AN1115: Configuring Peripherals for 32-Bit Devices in Simplicity Studio

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. The features described here do not apply to the EM3x family, to the Bluetooth SDK, or to the 32-Bit MCU SDK.

If you are developing for 8-bit devices, see *AN0823: Configuring Peripherals for 8-Bit Devices in Simplicity Studio*. The underlying Hardware Configurator data model is quite different from that for 32-bit devices. If you are developing with the Bluetooth SDK, see *UG136: Silicon Labs Bluetooth® C Application Developers Guide*.

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.

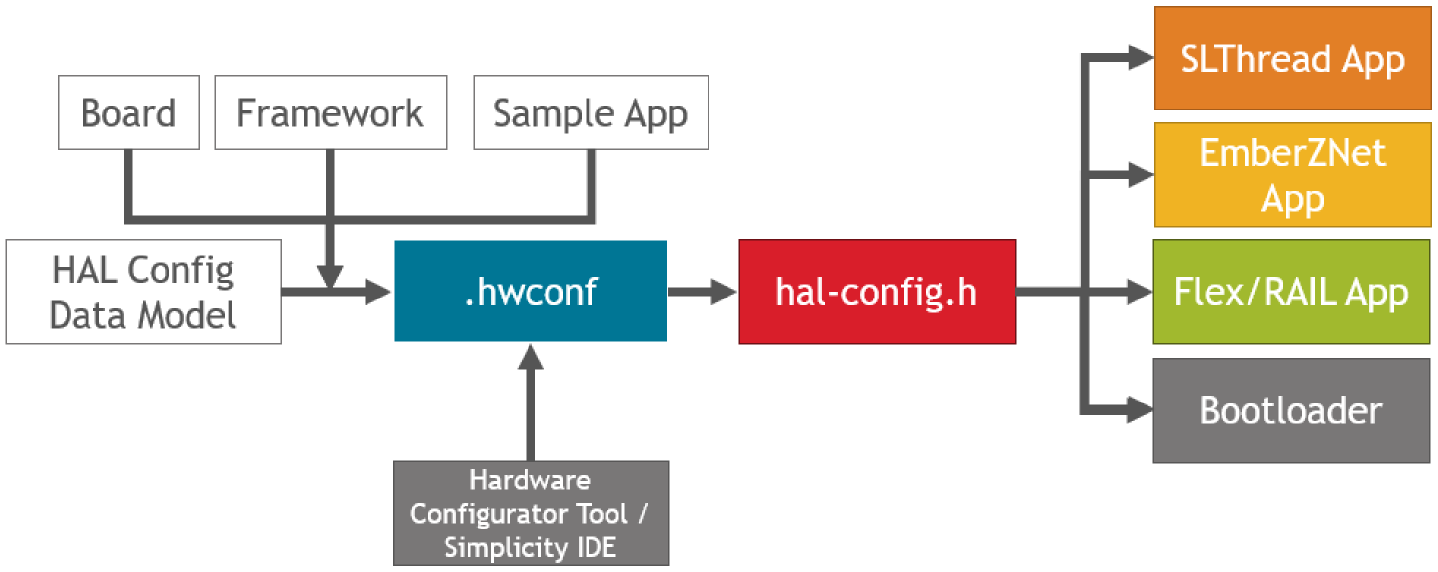
1 Introduction

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, Silicon Labs Thread, 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, Silicon Labs Thread, 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\sdks\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, Silicon Labs Thread, 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
- Silicon Labs Thread 2.6.0 or higher
- Flex 2.0.0 or higher.

Note: 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.

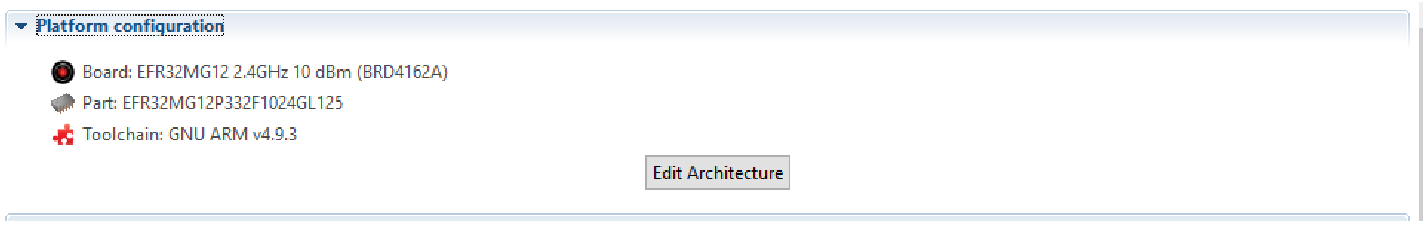
Not all Hardware Configurator options can be modified through the Simplicity IDE. The Hardware Configurator tool provides access to all options. In general, Silicon Labs recommends that you configure your project by enabling and disabling the relevant plugins, and then customize peripherals through the 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. The architecture shows:

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



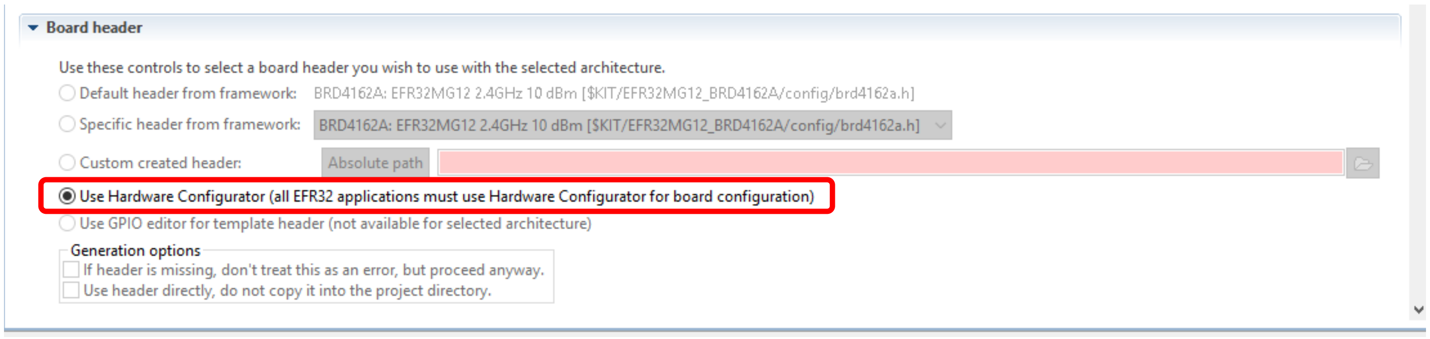
Platform configuration

Board: EFR32MG12 2.4GHz 10 dBm (BRD4162A)
Part: EFR32MG12P332F1024GL125
Toolchain: GNU ARM v4.9.3

Note: 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 Configuration

The HAL tab (HAL Configuration tab in EmberZNet) contains hooks to the Hardware Configurator. 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. Those options have been retained for compatibility with the EM35x and are enabled when that is the development target. Otherwise, the interface reflects the fact that Hardware Configurator is now working directly with the .hwconf file.


Board header

Use these controls to select a board header you wish to use with the selected architecture.

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]

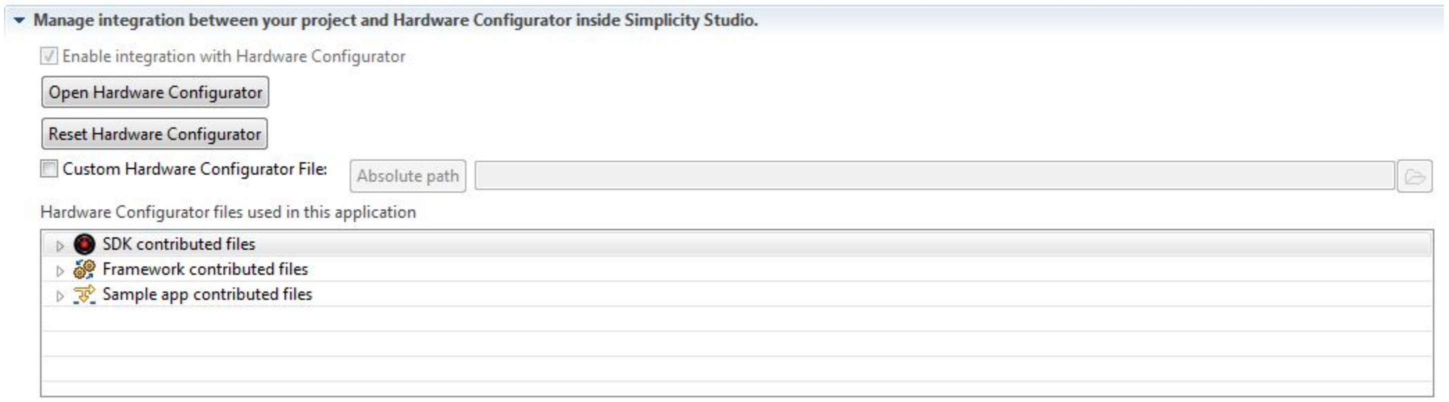
Custom created header: Absolute path

[✓ Use Hardware Configurator (all EFR32 applications must use Hardware Configurator for board configuration)]

Use GPIO editor for template header (not available for selected architecture)

Generation options
- 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.


Manage integration between your project and Hardware Configurator inside Simplicity Studio.

[✓] Enable integration with Hardware Configurator

Open Hardware Configurator

Reset Hardware Configurator

Custom Hardware Configurator File: Absolute path

Hardware Configurator files used in this application
- 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.

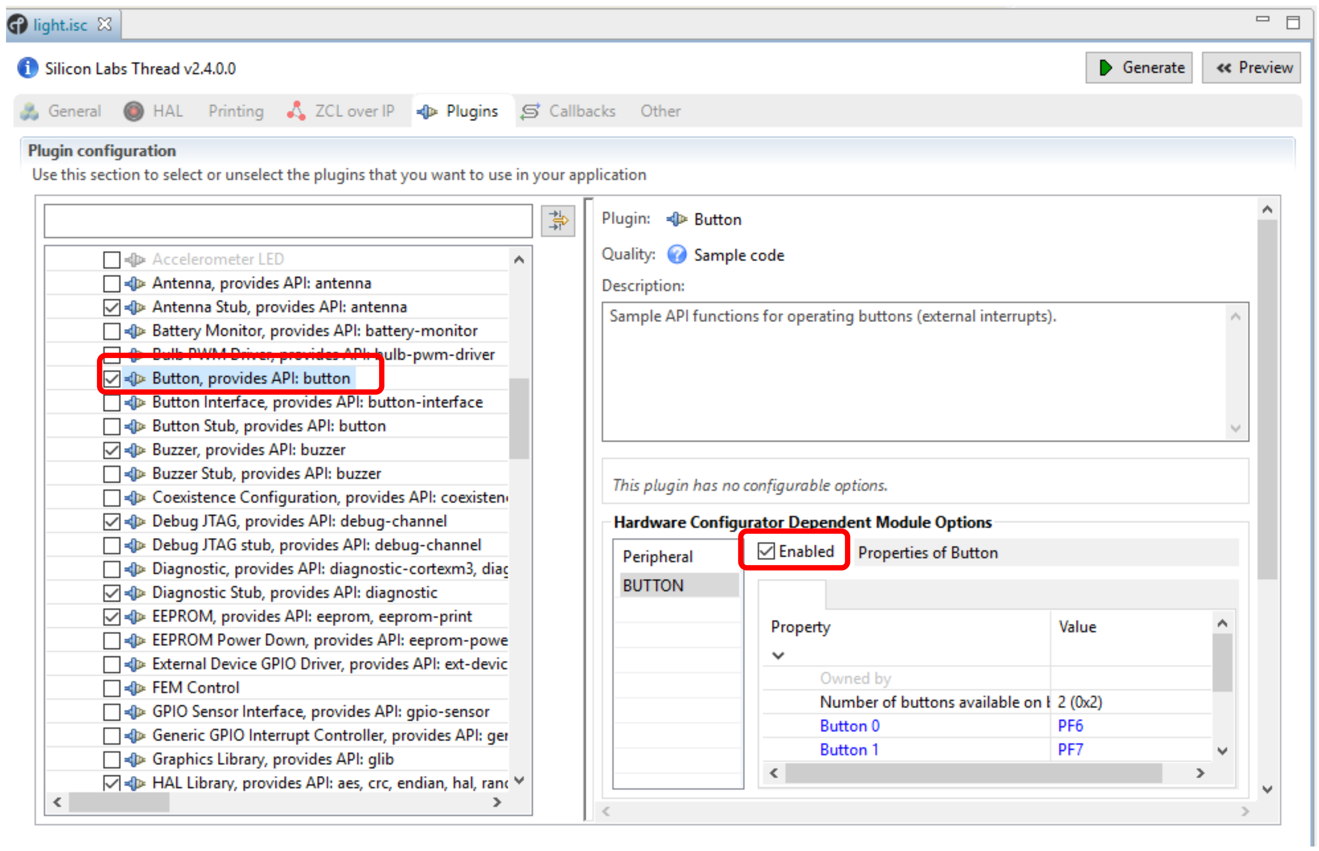
Finally click **[Reset Hardware Configurator]** to go back to the defaults based on the input files. This deletes any manual changes to the hardware configuration of the project.

2.2 Plugin Configuration

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

2.2.1 Using the Interface

The following figure shows the HAL Button plugin from the Silicon Labs Thread Light 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.



Silicon Labs Thread v2.4.0.0

Plugin configuration
Use this section to select or unselect the plugins that you want to use in your application

☐ Accelerometer LED
☐ Antenna, provides API: antenna
☑ Antenna Stub, provides API: antenna
☐ Battery Monitor, provides API: battery-monitor
☐ Bulb PWM Driver, provides API: bulb-pwm-driver
☑ Button, provides API: button
☐ Button Interface, provides API: button-interface
☐ Button Stub, provides API: button
☑ Buzzer, provides API: buzzer
☐ Buzzer Stub, provides API: buzzer
☐ Coexistence Configuration, provides API: coexisten
☑ Debug JTAG, provides API: debug-channel
☐ Debug JTAG stub, provides API: debug-channel
☐ Diagnostic, provides API: diagnostic-cortexm3, diag
☑ Diagnostic Stub, provides API: diagnostic
☑ EEPROM, provides API: eeprom, eeprom-print
☐ EEPROM Power Down, provides API: eeprom-powe
☐ External Device GPIO Driver, provides API: ext-devic
☐ FEM Control
☐ GPIO Sensor Interface, provides API: gpio-sensor
☐ Generic GPIO Interrupt Controller, provides API: ger
☐ Graphics Library, provides API: glib
☑ HAL Library, provides API: aes, crc, endian, hal, ranc

Plugin: Button
Quality: Sample code
Description:
Sample API functions for operating buttons (external interrupts).

This plugin has no configurable options.

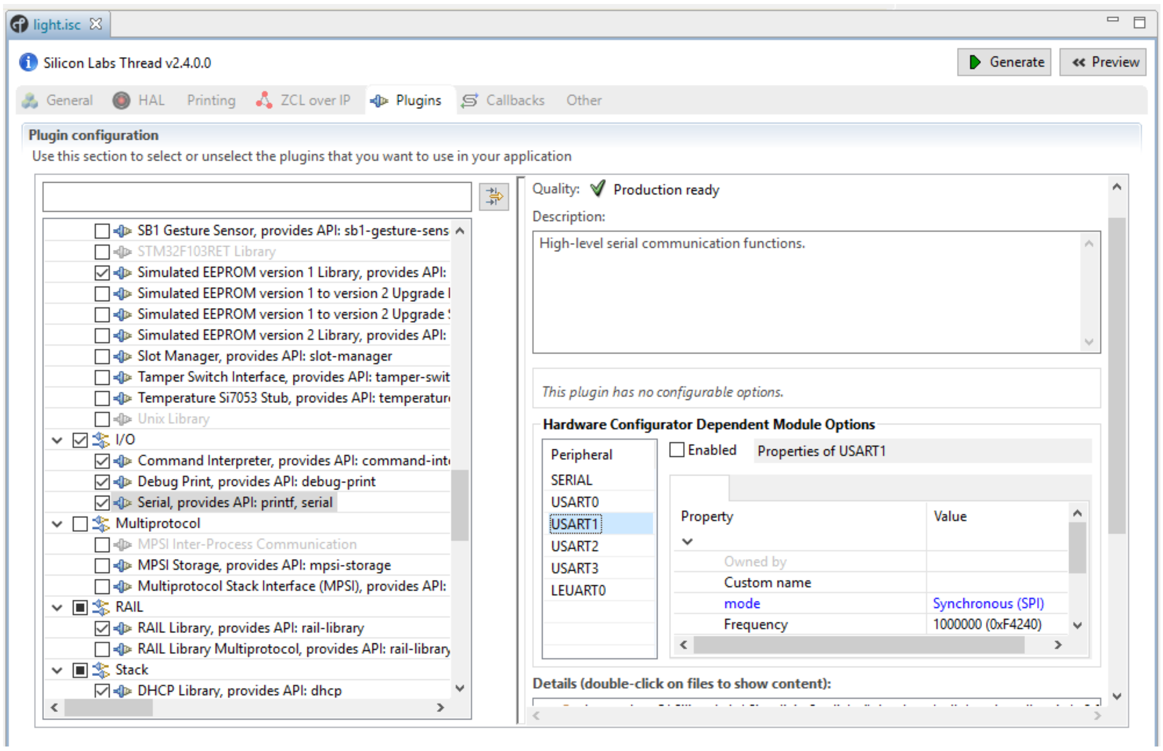
Hardware Configurator Dependent Module Options

Peripheral: BUTTON

☑ Enabled — Properties of Button

Property	Value
Owned by	
Number of buttons available on t	2 (0x2)
Button 0	PF6
Button 1	PF7

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 (shown in the following figure) have several. Simply select one of the peripherals from the list to see its configuration.



Silicon Labs Thread v2.4.0.0

Plugin configuration

Use this section to select or unselect the plugins that you want to use in your application

- [ ] SB1 Gesture Sensor, provides API: sb1-gesture-sens
- [ ] STM32F103RET Library
- [✓] Simulated EEPROM version 1 Library, provides API:
- [ ] Simulated EEPROM version 1 to version 2 Upgrade I
- [ ] Simulated EEPROM version 1 to version 2 Upgrade S
- [ ] Simulated EEPROM version 2 Library, provides API:
- [ ] Slot Manager, provides API: slot-manager
- [ ] Tamper Switch Interface, provides API: tamper-swit
- [ ] Temperature Si7053 Stub, provides API: temperatur
- [ ] Unix Library
- [✓] I/O
  - [✓] Command Interpreter, provides API: command-int
  - [✓] Debug Print, provides API: debug-print
  - [✓] Serial, provides API: printf, serial
- [ ] Multiprotocol
  - [ ] MPSI Inter-Process Communication
  - [ ] MPSI Storage, provides API: mpsi-storage
  - [ ] Multiprotocol Stack Interface (MPSI), provides API:
- [■] RAIL
  - [✓] RAIL Library, provides API: rail-library
  - [ ] RAIL Library Multiprotocol, provides API: rail-library
- [■] Stack
  - [✓] DHCP Library, provides API: dhcp

Quality: ✓ Production ready

Description:

High-level serial communication functions.

This plugin has no configurable options.

Hardware Configurator Dependent Module Options

Peripheral
SERIAL
USART0
USART1
USART2
USART3
LEUART0

[ ] Enabled   Properties of USART1

Property	Value
Owned by	
Custom name	
mode	Synchronous (SPI)
Frequency	1000000 (0xF4240)

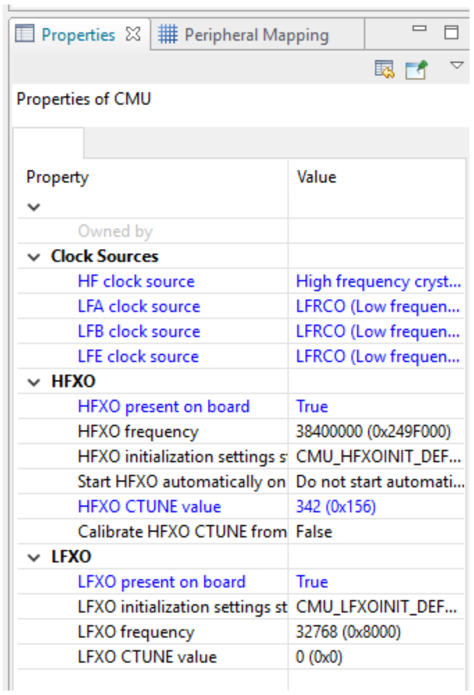
Details (double-click on files to show content):

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.



Properties of CMU

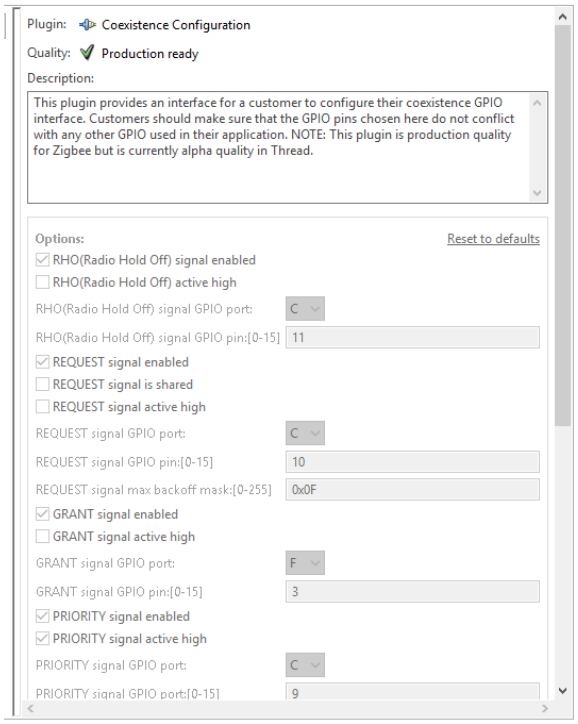
Property	Value
Owned by	
Clock Sources	
HF clock source	High frequency cryst...
LFA clock source	LFRCO (Low frequen...
LFB clock source	LFRCO (Low frequen...
LFE clock source	LFRCO (Low frequen...
HFXO	
HFXO present on board	True
HFXO frequency	38400000 (0x249F000)
HFXO initialization settings s	CMU_HFXOINIT_DEF...
Start HFXO automatically on	Do not start automati...
HFXO CTUNE value	342 (0x156)
Calibrate HFXO CTUNE from	False
LFXO	
LFXO present on board	True
LFXO initialization settings st	CMU_LFXOINIT_DEF...
LFXO frequency	32768 (0x8000)
LFXO CTUNE value	0 (0x0)

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
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. The Coexistence Configuration plugin options in Silicon Labs Thread SDK version 2.3 (a) and version 2.4 (b) are shown as an example.

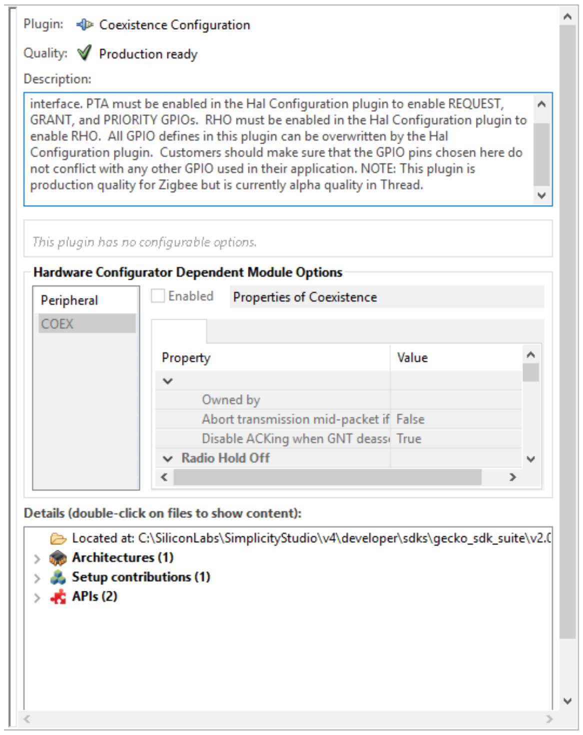


Plugin: Coexistence Configuration
Quality: Production ready
Description:
This plugin provides an interface for a customer to configure their coexistence GPIO interface. Customers should make sure that the GPIO pins chosen here do not conflict with any other GPIO used in their application. NOTE: This plugin is production quality for Zigbee but is currently alpha quality in Thread.

Options: Reset to defaults

☑ RHO(Radio Hold Off) signal enabled	
☐ RHO(Radio Hold Off) active high	
RHO(Radio Hold Off) signal GPIO port:	C
RHO(Radio Hold Off) signal GPIO pin:[0-15]	11
☑ REQUEST signal enabled	
☐ REQUEST signal is shared	
☐ REQUEST signal active high	
REQUEST signal GPIO port:	C
REQUEST signal GPIO pin:[0-15]	10
REQUEST signal max backoff mask:[0-255]	0x0F
☑ GRANT signal enabled	
☐ GRANT signal active high	
GRANT signal GPIO port:	F
GRANT signal GPIO pin:[0-15]	3
☑ PRIORITY signal enabled	
☑ PRIORITY signal active high	
PRIORITY signal GPIO port:	C
PRIORITY signal GPIO port:[0-15]	9

a)



Plugin: Coexistence Configuration
Quality: Production ready
Description:
interface. PTA must be enabled in the Hal Configuration plugin to enable REQUEST, GRANT, and PRIORITY GPIOs. RHO must be enabled in the Hal Configuration plugin to enable RHO. All GPIO defines in this plugin can be overwritten by the Hal Configuration plugin. Customers should make sure that the GPIO pins chosen here do not conflict with any other GPIO used in their application. NOTE: This plugin is production quality for Zigbee but is currently alpha quality in Thread.

This plugin has no configurable options.

Hardware Configurator Dependent Module Options

Peripheral: COEX
☐ Enabled — Properties of Coexistence

Property	Value
Owned by	
Abort transmission mid-packet if	False
Disable ACKing when GNT deass	True
Radio Hold Off	

Details (double-click on files to show content):

Located at: C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v2.0
Architectures (1)
Setup contributions (1)
APIs (2)

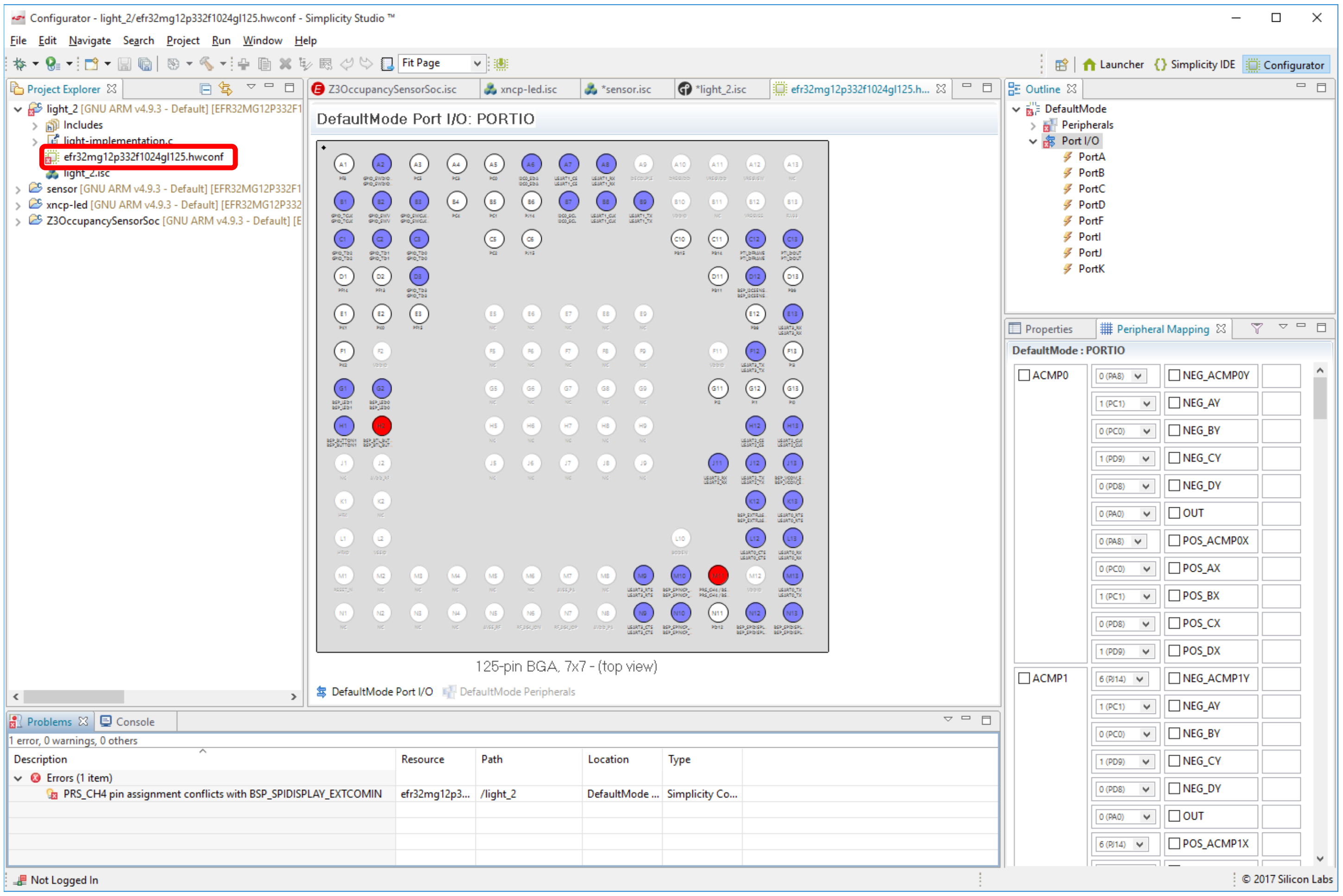
b)

Plugins in this category include:

- 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.


Configurator - light_2/efr32mg12p332f1024gl125.hwconf - Simplicity Studio ™
File Edit Navigate Search Project Run Window Help
Fit Page
Launcher
Simplicity IDE
Configurator
Project Explorer
light_2 [GNU ARM v4.9.3 - Default] [EFR32MG12P332F1
Includes
light-implementation.c
efr32mg12p332f1024gl125.hwconf
light_2.isc
sensor [GNU ARM v4.9.3 - Default] [EFR32MG12P332F1
xncp-led [GNU ARM v4.9.3 - Default] [EFR32MG12P332
Z3OccupancySensorSoc [GNU ARM v4.9.3 - Default] [E
Z3OccupancySensorSoc.isc
xncp-led.isc
*sensor.isc
*light_2.isc
efr32mg12p332f1024gl125.h...
DefaultMode Port I/O: PORTIO
125-pin BGA, 7x7 - (top view)
DefaultMode Port I/O
DefaultMode Peripherals
Outline
DefaultMode
Peripherals
Port I/O
PortA
PortB
PortC
PortD
PortF
PortI
PortJ
PortK
Properties
Peripheral Mapping
DefaultMode : PORTIO
Problems
Console
Not Logged In
© 2017 Silicon Labs

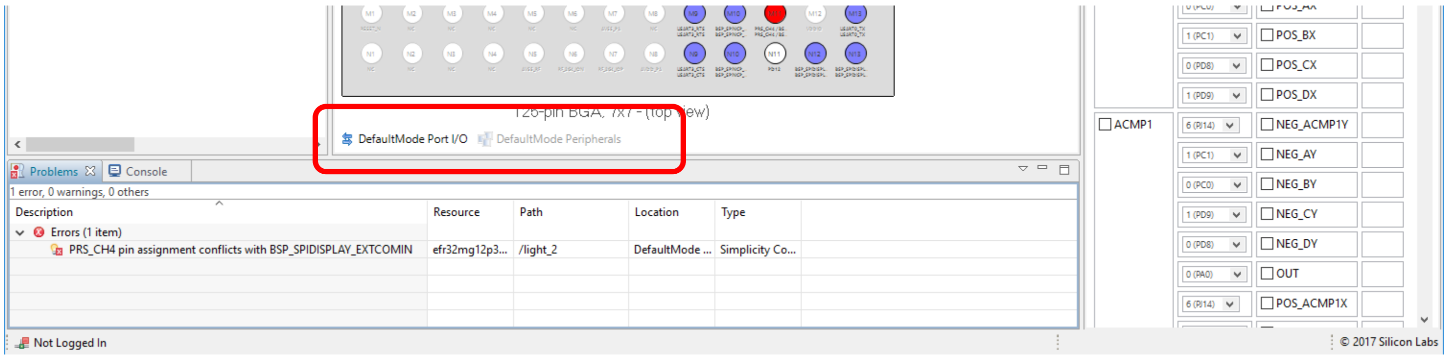
1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
PRS_CH4 pin assignment conflicts with BSP_SPIDISPLAY_EXTCOMIN	efr32mg12p3...	/light_2	DefaultMode ...	Simplicity Co...

ACMP0	0 (PA8)	NEG_ACMP0Y
	1 (PC1)	NEG_AY
	0 (PC0)	NEG_BY
	1 (PD9)	NEG_CY
	0 (PD8)	NEG_DY
	0 (PA0)	OUT
	0 (PA8)	POS_ACMP0X
	0 (PC0)	POS_AX
	1 (PC1)	POS_BX
	0 (PD8)	POS_CX
	1 (PD9)	POS_DX
ACMP1	6 (PI14)	NEG_ACMP1Y
	1 (PC1)	NEG_AY
	0 (PC0)	NEG_BY
	1 (PD9)	NEG_CY
	0 (PD8)	NEG_DY
	0 (PA0)	OUT
	6 (PI14)	POS_ACMP1X

The Hardware Configurator has two views, accessed through the two **DefaultMode** tabs at the bottom of the center pane:

- 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.

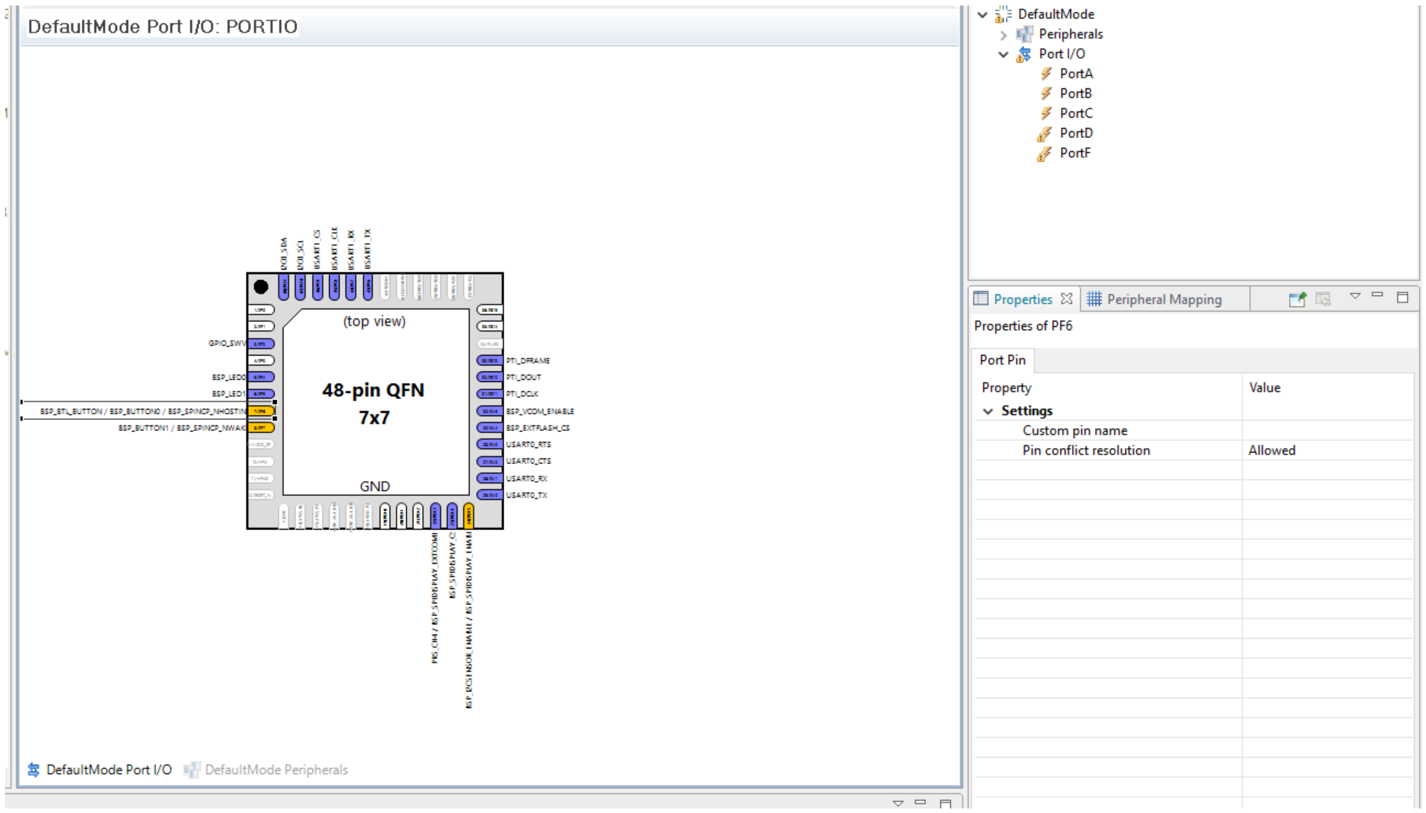


1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
PRS_CH4 pin assignment conflicts with BSP_SPIDISPLAY_EXTCOMIN	efr32mg12p3...	/light_2	DefaultMode ...	Simplicity Co...

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.



DefaultMode Port I/O: PORTIO

- DefaultMode
  - Peripherals
  - Port I/O
    - PortA
    - PortB
    - PortC
    - PortD
    - PortF

Properties of PF6

Port Pin

Property	Value
Settings	
Custom pin name	
Pin conflict resolution	Allowed

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.

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

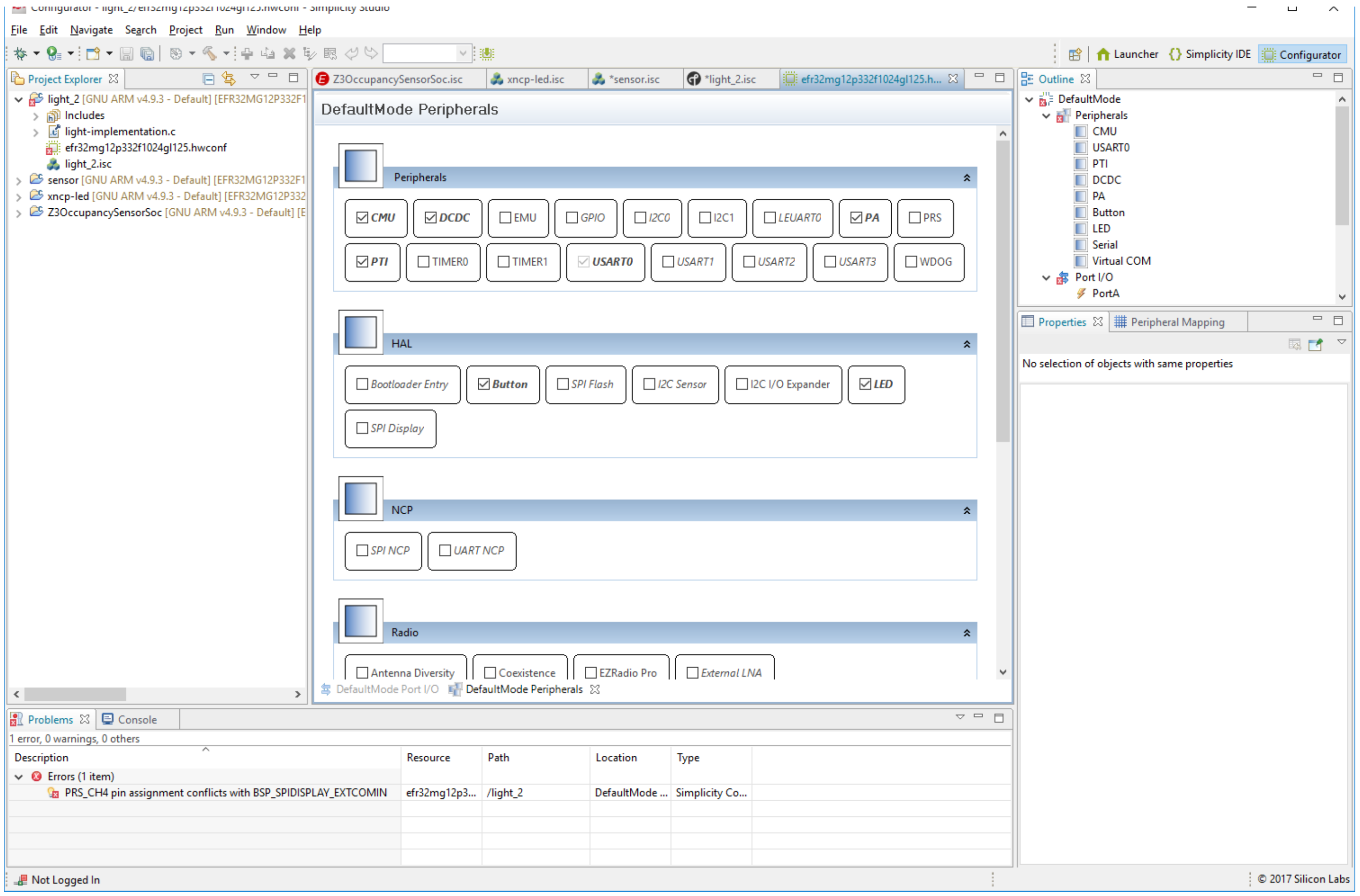
- A route can be selected in Port I/O and an APORT channel can be selected in the peripherals properties.
- ACMP output can be selected on the Port IO route selection and ACMP inputs can be configured in the Peripherals section.

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.



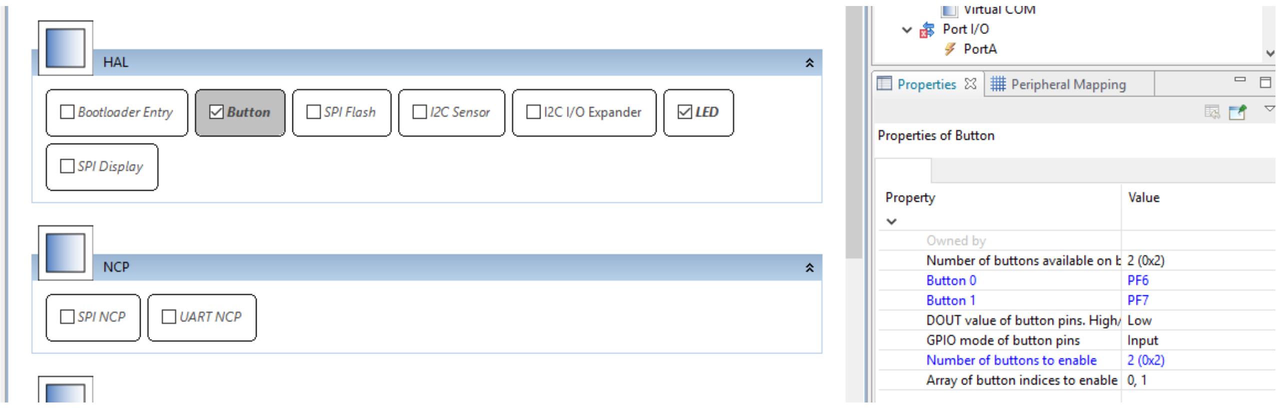
No selection of objects with same properties

1 error, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (1 item)				
PRS_CH4 pin assignment conflicts with BSP_SPIDISPLAY_EXTCOMIN	efr32mg12p3...	/light_2	DefaultMode ...	Simplicity Co...

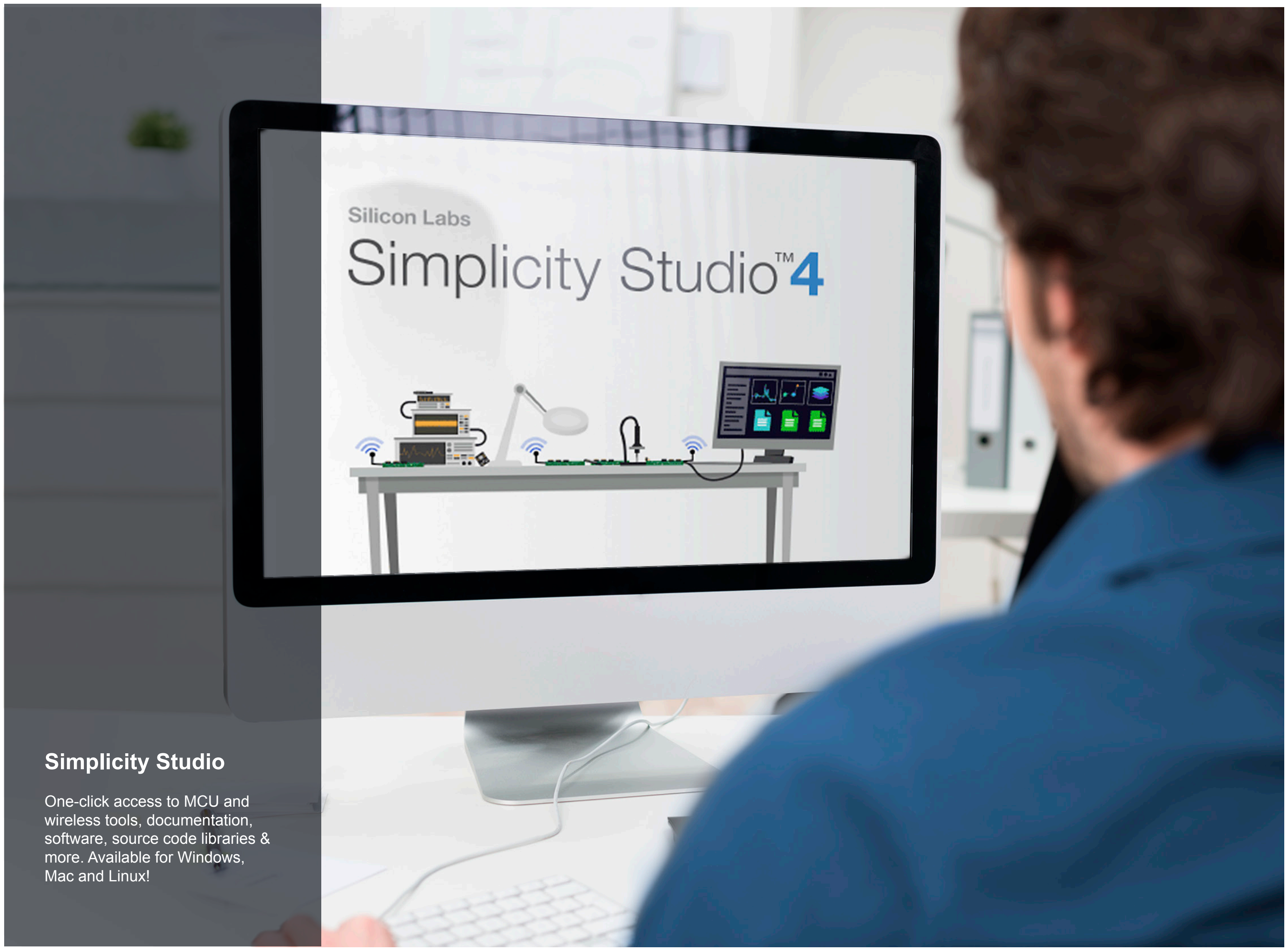
Click on a box to see its properties in the Properties pane. Properties can either have a drop-down menu with the available selections or a text input box for a numeric or text field. After making a selection for a property, click away from the property or press Enter to ensure the property value change 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 Thread Light example has a HAL group with a Button peripheral. Click on that peripheral to see its properties.



Properties of Button

Property	Value
Owned by	
Number of buttons available on t	2 (0x2)
Button 0	PF6
Button 1	PF7
DOUT value of button pins. High/	Low
GPIO mode of button pins	Input
Number of buttons to enable	2 (0x2)
Array of button indices to enable	0, 1



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com