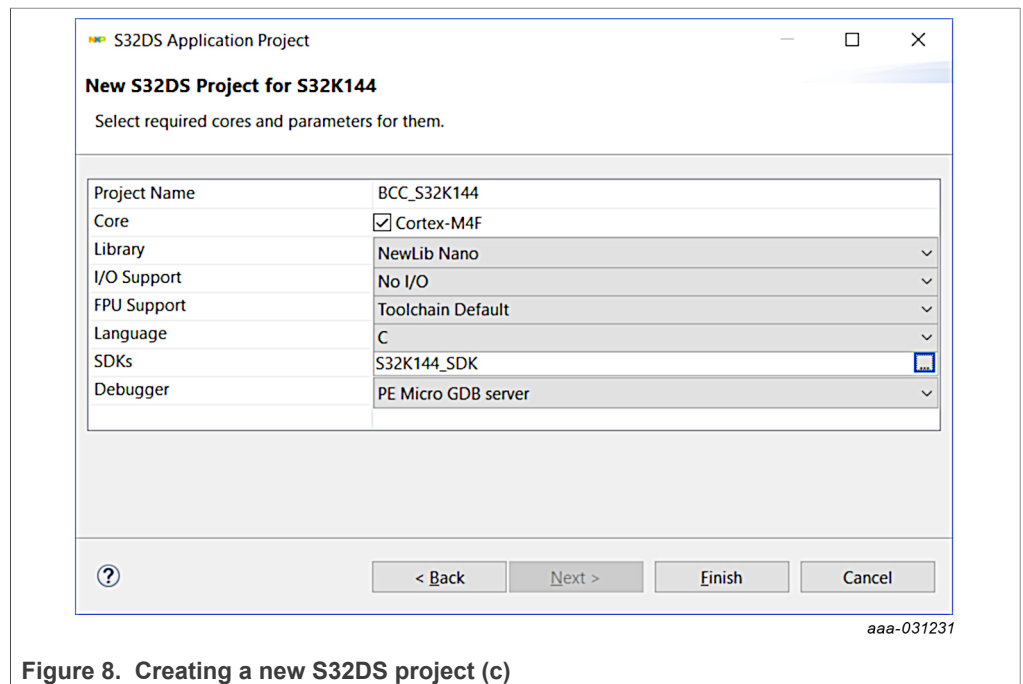


Figure 8. Creating a new S32DS project (c)

- Figure 9 shows the Project Explorer panel and a part of the `main.c` content after the creation of a new project. The project includes only startup code and minimal driver set from S32K144 SDK.

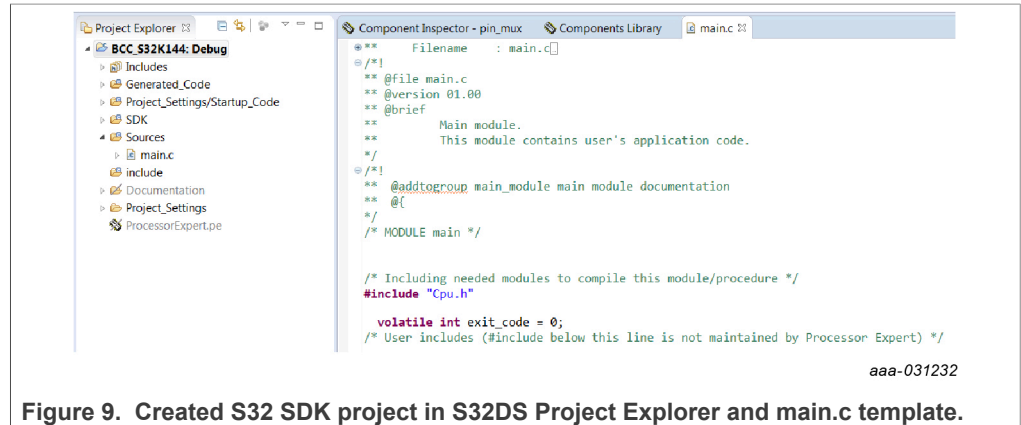


Figure 9. Created S32 SDK project in S32DS Project Explorer and main.c template.

4.4.1 Adding the BCC software driver to the project

This section describes how to add the BCC software driver to the project.

- Copy the content of BCC_SW_Driver to the Sources folder in your newly created project. See Figure 10 for a list of copied files (Lite version of this driver).

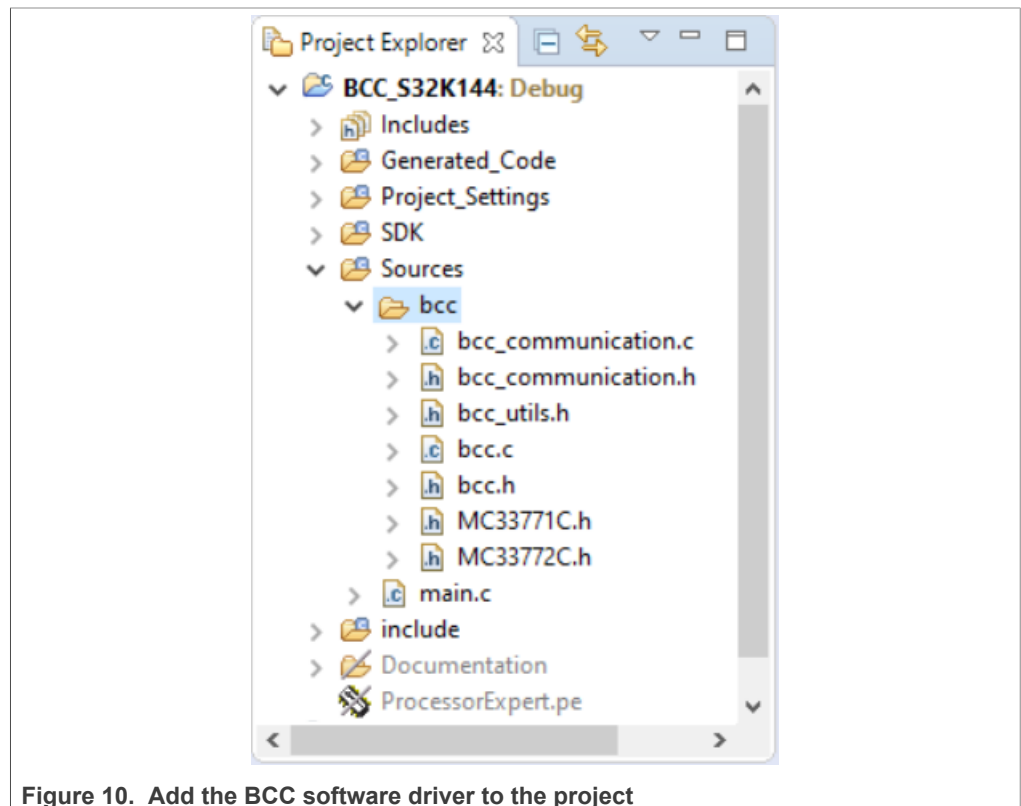


Figure 10. Add the BCC software driver to the project

2. Include the bcc.h header file in **main.c** in order to get access to the BCC software driver in user code. See [Figure 11](#). In order to get access to the diagnostic functions of BCC driver, include also **bcc_diagnostics.h** file.

```

/* Including needed modules to compile this module/procedure */
#include "Cpu.h"

volatile int exit_code = 0;
/* User includes (#include below this line is not maintained by Processor Expert) */
#include "bcc/bcc.h"

/*!
 \brief The main function for the project.
 */
    
```

aaa-031234

Figure 11. Link BCC software driver

4.4.2 Setting up the project

Once the new project has been created and the BCC software driver has been added into it, the project must be set up.

1. In order to implement the platform specific (external) functions of the BCC software driver, LPSPI and GPIO S32K1xx SDK drivers are required. The GPIO driver is already attached to the project. In order to generate the LPSPI driver into the project, double-click **lpspi** in the **Components Library** window. See [Figure 12](#). If the **Components Library** window is hidden, open it by clicking **Processor Expert > Show Views**. When the **lpspi** component is included in the PEx file, it is shown in the **Components** window. See [Figure 12](#).

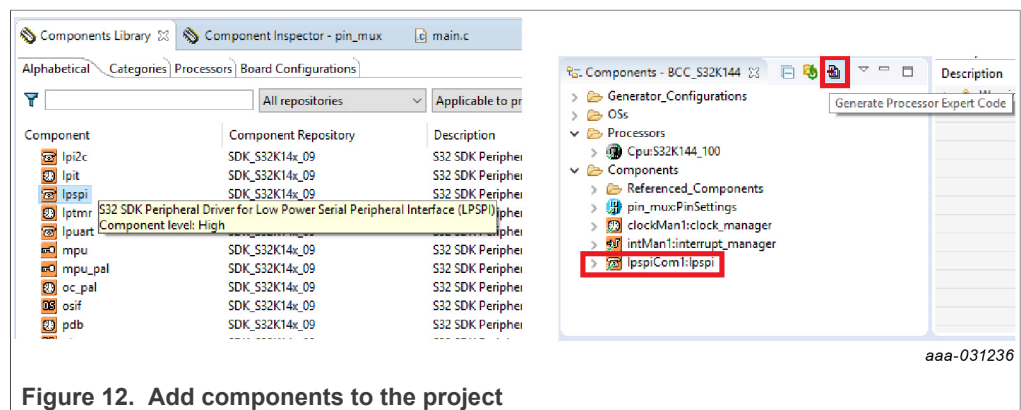


Figure 12. Add components to the project

2. In order to get the BCC software driver to run on S32K144, the **PinSettings** component must be edited to configure the LPSPI and GPIO pins being used. This entails making the correct MCU pin selections and then muxing them as needed. An

example of LPSPI0 pin muxing is shown in [Figure 13](#). You should also set the correct GPIO pin directions and initial values in the **PinSettings** component.

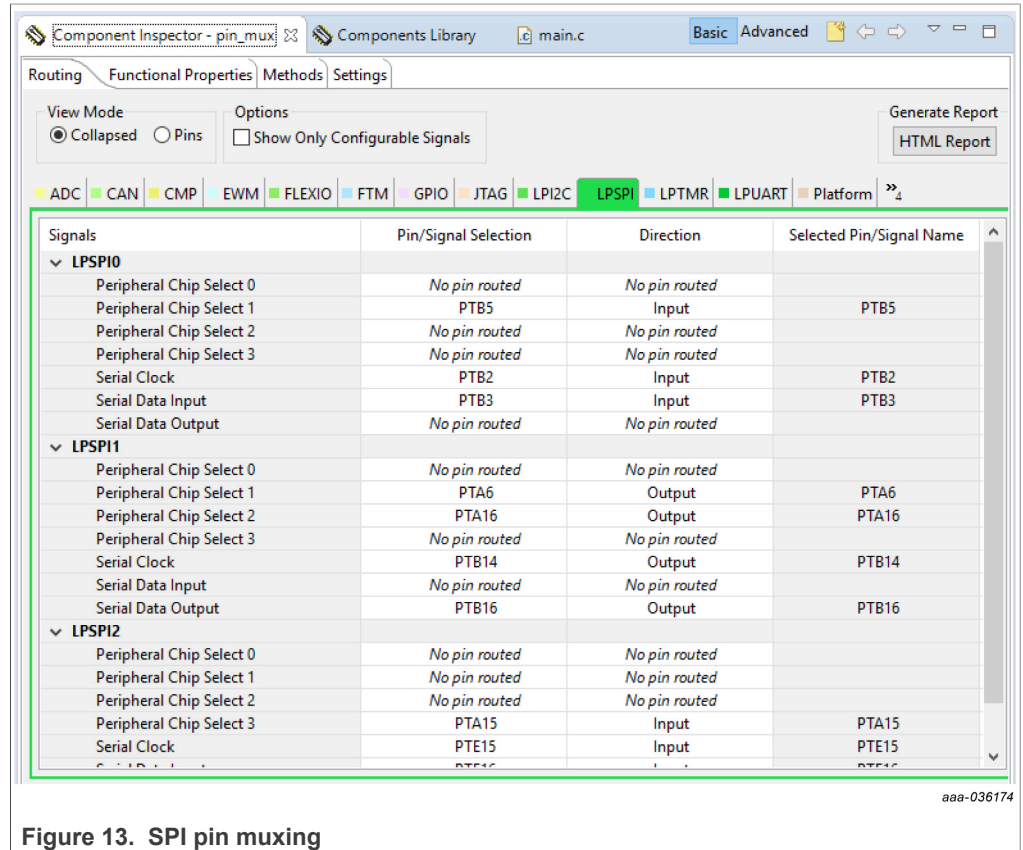


Figure 13. SPI pin muxing

- In order to get the BCC software driver to run on S32K144, check in the **clock_manager** component settings to see if the peripheral clock to LPSPI and PORT peripherals are enabled. The clock also must provide frequency in an allowed range for these peripherals. See [Figure 14](#) for the location of peripheral clock settings.

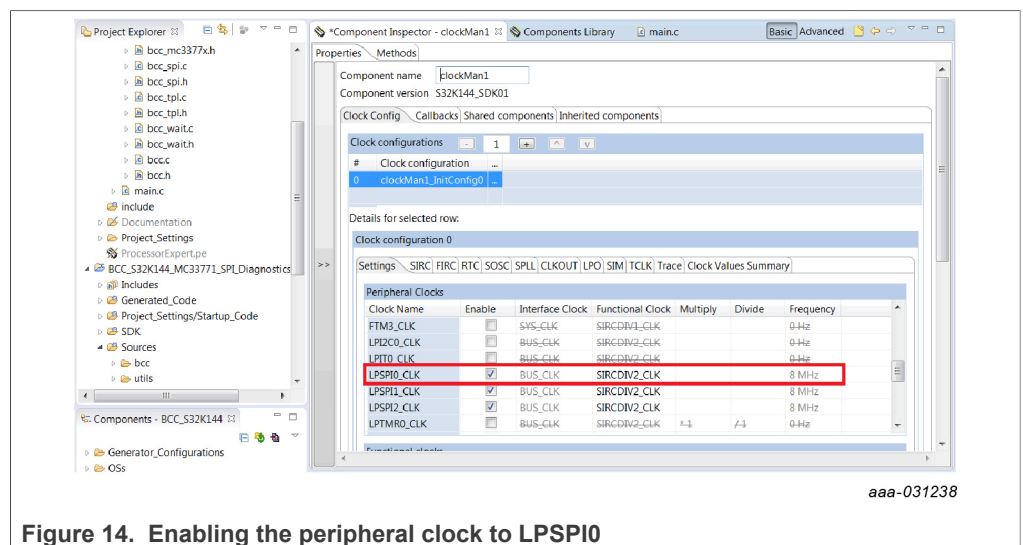


Figure 14. Enabling the peripheral clock to LPSPI0

4. In the **Components** window, click the **Generate Processor Expert Code** icon to generate the component settings into `Generated_Code` folder. See [Figure 12](#).
5. In order to set the clock configuration and mux the pins according to the settings generated from Processor Expert components, add following lines at the beginning of **main()** function:


```
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
```
6. Create a variable of type **bcc_drv_config_t** that will be passed to all used functions. This variable stores BCC software driver configuration and its internal data. This variable must be accessible during run-time and should be declared either in the **main()** function or as a global variable.
7. Configure the BCC as shown in [Figure 15](#). You can change all the individual items as needed.

```
int main(void)
{
    /* Write your local variable definition here */
    bcc_drv_config_t drvConfig = {0}; /* Driver configuration. */

    /** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
    #ifdef PEX_RTOS_INIT
        PEX_RTOS_INIT(); /* Initialization of the selected RTOS. Macro is defined
    #endif
    /** End of Processor Expert internal initialization. */

    /* Write your code here */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
        g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

    /* Pin-muxing. */
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

    /** Driver configuration. */
    drvConfig.drvInstance = 0U;
    drvConfig.commMode = BCC_MODE_SPI;
    drvConfig.devicesCnt = 1U;
    drvConfig.device[0] = BCC_DEVICE_MC33771C;
    drvConfig.cellCnt[0] = 14U;
    drvConfig.loopBack = false;

    /* Initialize used peripherals */

    /* Initialize device. */
    BCC_Init(&drvConfig);
}
```

Driver configuration

Peripherals initialization

Device initialization

Figure 15. Set up BCC

8. Set up the LPSPI peripherals that will be used with the BCC software driver. The easiest way is to set the LPSPI configuration in the **lpspi** Processor Expert component. After clicking the **Generate Processor Expert Code** icon, the **lpspi** configuration is generated in the `Generated_Code` folder. This configuration can be passed as a parameter of **LPSPI_MasterInit** (or **LPSPI_SlaveInit**) SDK function in **main()** function.

9. The BCC initialization function should be called. The user must pass reference to the driver configuration structure. Then, it is recommended to initialize the BCC configuration registers and clear the faults. See the provided examples.
10. Besides the peripheral and BCC initialization function, the external functions, listed in `bcc.h` and [Table 1](#), used by the BCC driver need to be implemented. See the `bcc_s32k144` folders in the provided examples projects as examples of their implementation for S32K144.

4.4.3 Writing your application code

When the BCC SDK SW driver is configured properly, you can use all of the prepared functions to construct your own application.

See the *API Programmer's Guide* [8] for function signatures and required parameters. Also, review the `Sources/bcc/bcc.h` and `Sources/bcc/bcc_diagnostics.h` header files, which contains prototypes for all available functions.

For more complex BCC use cases, refer to the provided example projects.

4.4.4 Compiling, downloading and debugging

To compile a project, click the compile icon in the toolbar. See [Figure 16](#).

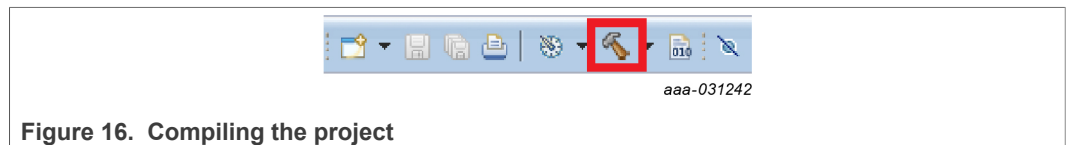


Figure 16. Compiling the project

The process for downloading an application on a board in S32 Design Studio for ARM may differ according to the selected MCU board. If you have questions, see the S32 Design Studio for ARM user's guide. To download and debug on an S32K144EVB-Q100 MCU board:

1. Click the arrow next to the debug icon in the toolbar and click **Debug Configurations...**. See [Figure 17](#).

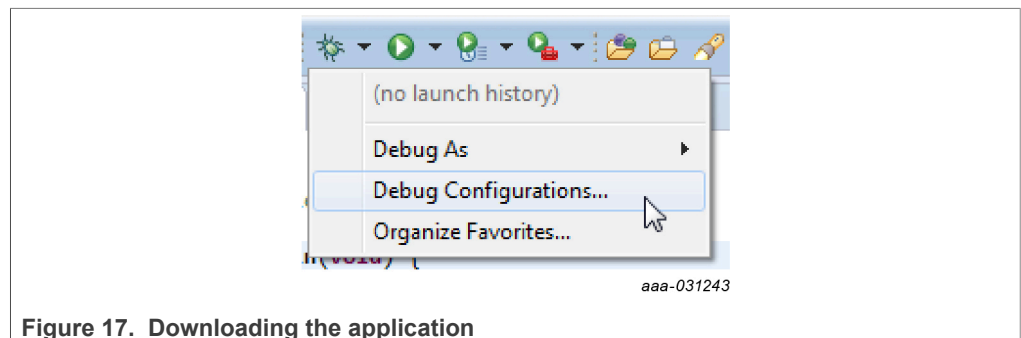


Figure 17. Downloading the application

2. In the **Debug Configurations** dialog box, select one of the existing configurations with a project name under **GDB PEMicro Interface Debugging**.
3. Make sure that **C/C++ Application** contains a path to the `.elf` file of the project in the **Main** tab of **Debug Configuration** window.
4. Choose a proper **debug interface** and **USB port** in the **Debugger** tab of the **Debug Configuration** window.
5. Apply changes and then click **Debug**. S32 Design Studio for ARM will download and launch the program on the board.

5 References

- [1] Product summary page for MC33771C: 14-Channel Li-ion Battery Cell Controller IC
<https://www.nxp.com/MC33771C>
- [2] Product summary page for MC33772C: 6-Channel Li-ion Battery Cell Controller IC
<https://www.nxp.com/MC33772C>
- [3] Product summary page for MC33664: Isolated Network High-Speed Transceiver
<https://www.nxp.com/MC33664>
- [4] Tool summary page for FRDM33664BEVB: Evaluation Board for MC33664ATL Isolated Network High-Speed Transceiver
<https://www.nxp.com/FRDM33664BEVB>
- [5] Tool summary page for FRDMDUAL33664EVB: Evaluation Board with two MC33664ATL Isolated Network High-Speed Transceivers
<https://www.nxp.com/FRDMDUAL33664EVB>
- [6] Tool summary page for FRDM33771CSPEVB: Evaluation Board for MC33771C with SPI Communication
<https://www.nxp.com/FRDM33771CSPEVB>
- [7] Tool summary page for FRDM33772CSPEVB: Evaluation Board for MC33772C with SPI Communication
<https://www.nxp.com/FRDM33772CSPEVB>
- [8] Tool summary page for KIT33772CTPLEVB: Evaluation Board for MC33772C with Isolated Daisy Chain Communication
<https://www.nxp.com/KIT33772CTPLEVB>
- [9] Tool summary page for RD33771CDSTEVB: Evaluation Board for MC33771C with Isolated Daisy Chain Communication
<https://www.nxp.com/RD33771CDSTEVB>
- [10] Tool summary page for Embedded SW: Battery Cell Controller Software Driver for MC33771C and MC33772C
<http://www.nxp.com/EMBEDDED-SW-MC33771C>
- [11] Tool summary page for S32 Design Studio IDE
<http://www.nxp.com/S32DS>
- [12] API programmer's guide — included in the BCC software driver zip file

6 Revision history

Revision history

Rev	Date	Description
3	20201218	Updates related to new version (2.2) of the driver. The new driver supports MC33772C device, uses a new format of register maps, extends the API functions and brings other improvements. All changes are listed in CHANGELOG.txt file inside the driver package.
2	20200427	Updates related to new version (2.1) of the driver. The new driver supports SPI communication mode. <ul style="list-style-type: none">• Section 1: added a link to FRDM33771CSPEVB• Section 2.1: added requirements for SPI communication mode• Section 2.3: added FRDM33771CSPEVB (SPI communication)• Section 3: updated BCC software driver version (replaced 2.0 by 2.1) and updated Figure 1• Section 3.1: updated Figure 2 and added "The SPI communication mode allows a single device only; no daisy chain is supported"• Table 1: updated description and added new rows for BCC_HardwareReset, BCC_MCU_TransferSpi, and BCC_MCU_WriteRstPin• Section 3.3: added description for configuration in SPI mode• Section 4.2: updated procedure and folder contents in Table 2• Section 4.4.2: updated Figure 15
1	20191219	Initial release

7 Legal information

7.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Suitability for use in automotive applications — This NXP Semiconductors product has been qualified for use in automotive applications. Unless otherwise agreed in writing, the product is not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out of the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US\$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

7.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Tab. 1.	Battery cell controller software driver API	5	Tab. 2.	Content of the downloaded zip file	10
---------	---	---	---------	--	----

Figures

Fig. 1.	Architecture and file structure of an S32K144 application based on the BCC SW driver	4	Fig. 9.	Created S32 SDK project in S32DS Project Explorer and main.c template.	15
Fig. 2.	Driver configuration	5	Fig. 10.	Add the BCC software driver to the project	15
Fig. 3.	Importing an example project (a)	11	Fig. 11.	Link BCC software driver	16
Fig. 4.	Importing an example project (b)	11	Fig. 12.	Add components to the project	16
Fig. 5.	Importing an example project (c)	12	Fig. 13.	SPI pin muxing	17
Fig. 6.	Creating a new S32DS project (a)	13	Fig. 14.	Enabling the peripheral clock to LPSPiO	17
Fig. 7.	Creating a new S32DS project (b)	14	Fig. 15.	Set up BCC	18
Fig. 8.	Creating a new S32DS project (c)	14	Fig. 16.	Compiling the project	19
			Fig. 17.	Downloading the application	19

Contents

1	Introduction	1
2	MCU compatibility	2
2.1	Peripheral requirements	2
2.2	Supported devices	2
2.3	Supported MCUs	2
2.4	Evaluation board settings	3
3	BCC software driver	3
3.1	Configuring the driver	4
3.2	Driver API	5
3.3	Required driver setup	8
3.4	Implementation notes	8
3.4.1	Message counter handling	8
4	Installing the software	9
4.1	Installing IDE	9
4.2	Downloading the software driver and example projects	9
4.3	Import an example project into IDE	10
4.4	Creating a new project with BCC SW driver	12
4.4.1	Adding the BCC software driver to the project	15
4.4.2	Setting up the project	16
4.4.3	Writing your application code	19
4.4.4	Compiling, downloading and debugging	19
5	References	20
6	Revision history	21
7	Legal information	22

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© NXP B.V. 2020.

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 December 2020

Document identifier: UM11346