

# AN1115: Configuring Peripherals for 32-Bit Devices using Hardware Configurator

---

In Zigbee EmberZNet SDK version 6.10.x and lower, peripherals are configured using the Hardware Configurator in Simplicity Studio. The Hardware Configurator simplifies peripheral configuration by presenting peripherals and peripheral properties in a graphical user interface. For some SDKs, many peripherals can also be configured from within the Simplicity IDE as plugin options.

Beginning with Gecko SDK Suite v3.x and Simplicity Studio 5, a new Pin Tool provides similar functionality for SDKs that have migrated to the new component-based framework. See the [Simplicity Studio® 5 User's Guide](#) for more information. If you are developing with Zigbee EmberZNet version 7.0 and higher, refer to *AN1321: Configuring Peripherals for 32 Bit Devices with Zigbee 7.0 and Higher*.

## KEY POINTS

- Describes the two essential Hardware Configurator files.
- Discusses modifying peripheral configurations by changing plugins in the Simplicity IDE.
- Provides details on using the Hardware Configurator tool.

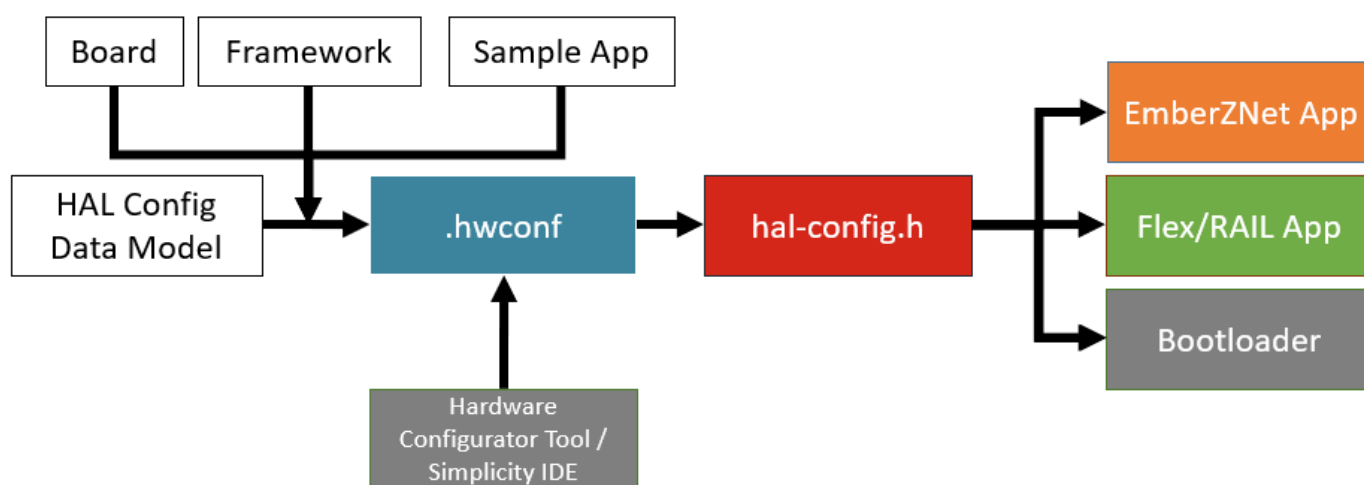
## 1. Introduction

Version 2.0.0.0 of the Gecko SDK Suite implemented a number of changes to the way developers can configure peripherals on their 32-bit devices. The Hardware Configurator tool has been enhanced to produce configuration output that is standardized across the EmberZNet and Flex SDKs. Developers in these SDKs, who had previously configured peripherals by editing header files, now can use the Hardware Configurator either through an interface in Simplicity IDE or through the Hardware Configurator tool.

The Hardware Configurator provides an easy-to-use method of peripheral configuration by presenting peripherals and peripheral properties in a graphical user interface.

### 1.1 File Inputs and Outputs

The inputs to the Hardware Configurator processor are shown in the following figure. Files that are specific to the selected part, the stack application framework, and the selected example application combine through the HAL Config data model to produce a project-specific .hwconf file (<boardname>\_<partname>.hwconf). Developers using any SDK can edit the .hwconf file using the Hardware Configurator tool by double-clicking the .hwconf file. Developers using the EmberZNet and Flex SDKs can also modify some options by editing plugins in the Simplicity IDE. The .hwconf file is an integral part of any project. If you need to provide your project files to Silicon Labs Support, provide the .hwconf file along with the .isc file and other project files.



When the .hwconf file is saved or when project files are generated, the Hardware Configurator processor uses the .hwconf file to produce a configuration header, hal-config.h. It contains a structured set of defines that are used to configure the HAL or BSP. This header is standardized across SDKs.

Developers can add their own defines to this file. In earlier approaches to hardware configuration, the initDevice files or board header were completely overwritten on generation, and updates would be lost. In the current approach, customizations are maintained as long as they are written to an unused generation region in hal-config.h.

The hal-config.h file is always in a fixed location:

```
<workspace>/<projectName>/hal-config/hal-config.h
```

## 1.2 The HAL Config Framework

The HAL Config framework is a set of standardized configuration options used to initialize and customize hardware peripherals and drivers. These options are constructed with three components:

- **Prefix:** Either BSP or HAL
- **Module:** Each module pertains to a set of hardware features based on an underlying peripheral and/or software enhancements.
- **Option:** Each module can be customized by defining configuration options. These might include enabling/disabling the module, specifying a peripheral signal route location, or mode to initialize the module.

A complete list of options for the current SDK may be found under the Simplicity Studio installation:

```
\developer\sdk\gecko_sdk_suite\<version>\platform\halconfig\reference.html
```

The following excerpt shows the Button framework.

```
// $[BUTTON]
#define BSP_BUTTON_PRESENT                (1)

#define BSP_BUTTON0_PIN                  (6)
#define BSP_BUTTON0_PORT                 (gpioPortF)

#define BSP_BUTTON1_PIN                  (7)
#define BSP_BUTTON1_PORT                 (gpioPortF)

#define BSP_BUTTON_COUNT                 (2)
#define BSP_BUTTON_INIT                  { { BSP_BUTTON0_PORT, BSP_BUTTON0_PIN }, { BSP_BUTTON1_PORT,
BSP_BUTTON1_PIN } }
#define BSP_BUTTON_GPIO_DOUT             (HAL_GPIO_DOUT_LOW)
#define BSP_BUTTON_GPIO_MODE             (HAL_GPIO_MODE_INPUT)
#define HAL_BUTTON_ENABLE                 { 0, 1 }
#define HAL_BUTTON_COUNT                 (2)
// [BUTTON]$
```

## 2. Integration with Simplicity IDE and EmberZNet and Flex SDKs

This section describes how to configure peripherals inside the Simplicity IDE for the following SDK releases:

- EmberZNet SDK 6.0.0 or higher
- Flex 2.0.0 or higher.

Because of the significant changes to hardware configuration from previous SDK versions, custom configurations in projects created with those earlier versions, particularly those for custom hardware, do not automatically update. You must edit the hardware configuration to match previously implemented customizations. If you are working on a Silicon Labs platform and had made your changes in earlier version of configuration plugins, upgrade rules will make most of the changes when you import the .isc file to the current SDK. Section [2.2.2 Upgrading from Gecko SDK Versions Prior to 2.0.0](#) discusses some of the key differences between earlier implementations of hardware configuration and the current implementation.

Also note, when creating a project for a custom hardware board you will be required to use the hardware configurator to manually set up any information related to your custom board, for example whether a LFXO is present.

The .hwconf file default configuration is based on the part selected when the project is created. Unless the part is changed on the General tab, the .hwconf file is not regenerated, only edited.

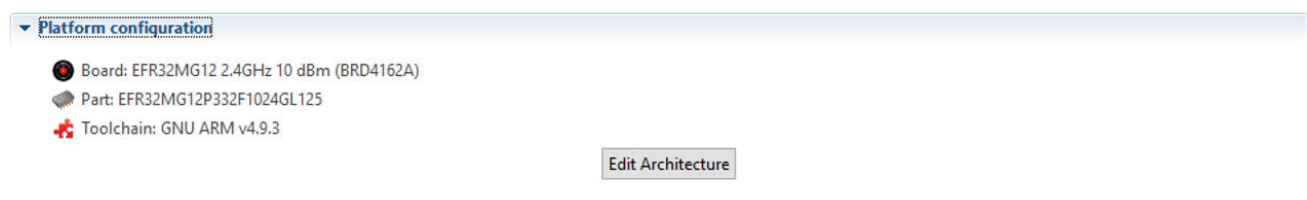
Not all Hardware Configurator options can be modified through the Simplicity IDE. The Hardware Configurator tool provides access to all options. In general, Silicon Labs recommends that you configure your project by enabling and disabling the relevant plugins, and then customize peripherals through the Hardware Configurator tool. Because both the Hardware Configurator tool and the Simplicity IDE interface are changing the underlying .hwconf file, changes made in one GUI can be seen in the other.

### 2.1 Functionality on the General and HAL Tabs

#### 2.1.1 General Tab

The Architecture selector allows you to modify the architecture you selected when you created the project, or that was automatically configured as part of the "one-click" method of creating an example project. Per the following figure the architecture shows:

- Board
- Part. Changing the part causes a new .hwconf file to be generated the next time you click [**Generate**].
- Toolchain



If you have made changes to the .hwconf file and then change the part selected in the Architecture, your .hwconf changes will not be transferred to the new .hwconf file that is generated.

## 2.1.2 HAL Tab

The HAL tab contains hooks to the Hardware Configurator. As shown in the following figure, the Board Header section enables Hardware Configurator, which is required for EFR32xG applications. A number of options are greyed out. Users of earlier SDK versions will note that they reflect the former architecture using the `hal-config.h` file. The interface reflects the fact that the Hardware Configurator works directly with the `.hwconf` file.

The screenshot shows the 'Board header' section of the HAL tab. It contains the following elements:

- A title bar: **Board header**
- Instruction text: Use these controls to select a board header you wish to use with the selected architecture.
- Four radio button options:
  - ☐ Default header from framework: BRD4162A: EFR32MG12 2.4GHz 10 dBm [\$(KIT)/EFR32MG12\_BRD4162A/config/brd4162a.h]
  - ☐ Specific header from framework: BRD4162A: EFR32MG12 2.4GHz 10 dBm [\$(KIT)/EFR32MG12\_BRD4162A/config/brd4162a.h] (dropdown arrow)
  - ☐ Custom created header: Absolute path (text field with file icon)
  - ☒ Use Hardware Configurator (all EFR32 applications must use Hardware Configurator for board configuration) (highlighted with a red box)
  - ☐ Use GPIO editor for template header (not available for selected architecture) (disabled)
- Generation options section:
  - ☐ If header is missing, don't treat this as an error, but proceed anyway.
  - ☐ Use header directly, do not copy it into the project directory.

Further down, another block configures integration with the Hardware Configurator. See the following figure for an example.

The screenshot shows the 'Manage integration between your project and Hardware Configurator inside Simplicity Studio' section. It contains the following elements:

- Title bar: **Manage integration between your project and Hardware Configurator inside Simplicity Studio.**
- ☒ Enable integration with Hardware Configurator
- Buttons: **Open Hardware Configurator** and **Reset Hardware Configurator**
- ☐ Custom Hardware Configurator File: Absolute path (text field with file icon)
- Section: Hardware Configurator files used in this application
- File list table:

	SDK contributed files
▶	Framework contributed files
▶	Sample app contributed files

If you want to import the hardware configuration from an existing project that is based on the same target hardware, check the **Custom Hardware Configurator File** checkbox and provide an absolute path to use an `.hwconf` file in a different project, or when you are importing a project. When the checkbox is checked, the code that generates an `.hwconf` file for the project is bypassed.

Click **[Open Hardware Configurator]** to open the tool. This has the same effect as double-clicking the `.hwconf` file. See section [3. Using the Hardware Configurator](#) for more information on the Hardware Configurator tool. The file block shows the input files to the Hardware Configurator processor. These are the same three categories shown in the figure in section [1.1 File Inputs and Outputs](#).

Finally click **[Reset Hardware Configurator]** to go back to the defaults based on the input files. This deletes any manual changes to the hardware configuration of the project.

## 2.2 Plugin Configuration

Plugins are configured as usual, by clicking on the plugin and making changes in the configuration options. Plugins that modify Hardware Configurator settings have a **Hardware Configurator Dependent Options** pane. You may need to scroll down in that pane to see all options.

Enabling/disabling plugins can also enable and disable Hardware Configurator modules and their dependent peripherals. When a plugin with Hardware Configurator dependencies is enabled/disabled, a pop-up dialog box shows which Hardware Configurator modules and peripherals are being enabled/disabled. See the following figure for an example.

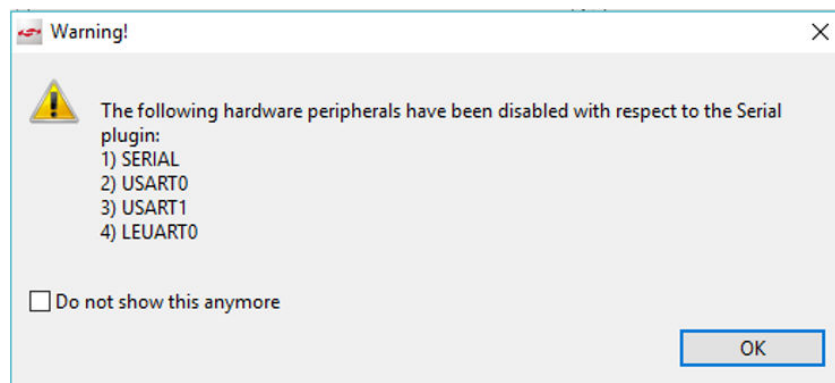


Figure 2.1. Hardware Configuration disabled

Also, when a plugin that shares Hardware Configurator modules/peripherals with another plugin is enabled/disabled, a similar warning updates the user which peripherals are being affected behind the scenes in the Hardware Configurator.

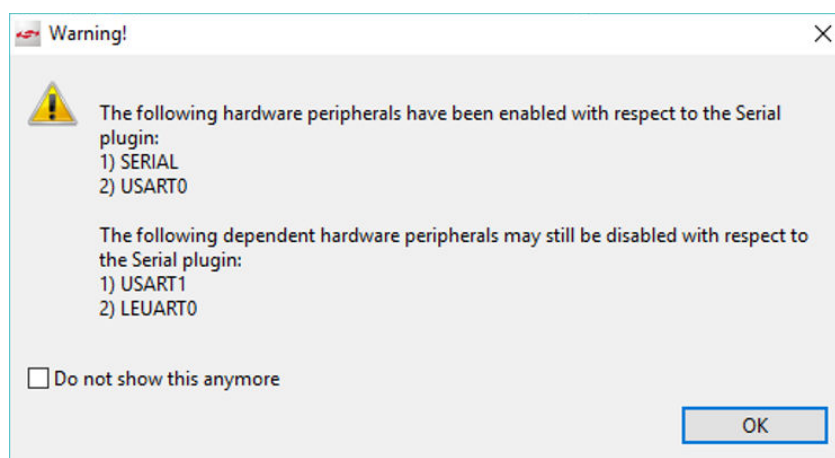
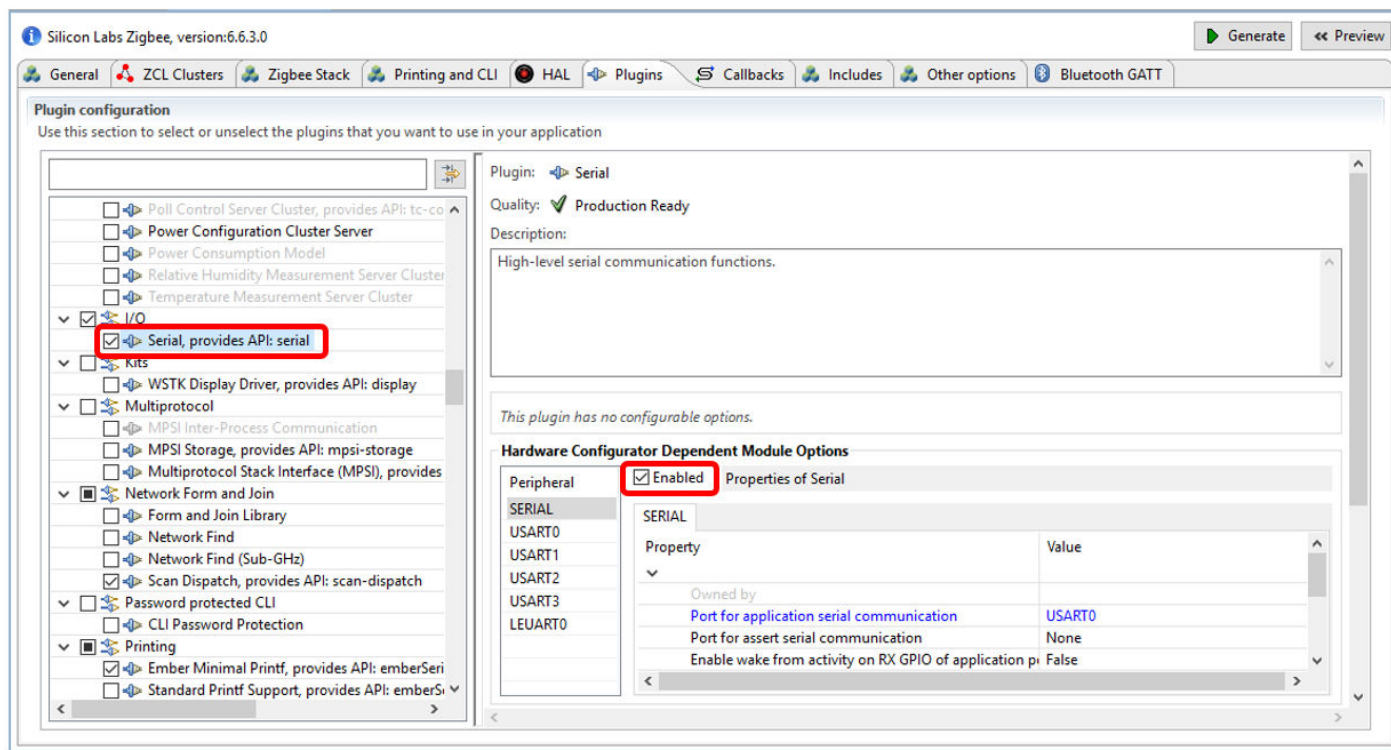


Figure 2.2. Hardware Configuration enabled

## 2.2.1 Using the Interface

The following figure shows the Serial plugin from the Silicon Labs Zigbee Z3LightSoc example. Click on a configurator option to open a drop down or otherwise edit the property. The .hwconf file begins with standard default settings. Options that are black are default from the Hardware Configurator, and options that are blue have been modified, either by an example application or by you. All of the example applications provided with the SDKs modify the defaults. Each Hardware Configurator Dependent Module peripheral has an **Enabled** checkbox. If you are enabling a plugin with a peripheral, you must not only check the plugin to enable it but must also enable one or more dependent peripherals.



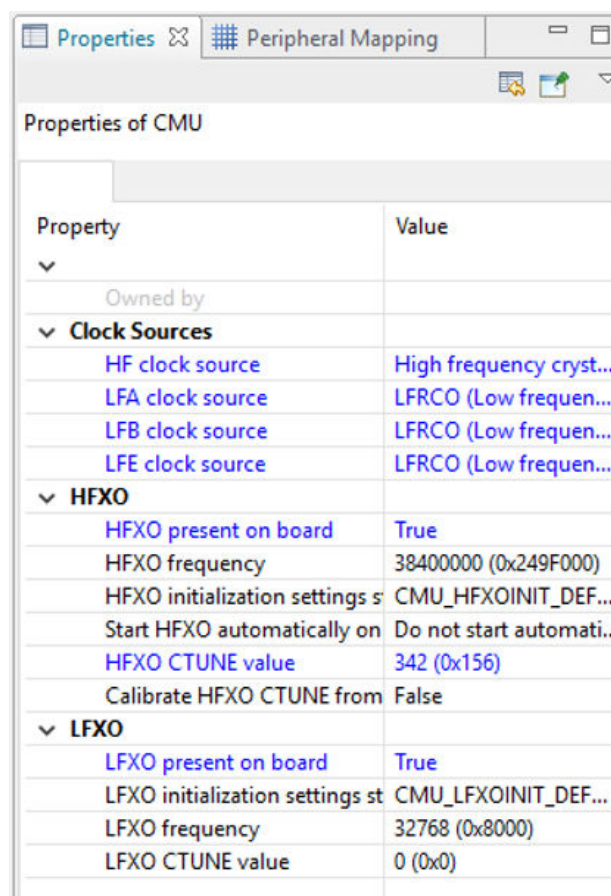
Peripherals can be enabled either through the **Enabled** checkbox of a plugin or through the checkbox on the peripheral box in the Hardware Configurator tool (see section 3. [Using the Hardware Configurator](#)). While this seems redundant when a plugin has only one peripheral, some plugins such as the Serial plugin have several. Simply select one of the peripherals from the list to see its configuration.

## 2.2.2 Upgrading from Gecko SDK Versions Prior to 2.0.0

Most importantly, modifications to Silicon Labs device peripherals are no longer made by editing the board header file but rather by changing the .hwconf file through the Hardware Configurator GUI. For example, to enable and configure a button or LED, which used to be done by editing hal-config.h, you now modify the options either on the plugins (new for EmberZNet) or on the module in Hardware Configurator.

Other enhancements include:

- The ability to modify the peripheral configuration in the Flex SDK RAILTest example.
- Easier and more transparent clock configuration through the Hardware Configurator CMU module as shown in the following figure.



A number of configuration options have been relocated. Developers who import projects into the new SDK will see upgrade rules to handle these changes. Changes include:

- Serial Configuration: Moved from the HAL (HAL Configuration in EmberZNet) tab to the Hardware Configurator options on the Serial plugin.
- The HAL Configuration plugin has been removed. All HAL Configuration options are located in the Hardware Configurator peripherals options, many as peripherals on the Serial plugin.

Hal Config Plugin Option	Hardware Configurator Peripheral/Plugin
Antenna diversity	Antenna peripheral/plugin
PTA and Radio Hold Off	Combined into COEX peripheral/plugin
Enable GPIO RX	Serial peripheral/plugin property
Enable VCOM	VCOM peripheral
Enable VUART	VUART peripheral
Enable Watchdog	WDOG peripheral
Enable LEUART0 / LEUART1	Peripherals associated with serial peripheral/plugin



Hal Config Plugin Option	Hardware Configurator Peripheral/Plugin
USART 0, 1, 2, 3	Peripherals associated with serial peripheral/plugin
UART NCP port:	UART NCP peripheral

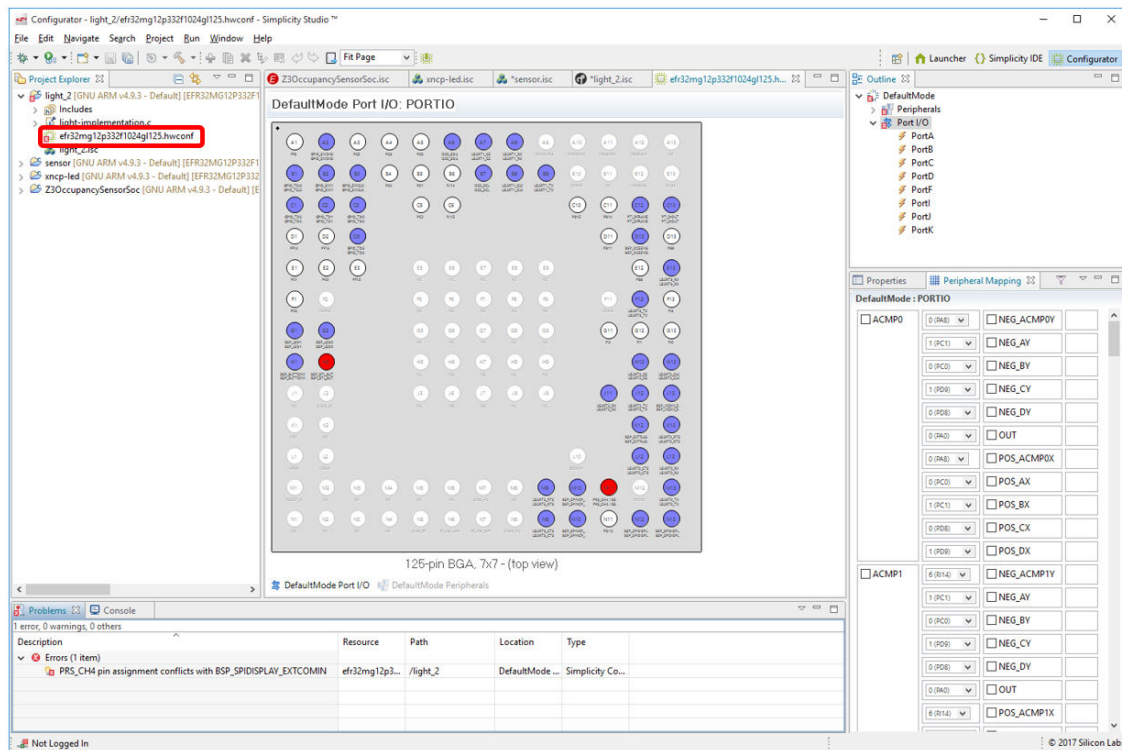
In some cases, configuration changes may still be made by selecting the plugin, but the options have been moved to the Hardware Configurator.

Plugins in this category include the following:

- Antenna diversity
- Coexistence Configuration
- GPIO activation
- SPI Flash
- NCP UART

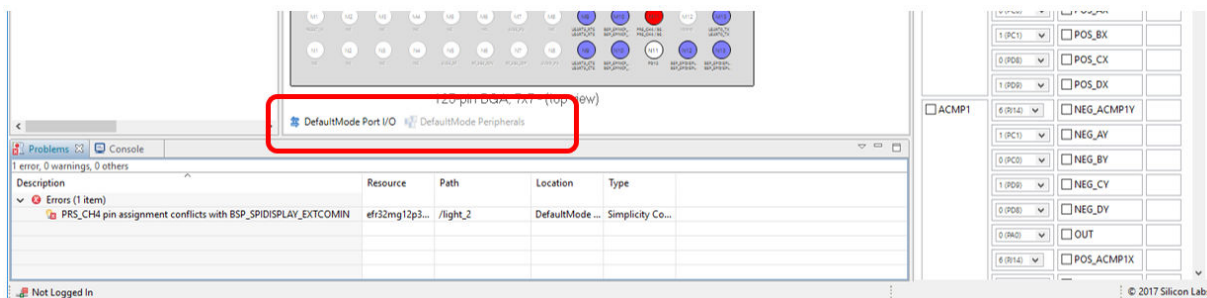
### 3. Using the Hardware Configurator

To open the Hardware Configurator for an open project, double-click the .hwconf file in Project Explorer or click **[Open Hardware Configurator]** on the HAL tab.



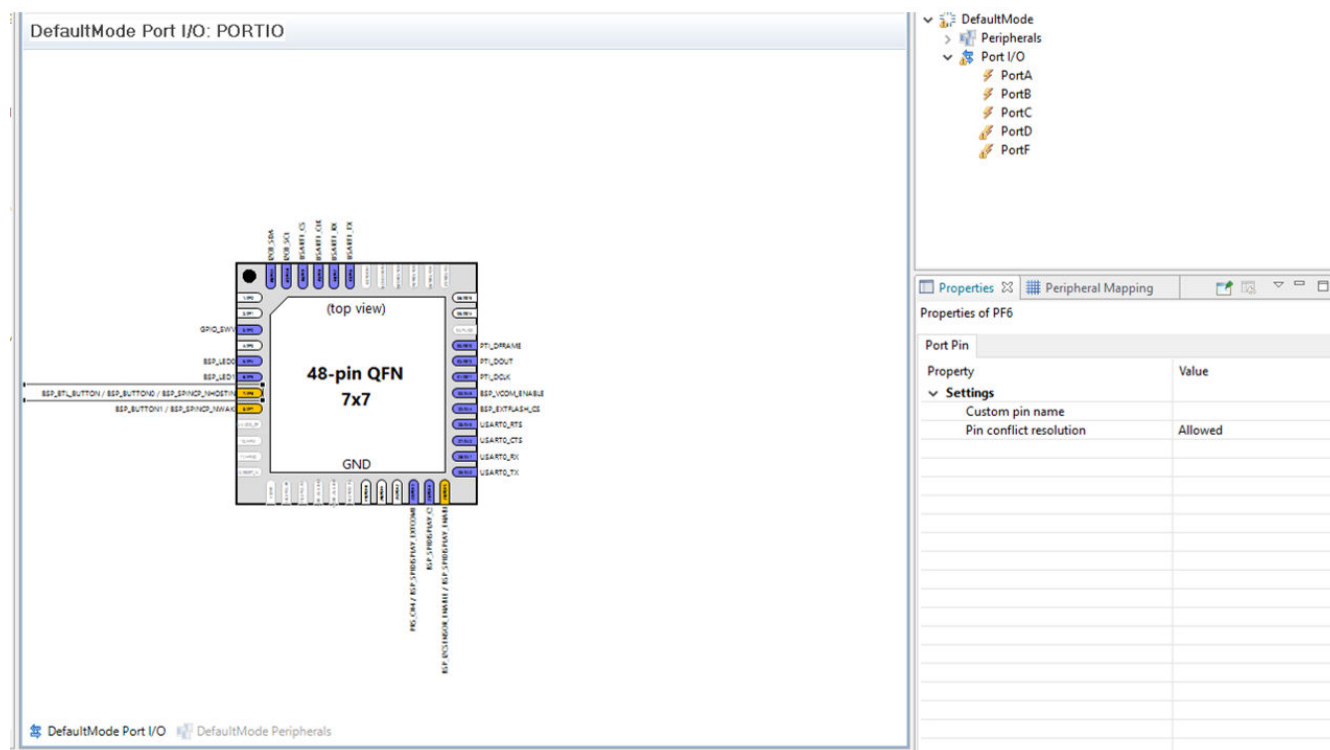
The Hardware Configurator has two views, accessed through the two **DefaultMode** tabs at the bottom of the center pane (shown in the following figure):

- Port I/O—Used to configure the pin locations and port pins
- Peripherals—Used to configure modules such as Timers, USARTs, and HAL peripherals. Most configuration changes can be made through this tab.



### 3.1 Port I/O

The **Port I/O** tab displays a package drawing for the selected device. This drawing updates to display changed pin assignments. An example is shown in the following figure.



Hover the cursor over a pin to see its definition. White = unused, blue = assigned, orange = two or more signals going to the pin, conflict allowed, and red (not shown) = two or more signals going to the pin, conflict not allowed. While you can make some changes to the peripheral associated with a given pin, Silicon Labs recommends changing peripheral configuration through the peripheral tab. Use this configuration for pin customizations such as renaming the pin.

Some changes must be made in both the Port I/O and Peripherals tab. For example:

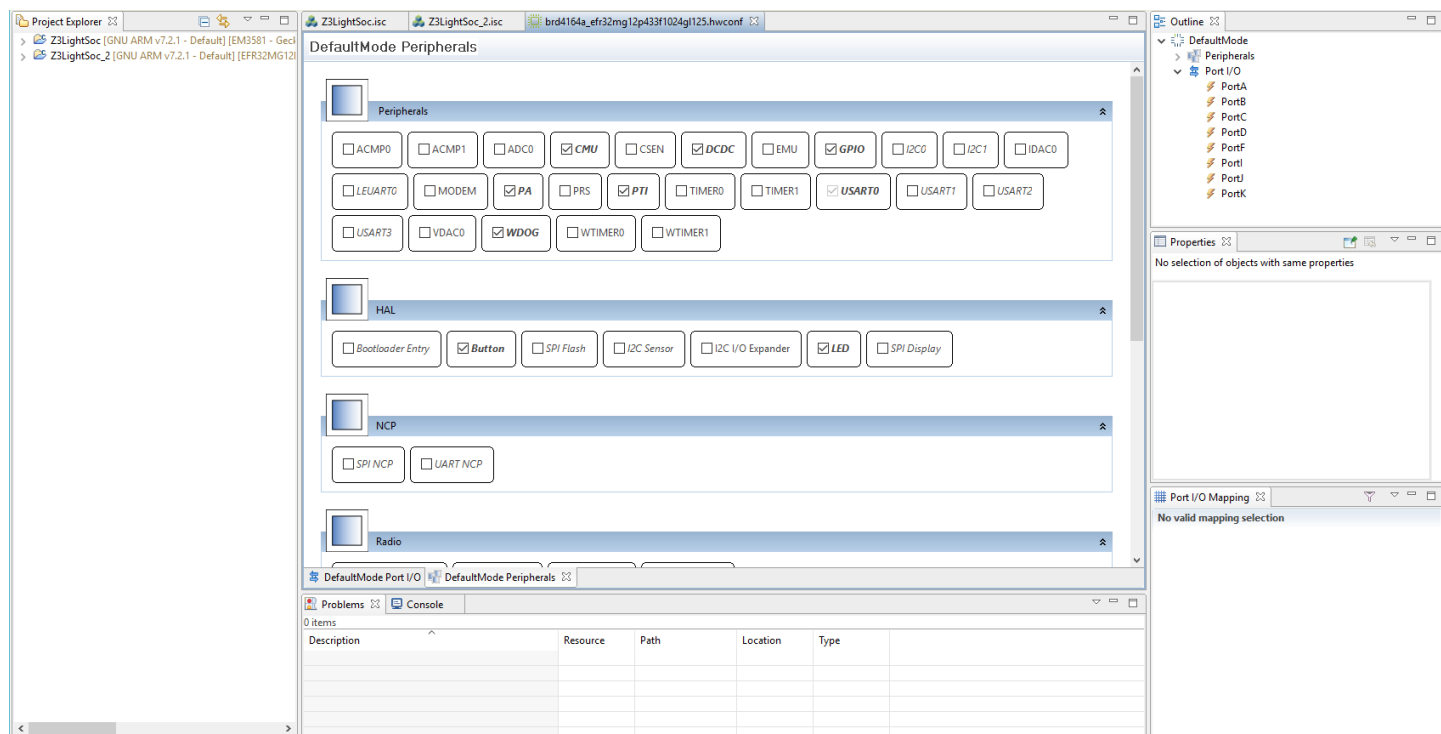
- A route can be selected in Port I/O and an APORT channel can be selected in the peripherals properties.
- ACMP output can be selected on the Port IO route selection and ACMP inputs can be configured in the Peripherals section.

A printable report can be generated by right-clicking on the pinout diagram and selecting **Pin Configuration Report**. This opens a report as a webpage in a browser that can be saved, printed, or archived. The **Module Configuration Report** option generates a similar set of tables organized by module rather than by pin order.

**Note:** To share information about the hardware configurations for a device Silicon Labs recommends sharing the .hwconf file itself.

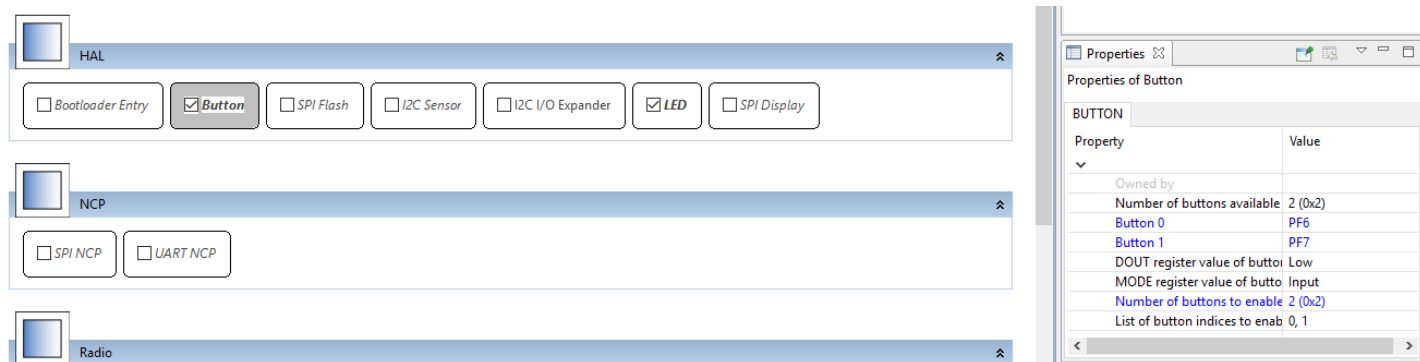
## 3.2 Peripherals

The Peripherals tab contains boxes, each of which represents either a hardware peripheral or a software concept that is HAL-related but not physically on the chip. Peripheral boxes are organized into groups. Collapse or expand a group by clicking the arrow icon on the upper right of the group. Hover your cursor over a box to see information about the peripheral.



Click on a box to see its properties in the Properties pane. Properties can either have a drop-down menu with the available selections or a text input box for a numeric or text field. After making a selection for a property, click away from the property or press Enter to ensure the property value change occurs. If you cannot disable a peripheral it may be owned by another peripheral. Go to the settings for that peripheral to free up any dependent peripherals.

**Note:** The defines for the peripheral are only generated if the checkbox on the peripheral box is checked. For example, the interface for a project based on the Silicon Labs Zigbee Z3LightSoc example has a HAL group with a Button peripheral. Click on that peripheral to see its properties.



# Smart. Connected. Energy-Friendly.



**IoT Portfolio**  
[www.silabs.com/products](http://www.silabs.com/products)



**Quality**  
[www.silabs.com/quality](http://www.silabs.com/quality)



**Support & Community**  
[www.silabs.com/community](http://www.silabs.com/community)

## Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

**Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit [www.silabs.com/about-us/inclusive-lexicon-project](http://www.silabs.com/about-us/inclusive-lexicon-project)**

## Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect®, n-Link®, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.  
400 West Cesar Chavez  
Austin, TX 78701  
USA

[www.silabs.com](http://www.silabs.com)