Model-Based Design Toolbox Battery Management Systems

Quick Start Guide

Automatic Code Generation for Battery Cell Controllers
Version 1.2.0

Target-Based Automatic Code Generation Tools
For MATLAB™/Simulink™/Stateflow™ Models working with Simulink Coder ™ and Embedded Coder®

Summary

1 Installation

Installing the Model-Based Design Toolbox is the first step in setting up and running automatic C code generation from MATLAB/Simulink for NXP's battery cell controllers and development boards.

1.1 System Requirements

For a flawless development experience the minimum recommended PC platform is:

- *Windows® OS*: any x64 processor
- At least 4 GB of RAM
- At least 6 GB of free disk space.
- Internet connectivity for web downloads.

Operating System Supported

	SP Level	64-bit
Windows 10		X
Windows 11		X

1.2 Software Requirements

MBDT for BMS Requires the following NXP Software product:

- Model-Based Design Toolbox for S32K3xx 1.4.x

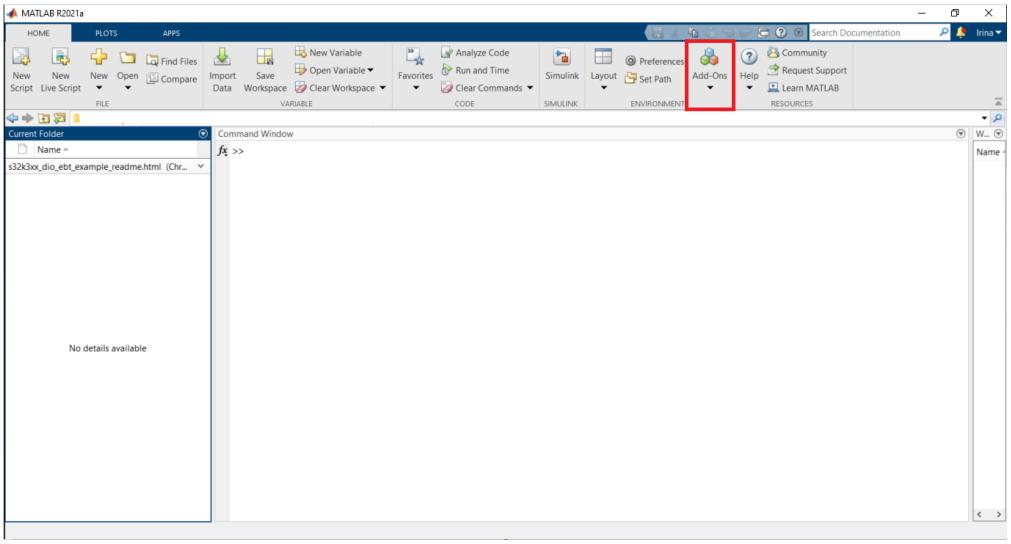
1.3 Installation Steps

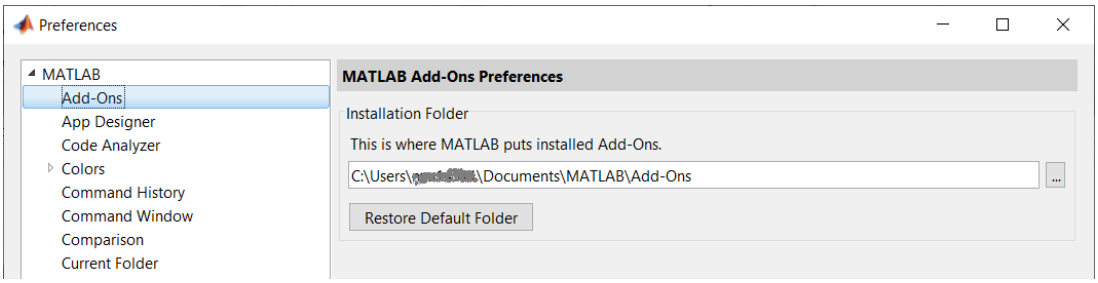
NXP's Model-Based Design Toolbox is delivered as a MATLAB Toolbox Package that can be installed offline or online from MathWorks Add-ons.

1.3.1 Install NXP Support Package for BMS

This package guides you through the download, installation, and activation process of the MBDT for BMS. For demonstration purposes only, the 1.0.0 version has been used throughout the following steps.

a. Go to MATLAB Add-Ons





The default location can be changed before installation by modifying the Add-Ons path from MATLAB Preferences.

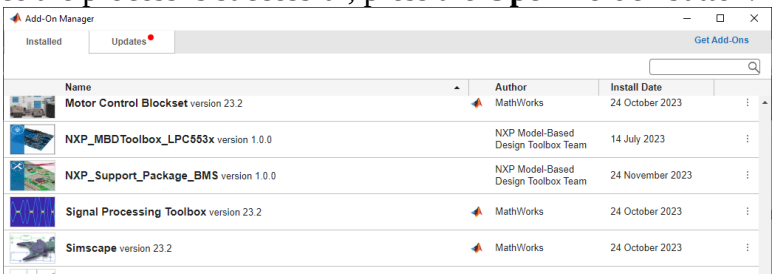
Note: It is strongly recommended to install the MATLAB and NXP Toolbox into a location that does not contain special characters, empty spaces, or mapped drives. Also, consider choosing a short path like C:/*MATLABAddOns.*

b. Search for the “NXP Support Package BMS”.

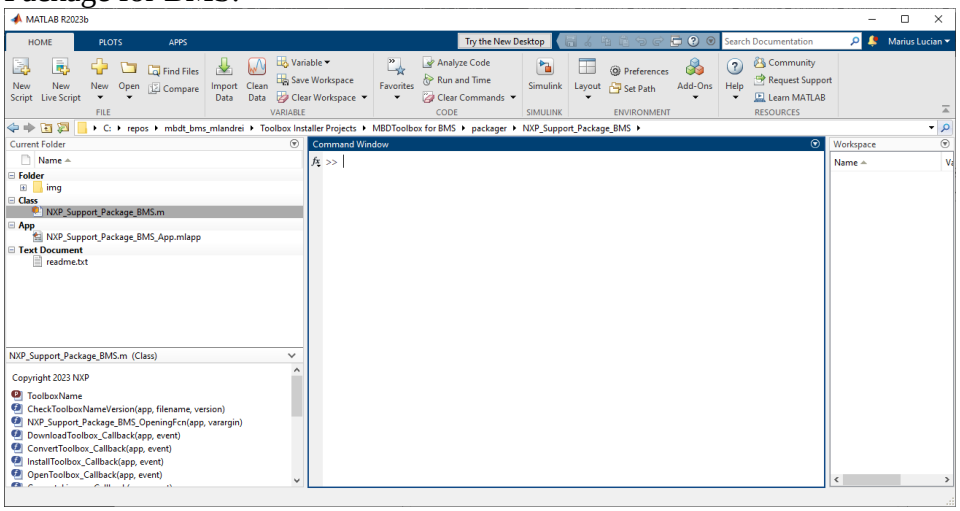
c. Install the “NXP Support Package BMS” by pressing the **Add** button.

d. Read the License Agreement and press **I Accept.**

e. Once the process is successful, press the **Open Folder** button.



f. Run the **NXP_Support_Package_BMS.m** script to start the NXP Support Package for BMS.



1.3.2 Install NXP Model-Based Design Toolbox for BMS

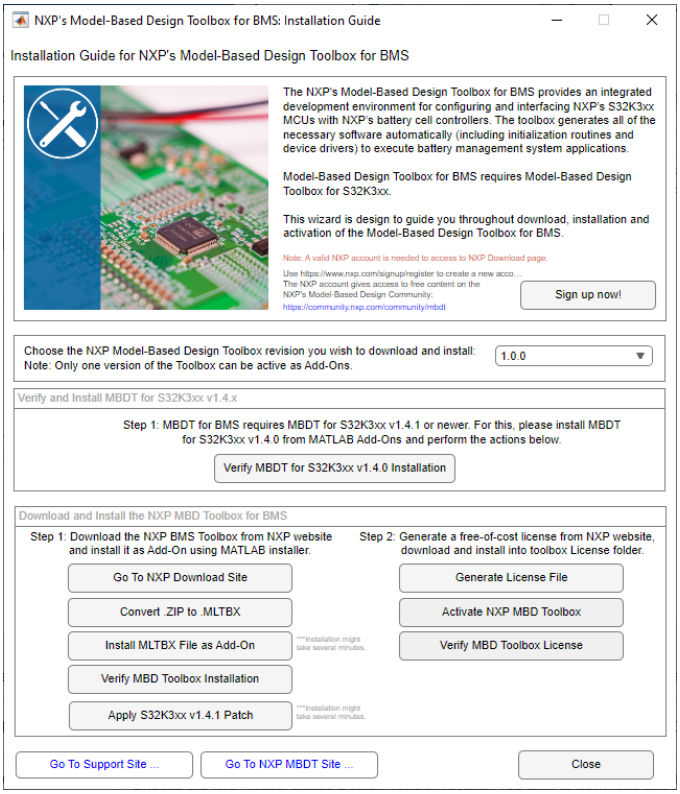
NXP Support Package for BMS is a graphical user interface guide that helps to download and install the Model-Based Design Toolbox and also generates and installs the free-of-cost license from the NXP website.

1. Verify MBDT for S32K3xx v1.4.x Installation

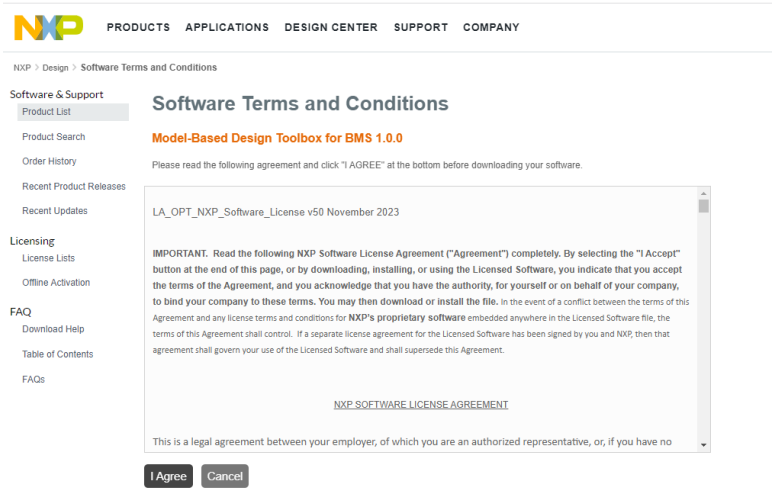
   MBDT for BMS 1.2.0 requires MBDT for S32K3xx 1.4.0, 1.4.1 **or** 1.4.2 to be installed and configured in the current MATLAB instance. Press **Verify MBDT for S32K3xx v1.4.0 Installation.**

   If MBDT for S32K3xx is not installed or you have a different version than 1.4.x, please proceed to install first the MBDT for S32K3xx 1.4.0!

2. Press **Go to NXP Download Site** Button. In the newly opened window, Review the Terms and Conditions as you scroll down, and press the **I Agree** Button.



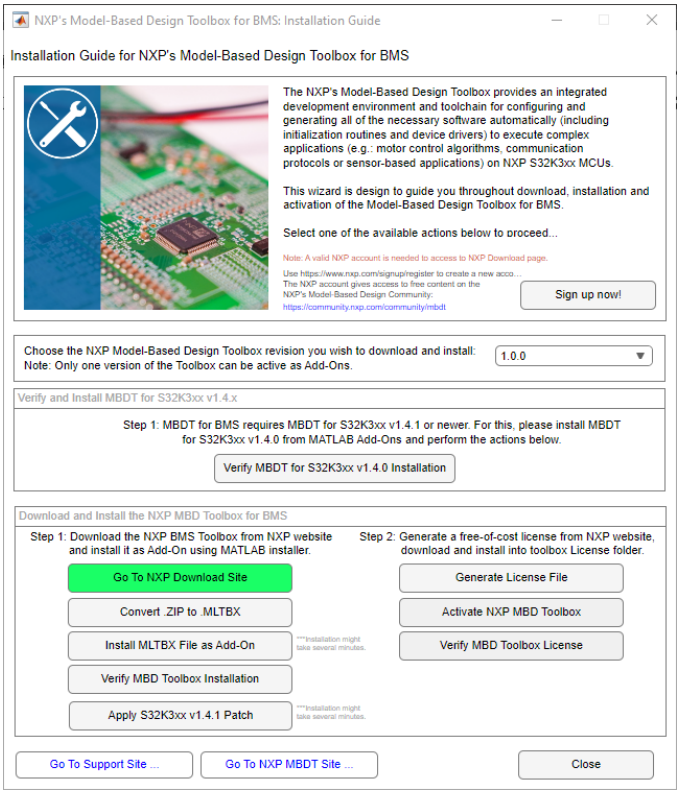
Note: If the page is not displayed as below, please go to the location presented in chapter *1.2.5*. After selecting **NXP Software** go to **Automotive SW - Model-Based Design Toolbox** and select the latest release available.



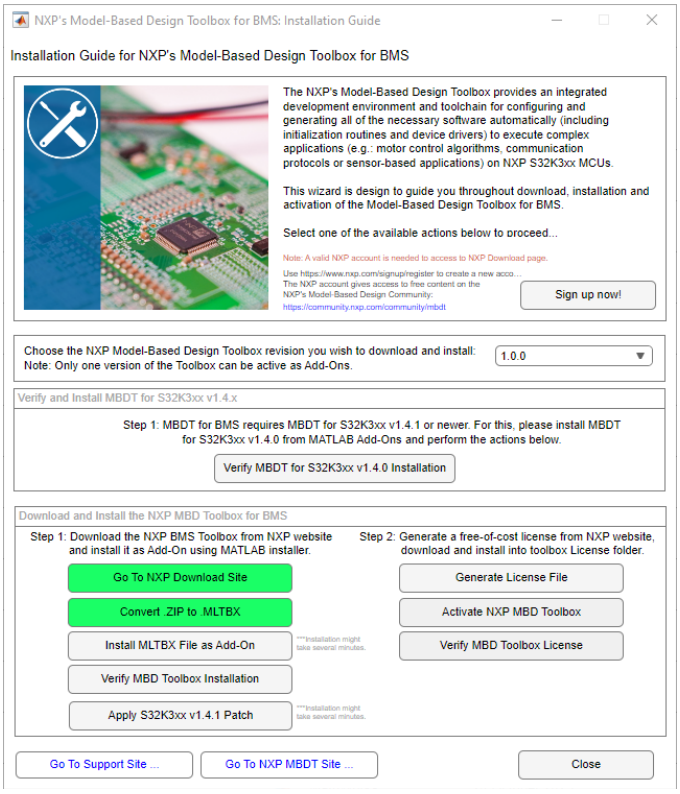
3. Download the **SW32_MBDT_BMS_1.2.0_DYYMM.mltbx** file.

   Note: The downloaded file has the **.zip** extension instead of **.mltbx**. The next step helps to convert to the right format.

4. Go back to **NXP Support Package for BMS** and press the **Convert .ZIP to .MLTBX** button. In the newly opened Browsing window, select the file downloaded and press **Open.**

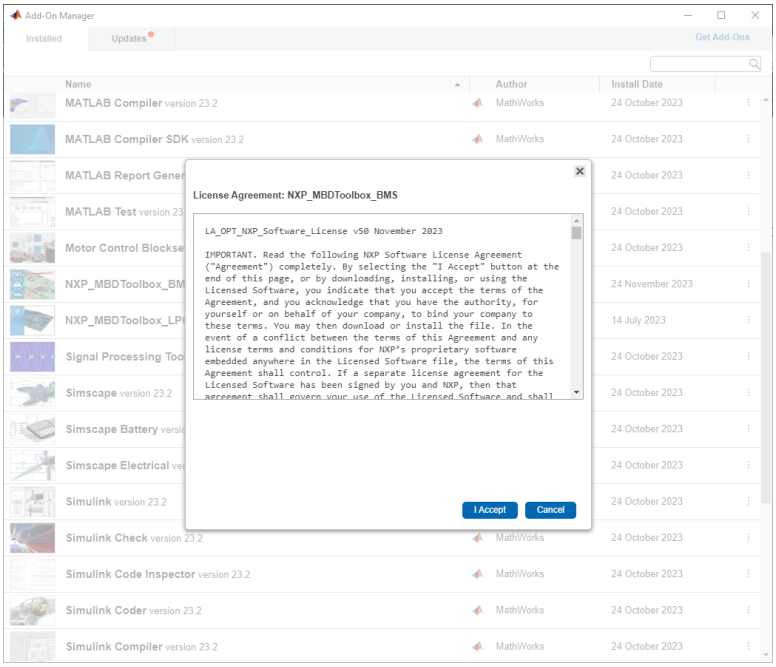


5. Go back to **NXP Support Package for BMS** and select the **Install MLTBX File as Add-On** button. In the newly opened window, browse for the MLTBX file and press **Open.**

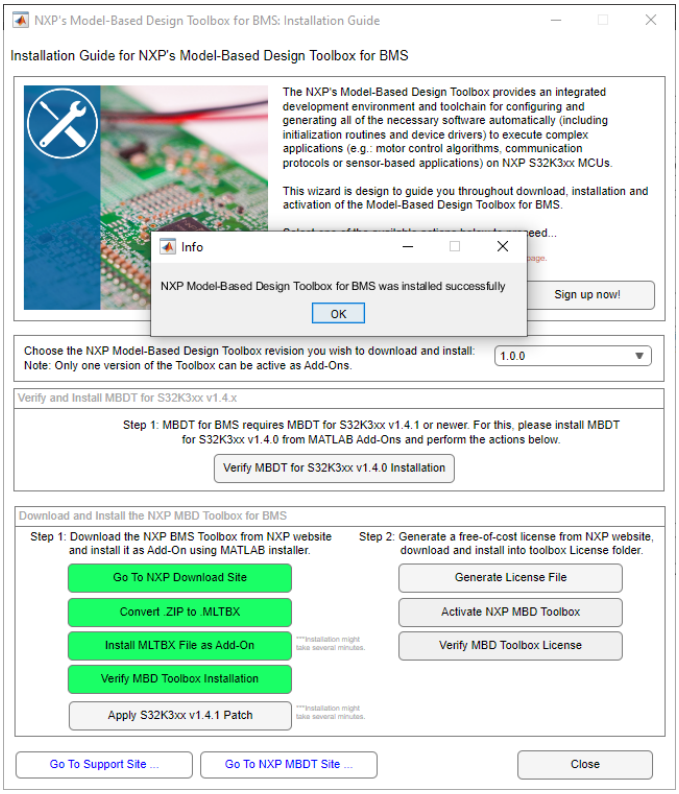


6. In the MATLAB Add-On Manager, Review the Terms and Conditions as you scroll down, and press **I Accept** Button. This action starts MBDT for BMS Toolbox installation process.

   Note: Installation might take several minutes.



7. Once the installation is complete, go back to **NXP Support Package for BMS** and press the **Verify MBD Toolbox Installation button**.



8. Update MBDT for S32K3xx 1.4.0/1.4.1/1.4.2 to version 1.4.3, by pressing the Apply S32K3 v1.4.3 Patch.

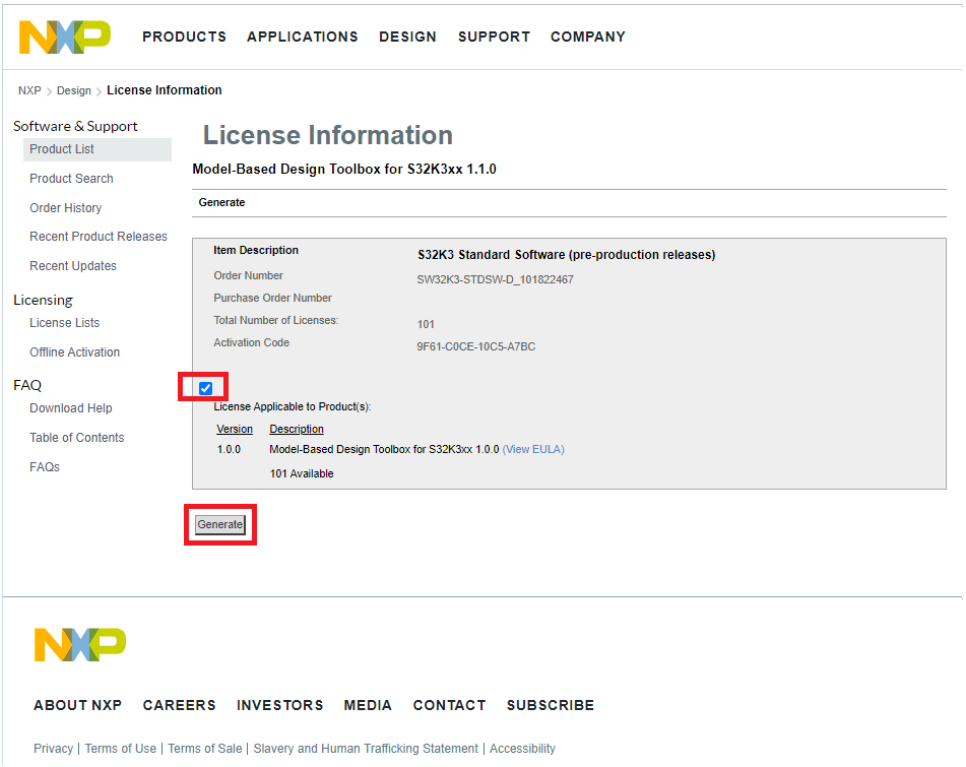
1.3.3 Generate and Activate NXP MBDT for BMS license

Even though the MBDT for BMS is free of charge, users still need to generate and install a free license. The following steps guide you on how to achieve such a license.

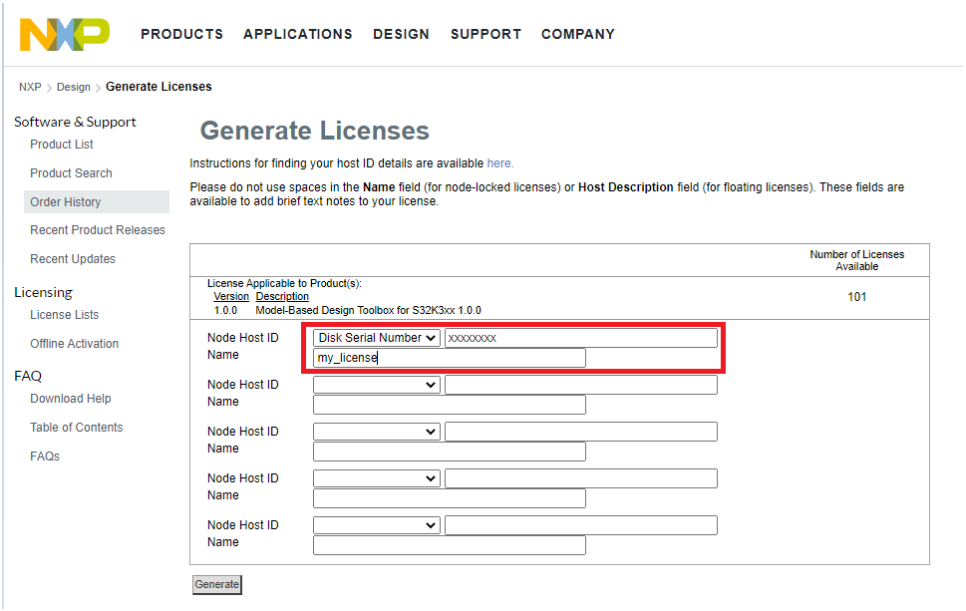
1. Press the Generate License File in the **NXP Support Package for BMS.**

2. In the newly opened webpage, select the checkbox as shown below, and press the generate button.

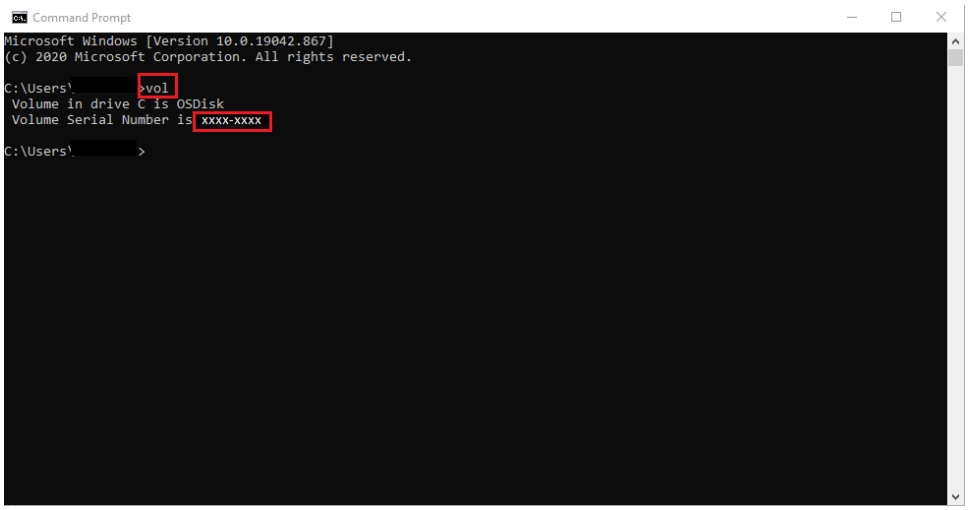
   Note: The following page shows the process required for the MBDT for S32K3xx toolbox. The License for MBDT for BMS can be achieved similarly. If a similar webpage as shown below is not being displayed, please go to the same page as described in the **Note** mentioned in step **1**, in section **1.2.2**. After selecting the latest available release, navigate to the **License Keys** tab.



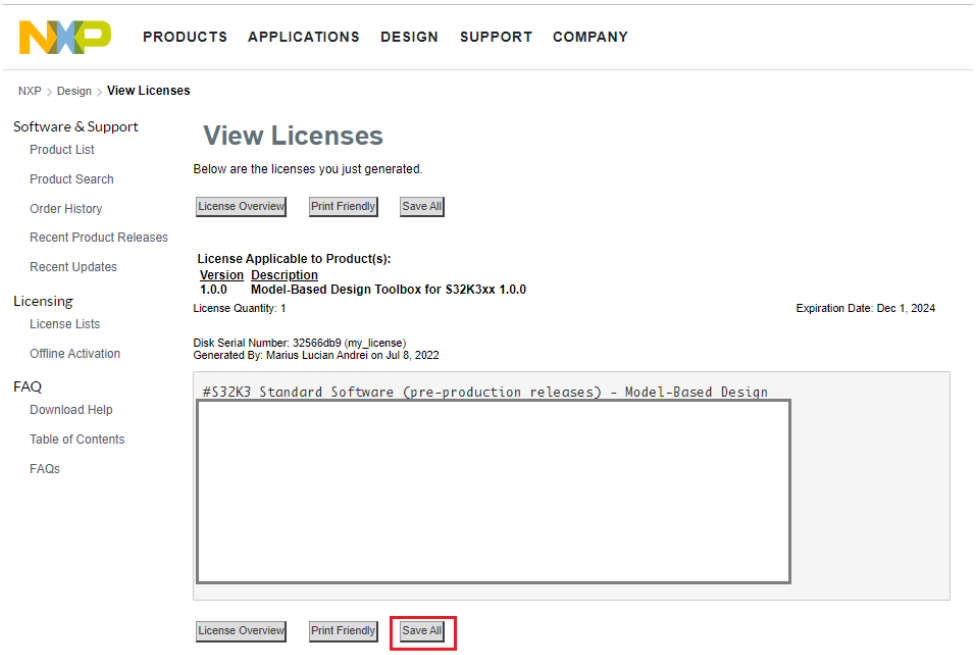
3. Select Disk Serial Number, and type the host id number. Give a name (**no spaces**) to the license, and press the **Generate** button.



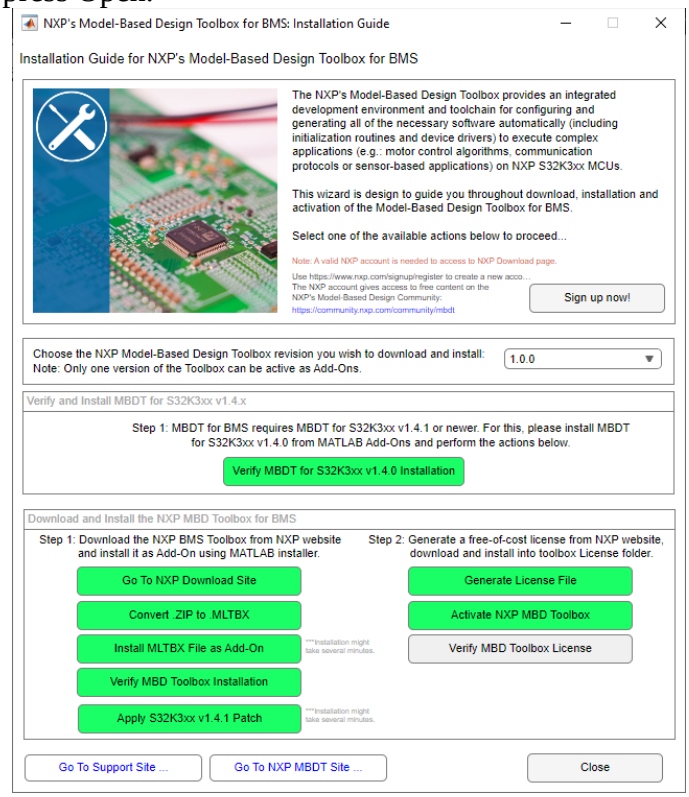
4. To find the host ID for your hard drive, please open a Windows Command Prompt and execute the “vol” command.



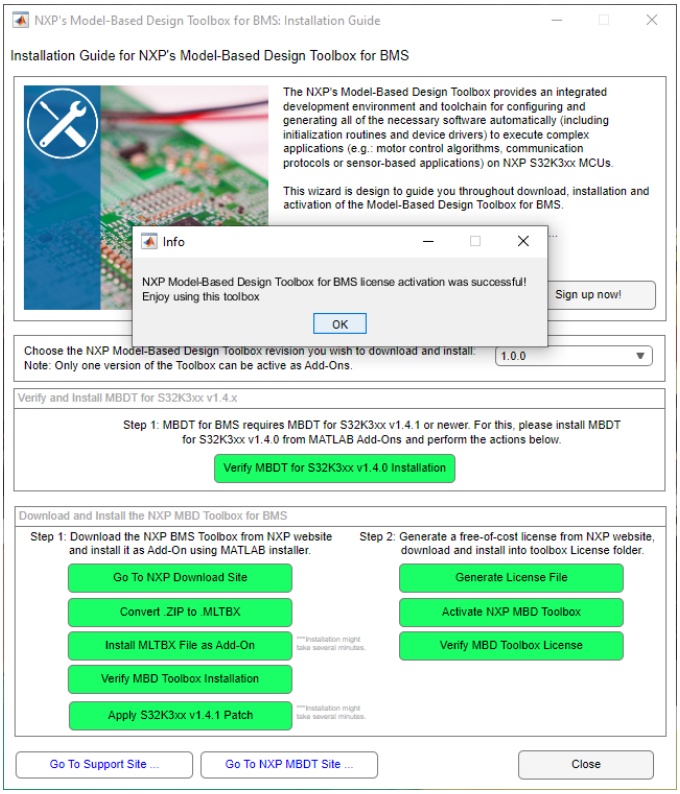
5. Now that the license has been successfully generated, press the **Save all** button.



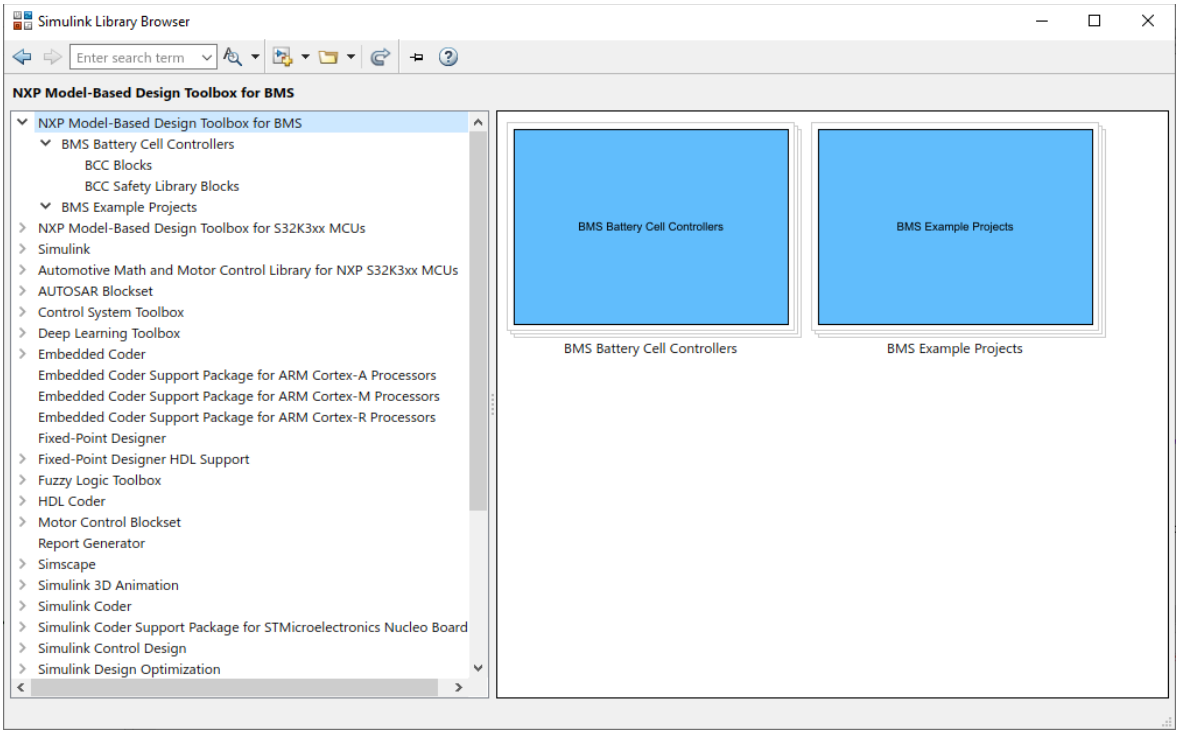
6. Back to **NXP Support Package for BMS,** press the **Activate NXP MBD Toolbox** button. In the newly opened window, Browse for the downloaded *license.dat* or *license.lic* file, and press Open.



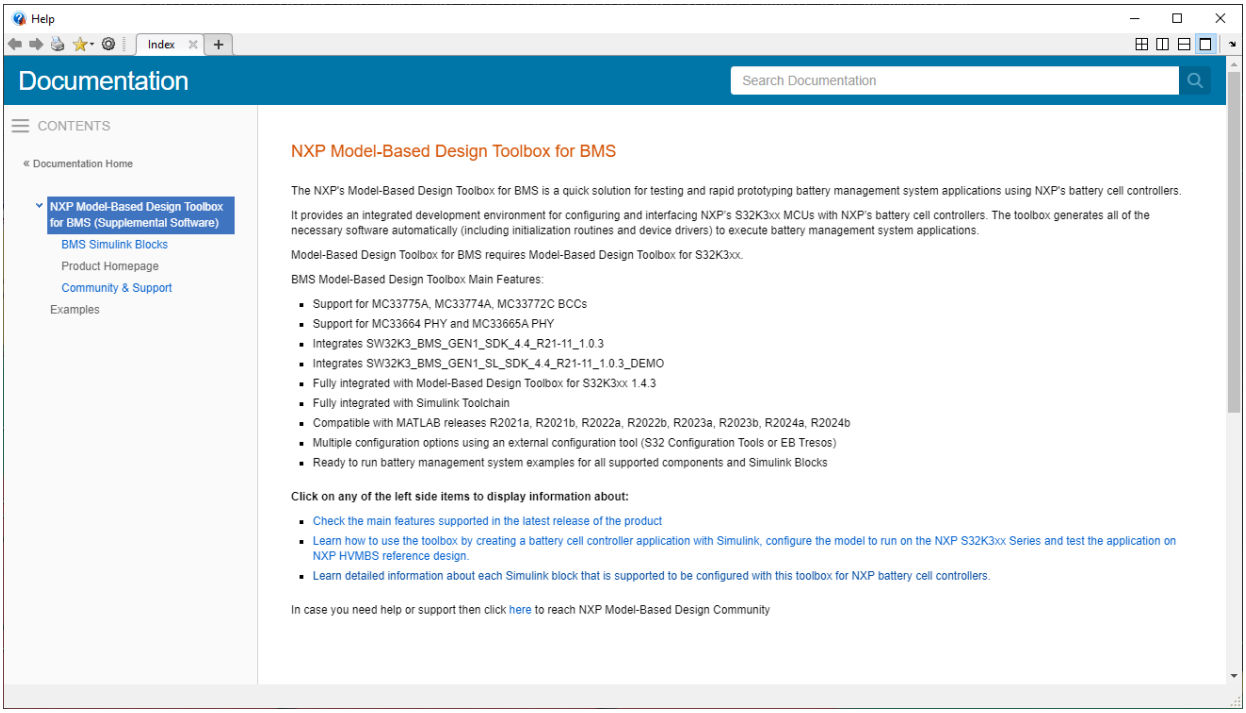
7. Final step is to check the license activation status, by pressing the **Verify MBD Toolbox License** button. If everything goes well, a similar popup window as below will be displayed.



NXP's Model-Based Design Toolbox layout and Simulink Library are shown below:



NXP's Model-Based Design Toolbox documentation, help, and examples are fully integrated with the MATLAB development environment. Get more details by accessing the standard Help and **Supplemental Software** section:



1.3.4 Setting the Path for Model-Based Design Toolbox and Toolchain Generation

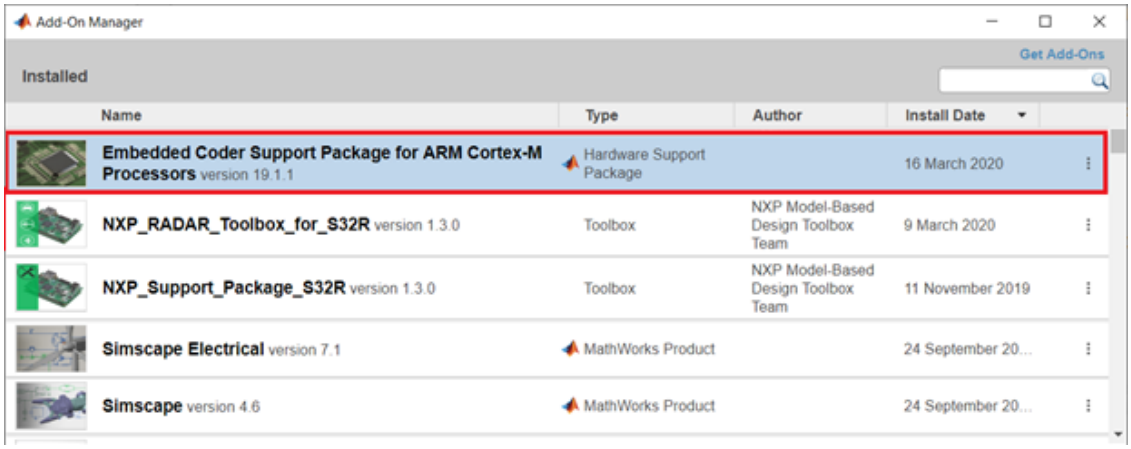
The Model-Based Design Toolbox uses the Toolchain mechanism exposed by the Simulink to enable automatic code generation with Embedded Coder toolbox. By default, the toolchain is configured for the supported MATLAB versions **R2021a - R2024b**. For newer MATLAB releases, the user needs to execute an MBDT for S32K3xx 1.4.x toolbox m-script to generate the appropriate settings for her/his installation environment.

This is done by changing the MATLAB Current Directory to the toolbox installation directory (e.g.: `..\MATLAB\Add-Ons\Toolboxes\NXP_MBDToolbox_BMS\`) and running the "`mbd_s32k3_path.m`" and "`mbd_s32k3.target.create_codertarget`" scripts.

```
>> mbd_s32k3_path
Treating 'C[…]\S32K3\src' as MBD Toolbox installation root.
MBD Toolbox path prepended.
Registering the toolchain ...
C:\Windows\System32\where.exe
Successful.
Creating folders for the target 'NXP S32K3xx' in the folder
'C:\[…]\S32K3\src\mbdtbx_s32k3\codertarget\2021a'...
Creating the framework for the target 'NXP S32K3xx'...
Registering the target 'NXP S32K3xx'...
Done.
```

```
>> mbd_bms_path
```

This mechanism requires users to install the Embedded Coder Support Package for ARM Cortex-M Processor as a prerequisite.



The "`mbd_bms_path.m`" script verifies the user setup dependencies and will issue instructions for a successful installation and configuration of the toolbox.

2 Run Models

2.1 Examples Library & Help

NXP's Model-Based Design Toolbox comes with an Examples Library collection that lets you test different MCU on-chip modules and run complex applications.

The Examples Library `mbd_bms_examples.slx` can be opened from "`{Model Based Design Install Directory}\BMS_Examples` folder or directly from the `Simulink Library Browser` main window.

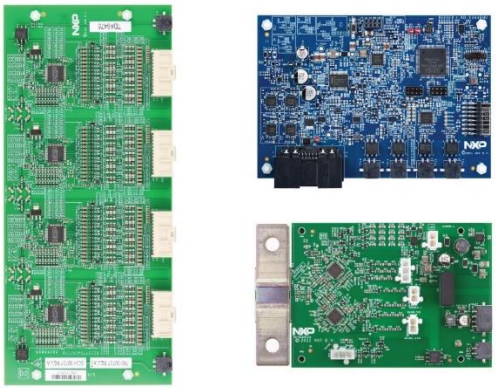
Each category contains multiple examples that showcase different Model-Based Design Toolbox capabilities that are categorized into different groups.

The examples are also available from standard MATLAB Help for NXP's Model-Based Design Toolbox Example.

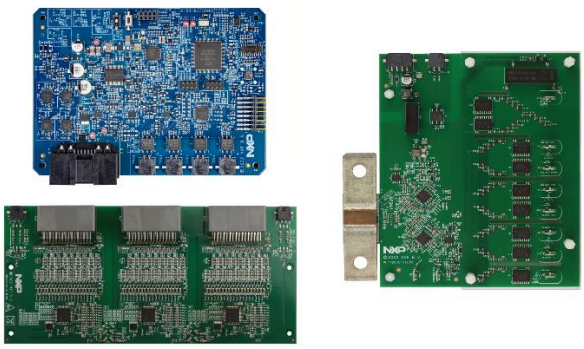
2.2 Hardware Setup

All examples provided with the Model-Based Design Toolbox were developed with:

- HVBMS Reference Design Bundle Using ETPL RD-HVBMSCTBUN:



- 800 V Battery Management System (BMS) Reference Designs Using ETPL RD-HVBMSCT800BUN:



- 14 V Battery Management System (BMS) Reference Design, Lead-Acid Replacement - RD33772C14VEVM

2.3 Running the HVBMS Examples

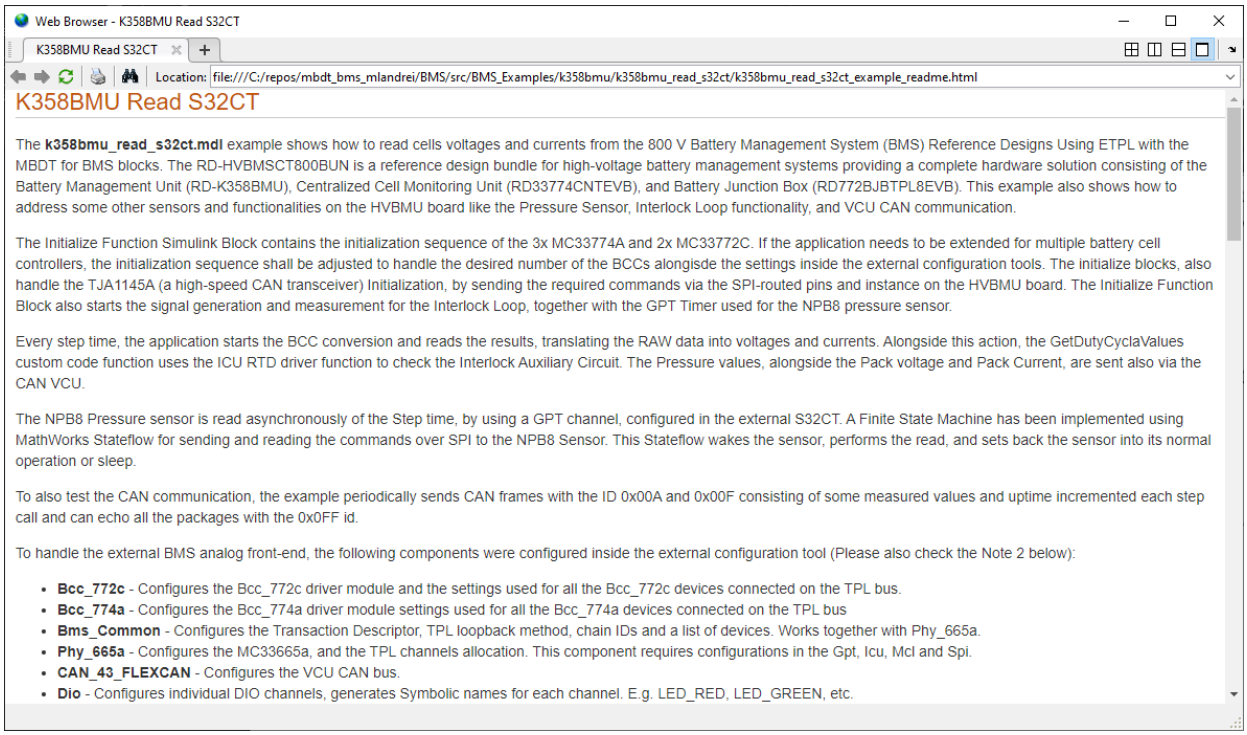
If the software setup is completed successfully, then all ingredients are present for running the Model-Based Design Toolbox-specific examples. The examples delivered by the MBDT for BMS toolbox are targeting the RD-HVBMSCTBUN, RD-HVBMSCT800BUN BMS bundles or the RD33772C14VEVM. Moreover, for the supported derivatives, the Model-Based Design Toolbox provides examples using both S32 Configuration Tools (**modelName_s32ct.mdl/slx**) and EB Tresos (**modelName_ebt.mdl/slx**) to demonstrate the interaction with the configuration tools it provides integration with.

Navigate to "`\BMS_Examples\`" folder and open the model according to the hardware used and the desired **configuration tool** (e.g `k358bmu_read_s32ct.mdl`)

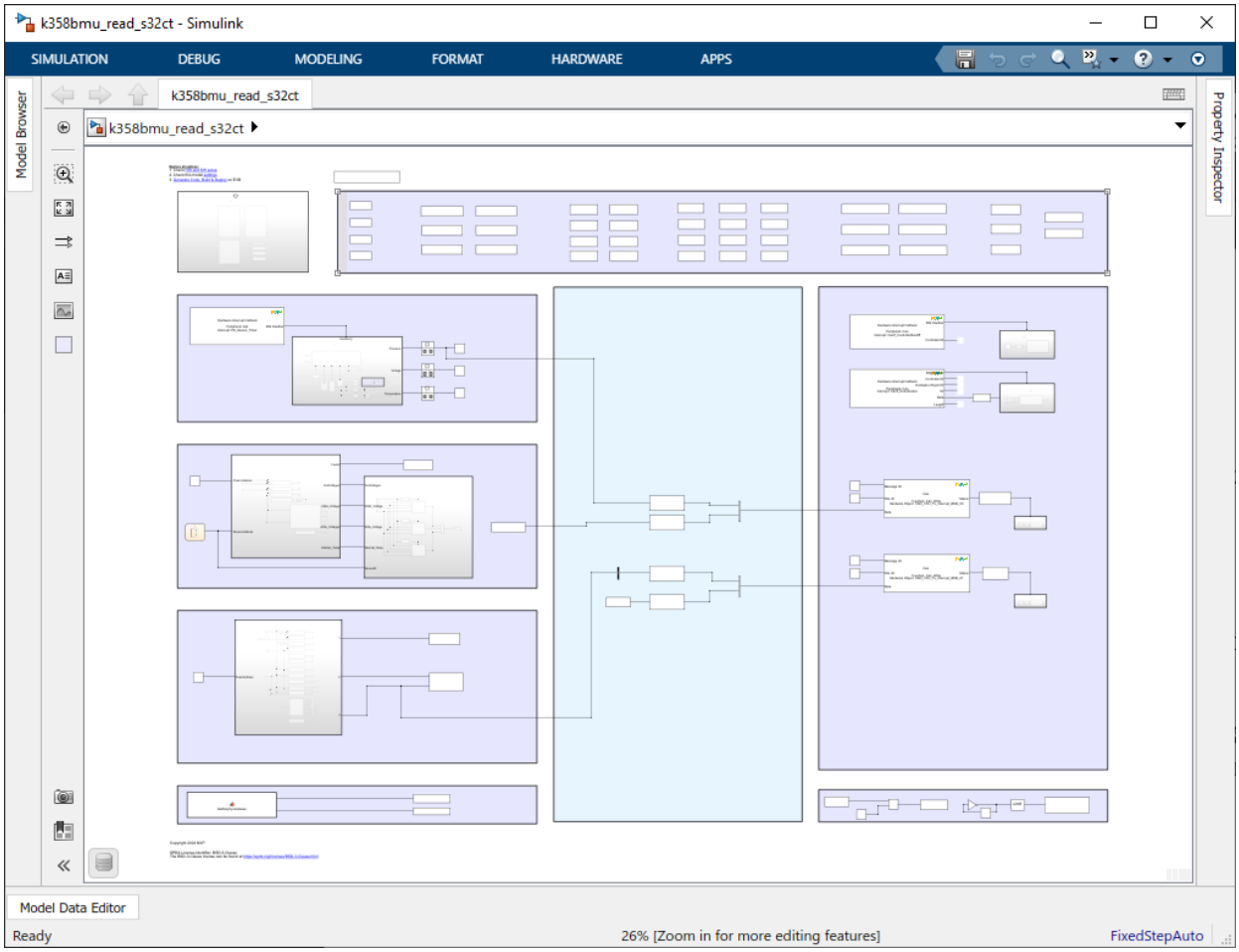
The `k358bmu_read_s32ct.mdl` example shows how to read cell voltages and currents from the 800 V Battery Management System (BMS) Reference Designs Using ETPL with the MBDT for BMS blocks. The RD-HVBMSCT800BUN is a reference design bundle for high-voltage battery management systems providing a complete hardware solution consisting of the Battery Management Unit (RD-K358BMU), Centralized Cell Monitoring Unit (RD33774CNTEVB), and Battery Junction Box (RD772BJBTPL8EVB). This example also shows how to address some other sensors and functionalities on the HVBMU board like the Pressure Sensor, Interlock Loop functionality, and VCU CAN communication.

Follow the next steps to run the example:

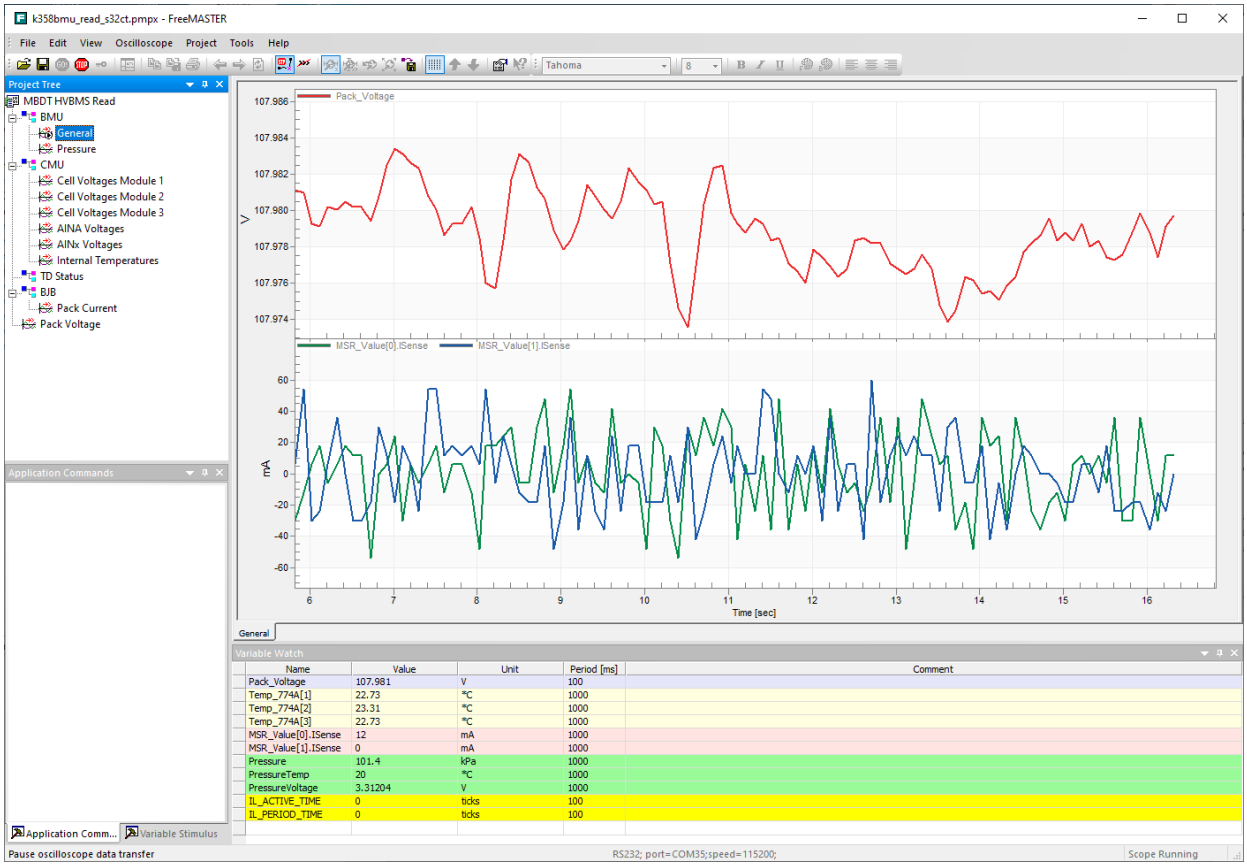
1. Open and README.html file to understand the hardware and software requirements for running the application.



2. Press the `Build Model` button and wait until the code is generated, compiled, and downloaded to the evaluation board. Alternatively, you can press on the text highlighted in the model to start the process automatically.



Open the FreeMASTER project associated with the model, and connect to the BMU. If successful, the BMU.General tab shows the pack voltage and the pack current measured by the battery management system.



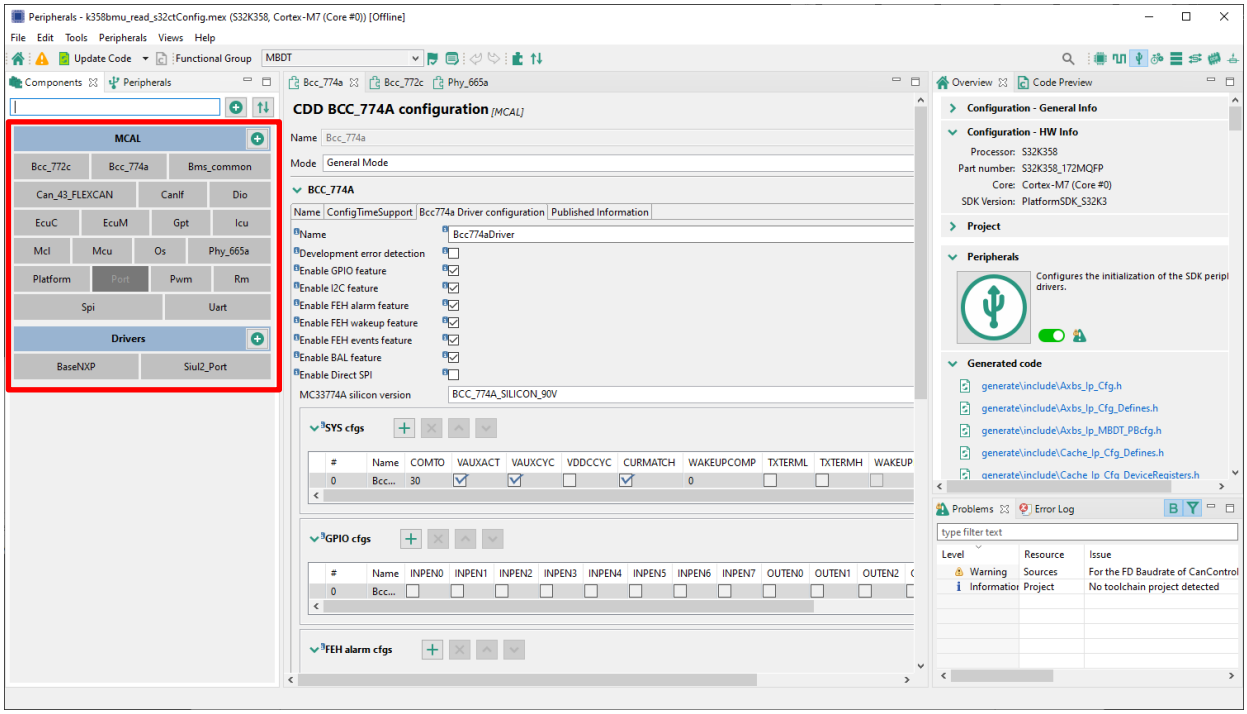
3 BMS Examples Development Guideline

This section describes the main steps and components that need to be configured for developing Simulink models to handle the NXP Analog Front End.

3.1 Configure components in the S32 Configuration Tool or EB tresos Studio

To handle the external BMS analog front end, the following components were configured inside the external configuration tool:

- **Bcc_772c** - Configures the Bcc_772c driver module and the settings used for all the Bcc_772c devices connected on the TPL bus.
- **Bcc_774a** - Configures the Bcc_774a driver module settings used for all the Bcc_774a devices connected on the TPL bus
- **Bms_Common** - Configures the Transaction Descriptor, TPL loopback method, chain IDs, and a list of devices. Works together with Phy_665a.
- **Phy_665a** - Configures the MC33665a and the TPL channels allocation. This component requires configurations in the Gpt, Icu, Mcl, and Spi.
- **CAN_43_FLEXCAN** - Configures the VCU CAN bus.
- **Dio** - Configures individual DIO channels, and generates Symbolic names for each channel. E.g. LED_RED, LED_GREEN, etc.
- **Gpt** - Configures the timers channels used by the Phy_665a. Note: this component also configures the StepTimer and ProfilerTimer used by the Simulink model.
- **Icu** - Configures the Icu Channel Configuration Mapping for the Phy_665a sideband signal, required by the Phy_665a and the channel required for the Interlock loop.
- **Mcl** - Configures the DMA Logic Channels used by the Phy_665a.
- **Mcu** - Configures the MCU module; Enables and configures the internal MCU clock sources.
- **Platform** - Configures Hardware Interrupt Handlers, implemented by the Real-Time Drivers for the peripherals used across the entire application. E.g. DMA Channels, STM & GPT timers, LPSPI, SIUL2 & LPUART. This component also sets the priorities for the Hardware Interrupts.
- **Pwm** - Configures the PWM channel required for the Interlock loop.
- **Port** - Configures the external MCU SIUL2 pins muxing and functionality.
- **Rm** - Configures the Direct Memory Access Multiplexer (DMAMUX) routes for the DMA sources configuration.
- **Spi** - Configures the Spi instances and sequences used to communicate with the Phy_665a, the SPI communication for the CAN Transceiver, and the SPI communication for the Pressure Sensor.
- **Uart** - Configures the LPUART instance used for debugging and FreeMASTER communication.



This model uses a custom S32 Configuration Tool project for the peripherals configuration, located next to the Simulink model, in the K358BMU_read_s32ctConfig.mex file. This custom configuration project can be launched in S32 Configuration Tool from any of the MBDT Blocks, which are placed inside the Simulink model, by opening the Block`s mask and pressing the "Configure" button. After the required changes are applied and saved in the **k358bmu_read_s32ctConfig.mex**, the S32 Configuration Tool project needs to be closed and the "Refresh" button needs to be pressed in the Block`s mask. By pressing the Refresh button, the new configuration settings are updated inside the Simulink model.

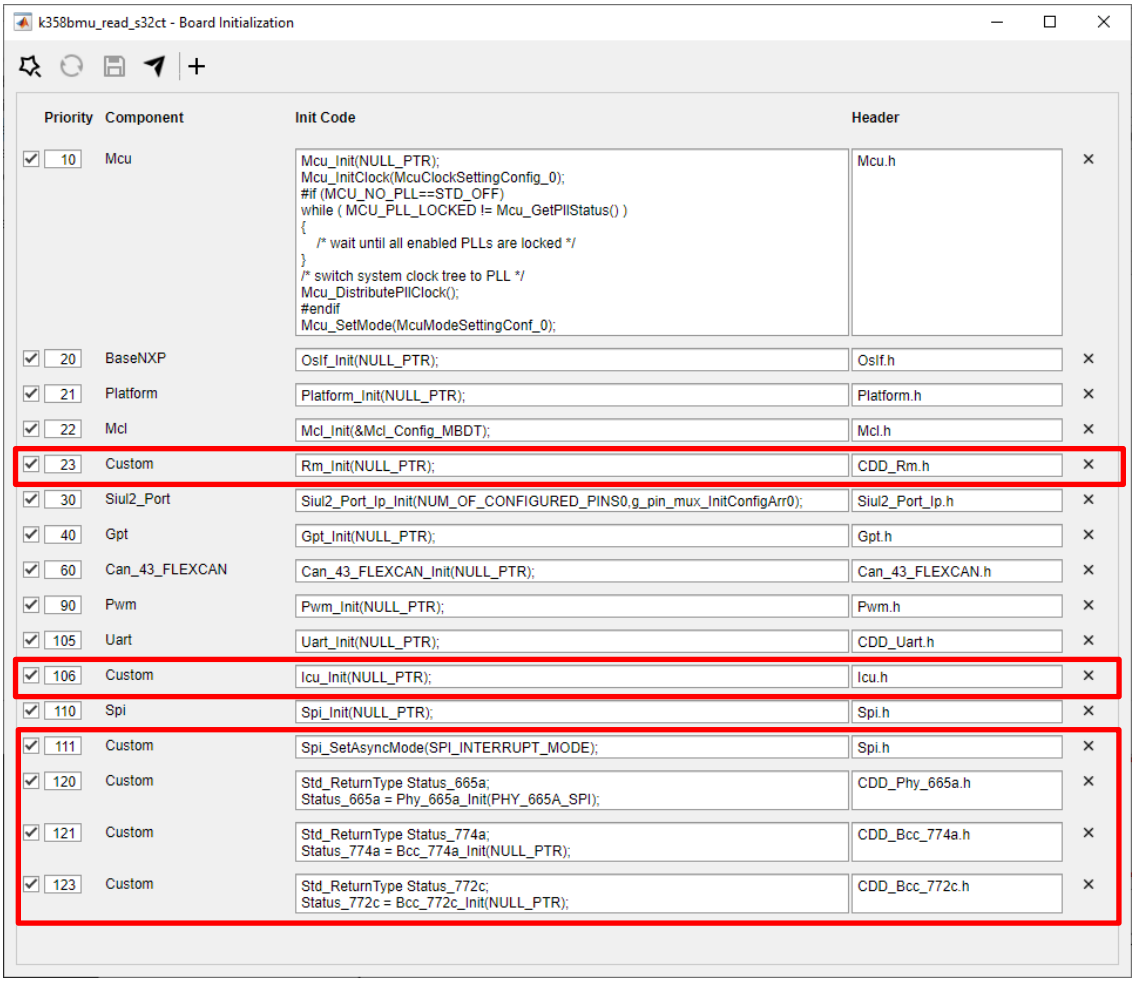
If the k358bmu_read_s32ctConfig.mex file or <model_name>TresosProject folder is deleted, the configuration is lost!

If the Simulink model changes are tracked via a Software Version Control system (Git, SVN, etc), then the **k358bmu_read_s32ctConfig.mex** file or **<model_name>TresosProject folder** is required to be pushed as well, and located next to the k358bmu_read_s32ct Simulink file.

3.2 Configure Board Initialization

Board Initialization needs to be extended so that the additional components configured in the external configuration tools (e.g. Bcc_772c, Bcc_775a, Phy_664, or Phy_665a) are also initialized. Model-Based Design Toolbox inserts automatically the configuration code for the components natively supported by the MBDT for S32K3, but the list needs to be extended with the ones which handle the Battery Cell Controllers.

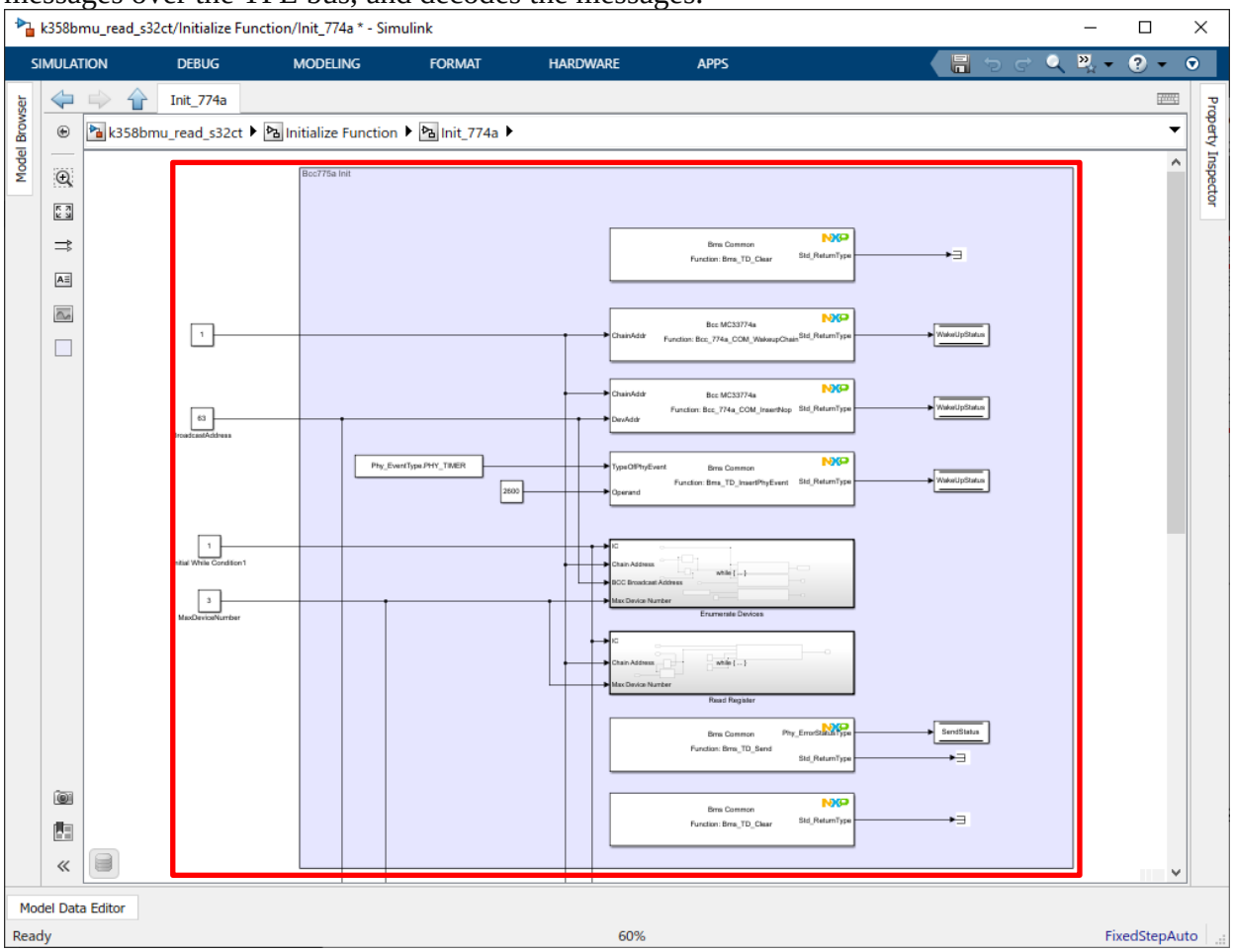
To access the **Board Initialization GUI**, please go to **Model Settings -> Hardware Implementation -> Target Hardware Resources -> Hardware -> Configure Block Initialization**.



For this example, all the Components marked as **Custom** were added. Note that the Priority column specifies the order in which the code will be generated. An External Board Initialization Template can also be provided. For more details on how to provide an initialization template, please check the MBDT for S32K3xx Release Notes and Quick Start Guides.

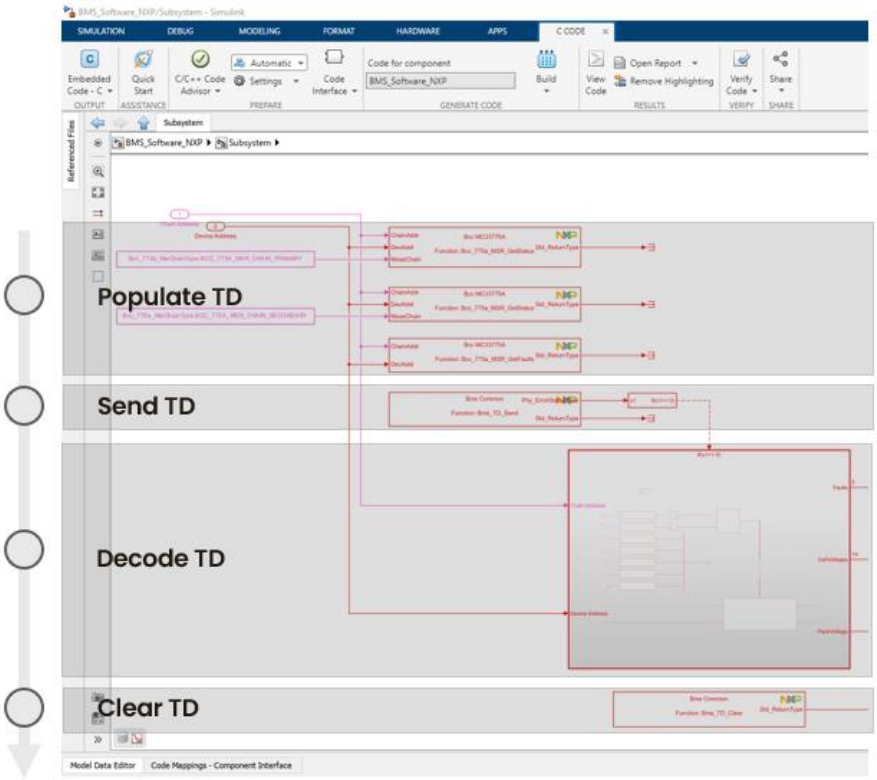
3.3 Initialize Subsystem

The **Initialize subsystem** provides the sequence blocks for the TPL bus Wake up and device enumeration, while the main subsystem starts the battery cell controller conversion, receives the messages over the TPL bus, and decodes the messages.



3.4 Organize the TPL data exchange using Transactions Descriptors

The messages sent over the TPL bus are organized in Transaction Descriptors.



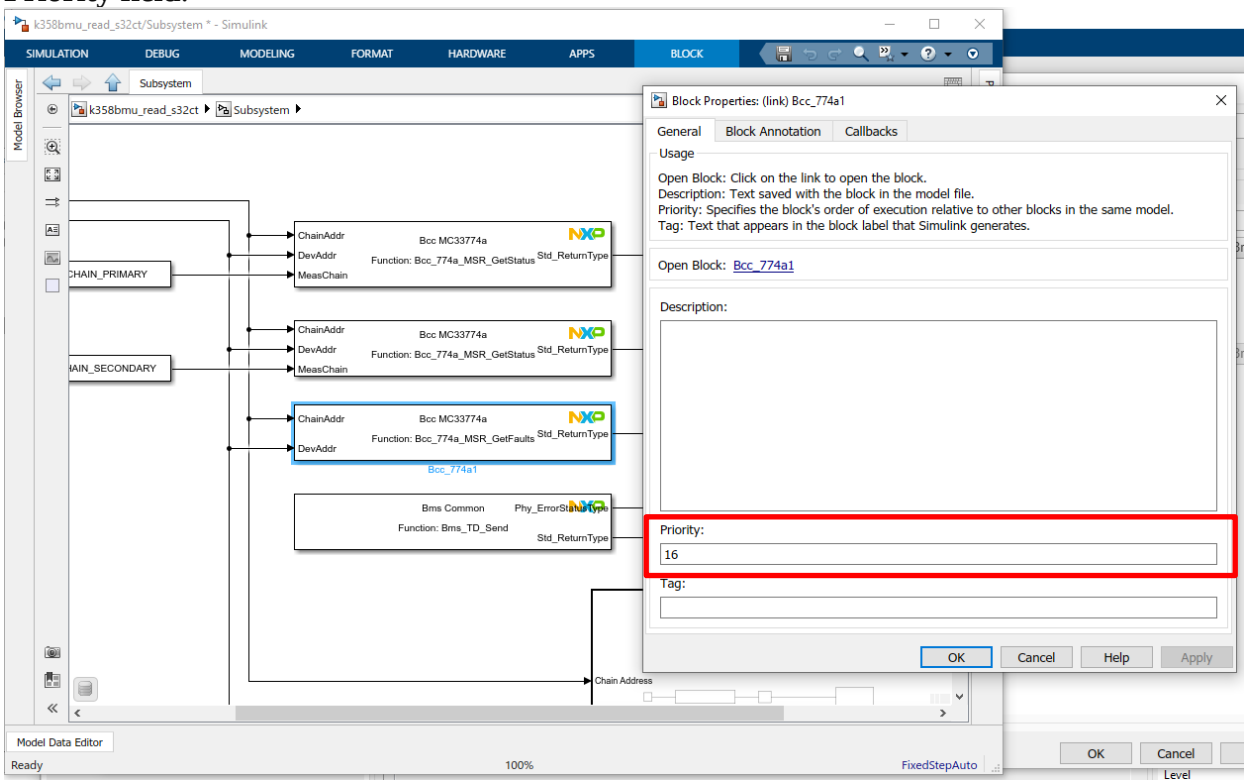
The main steps for data exchange between the MCU and BCCs via TPL are:

- **Populate Transaction Descriptor** - Use the blocks Bcc_772c, Bcc_775a, and Bcc_774a to populate the TD Handlers. Please select the proper TD Handler configured for the TPL version ID and device type.
- **Send Transaction Descriptor** - Use the Bms_TD_Send from the Bms_Common Block.
- **Decode Transaction Descriptor** - Use the Bcc_TD Block to access the RAW bytes from the TD handler. Additionally, the Bms_772C_SL and Bms_TPL3_SL_E2E Blocks can be used to search for the required data inside the TD Handler. When the commands are sent over the TPL bus (E.g. when starting a conversion no response is waited from the Battery Cell Controller) then this step can be omitted.
- **Clear Transaction Descriptor** - Use the Bms_TD_Clear from the Bms_Common Block.

Note! It is mandatory to set the following C compiler flag when the **Bms_772C_SL** or **Bms_TPL3_SL_E2E Blocks** are used inside the model (BMS SDK SL): "**-fno-short-enums"**. The C Compiler flags can be modified in the **Model Settings -> Code Generation -> Build process -> Toolchain Settings**.

The functions inside the Bcc_772c, Bcc_775a, and Bcc_774a Blocks are populating the selected TD Handler (TD Handler is selected in each Block`s mask and declared/configured in the external configuration tools). The TD Handlers are physically sent on the TPL bus only when the Bms_TD_Send function of the Bms_Common Block is executed.

Note that the order in which the Blocks generate the C code (the same as the order in which the instructions are populated inside the TD Handlers) is very important. The Bcc Blocks can be ordered by specifying a priority for each block by **right click on the block -> Properties ... -> Priority field**.



How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

NXP Semiconductor reserves the right to make changes without further notice to any products herein. NXP Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. NXP Semiconductor does not convey any license under its patent rights nor the rights of others. NXP Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the NXP Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use NXP Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold NXP Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that NXP Semiconductor was negligent regarding the design or manufacture of the part.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.
Microsoft and .NET Framework are trademarks of Microsoft Corporation.
Flexera Software, Flexlm, and FlexNet Publisher are registered trademarks or trademarks of Flexera Software, Inc. and/or InstallShield Co. Inc. in the United States of America and/or other countries.
NXP, the NXP logo, CodeWarrior and ColdFire are trademarks of NXP Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Flexis and Processor Expert are trademarks of NXP Semiconductor, Inc. All other product or service names are the property of their respective owners