# AN0823: Simplicity Configurator User's Guide

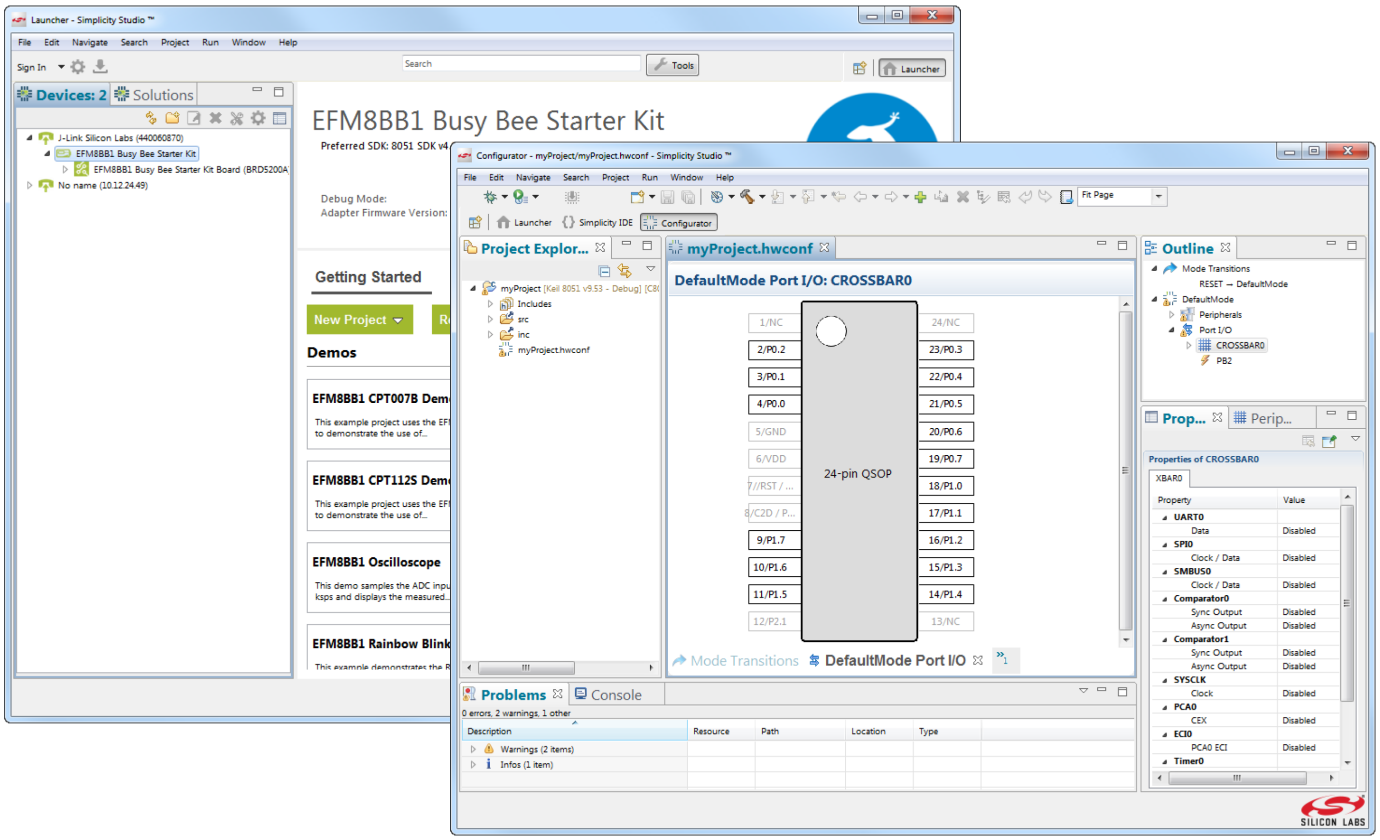
Simplicity Configurator is part of Simplicity Studio and greatly simplifies EFM32, EFM8, and C8051 MCU peripheral initialization by presenting peripherals and peripheral properties in a graphical user interface.

The majority of the initialization firmware can be generated by selecting peripherals and property values from combo boxes or entering register values in text boxes. Some peripherals provide calculators, such as baud rate calculators, timer overflow rate calculators, and SPI clock rate calculators, that can be used to automatically confirm the necessary reload register value to generate the specified clock rate. Configurator also provides real-time validation of properties to ensure that a configuration is valid before downloading code to the MCU.

Download and install Simplicity Studio from here: http://www.silabs.com/simplicity-studio.

**KEY POINTS**

- Simplicity Configurator automatically generates code for hardware peripherals and port I/O.
- Configurator highlights errors and warnings to prevent invalid configurations and code.
- Mode transitions enable movement between different peripheral configurations.
- Automatically-generated code is visible and editable to provide as much flexibility as possible.


Launcher - Simplicity Studio™
EFM8BB1 Busy Bee Starter Kit
Getting Started
New Project
Demos
Configurator - myProject/myProject.hwconf - Simplicity Studio™
Project Explorer
myProject.hwconf
DefaultMode Port I/O: CROSSBAR0
24-pin QSOP
Mode Transitions
DefaultMode Port I/O
Outline
Properties of CROSSBAR0
Problems
Console
0 errors, 2 warnings, 1 other

## 1. Relevant Resources

- Simplicity IDE Guide—In Simplicity IDE, select [**Help**]>[**Help Contents**] to display the Simplicity IDE Guide as well as any installed documentation including data sheets and user guides.
- Simplicity Configurator Guide—In the Simplicity IDE, select [**Help**]>[**Help Contents**] to display this guide.
- *AN0822: Simplicity Studio User Guide*—In addition to the documentation within the tool itself, this document provides a discussion of the Simplicity Studio tool. Application Notes are available on http://www.silabs.com/32bit-appnotes and http://www.silabs.com/8bit-appnotes.
- *AN0821: Simplicity Studio C8051F85x Walkthrough*—This document provides step-by-step instructions to create a project using the C8051F850 ToolStick, Simplicity Configurator, and Simplicity IDE. Application Notes are available on http://www.silabs.com/32bit-appnotes and http://www.silabs.com/8bit-appnotes.

## 2. Creating a New Configurator Project

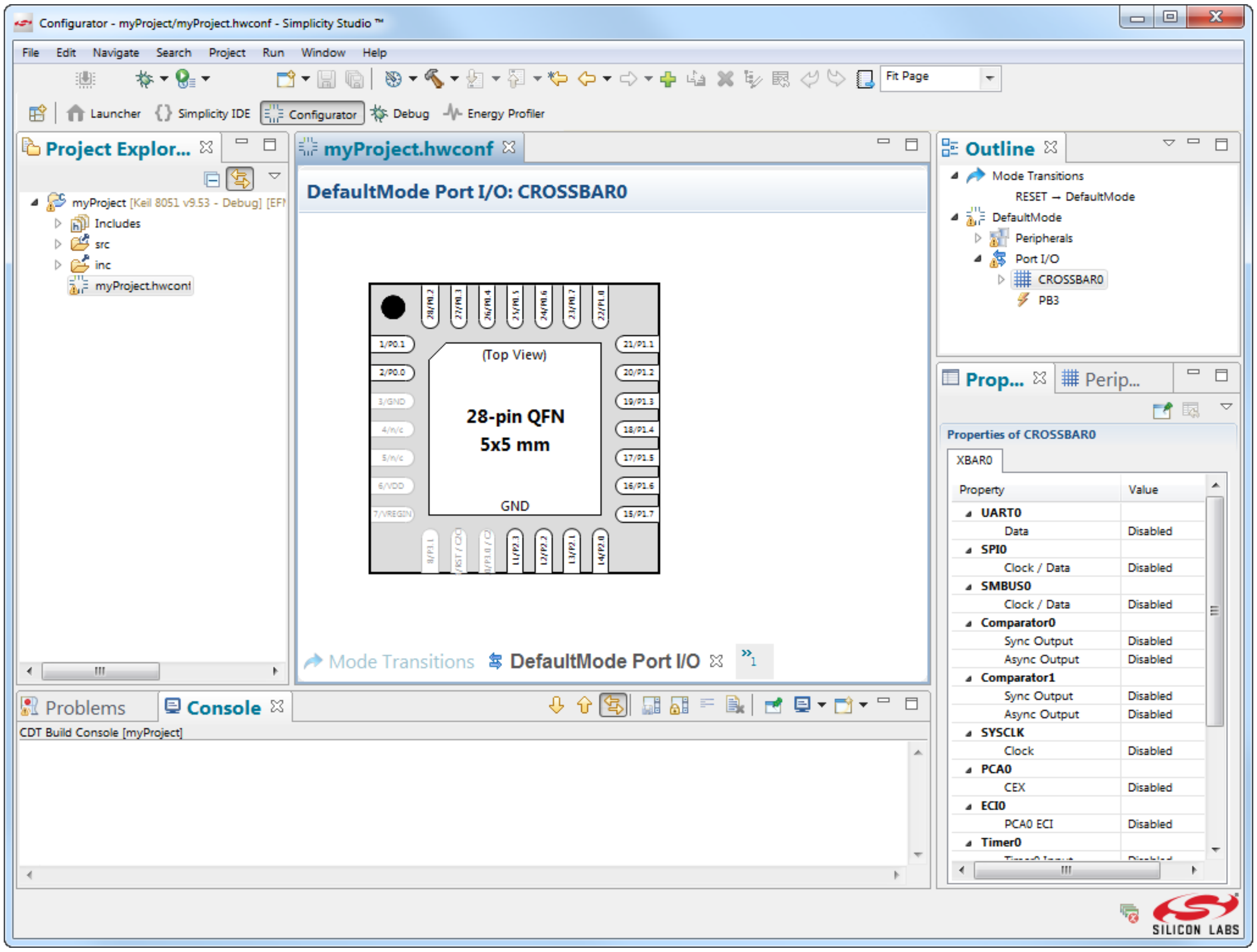
Create a Simplicity Configurator project from the Simplicity Studio screen by searching or detecting a device in the main Simplicity Studio screen and clicking the [**Configurator**] tile. From within the Simplicity IDE, create a new project by selecting [**File**]>[**New**]>[**Silicon Labs MCU Project**] from the menu bar.

After creating a new Simplicity Configurator project, Studio will automatically switch to the [**Configurator**] perspective. This perspective is tailored specifically for use with Configurator to initialize MCU peripherals. Special views in this perspective allow peripheral registers to be configured. Switch between the [**Configurator**], [**Development**], [**Debug**], and [**Simplicity**] perspectives at any time by clicking the appropriate perspective button in the top right. If the desired perspective is missing, click the [**Open Perspective**]>[**Other...**] button to launch the [**Open Perspective**] dialog to choose from a list of all perspectives.

The available views in a Simplicity Configurator project are as follows:

- Mode Transitions—Used to define different MCU peripheral initialization states, such as default mode, from reset and other optional modes (example: low-power mode). This editor can also be used to define transitions between states.
- Port I/O—Used to configure the pin locations, crossbar, and port pins.
- Peripherals—Used to configure hardware peripherals such as ADCs, Timers, LEUART, PCAs, etc.

Double clicking on `*.hwconf` file will launch the Configurator editor in the center of the screen. The [**Port I/O**] tab is selected by default after creating a new project.



Properties of CROSSBAR0

XBAR0

| Property | Value |
|---|---|
| UART0 | |
| Data | Disabled |
| SPI0 | |
| Clock / Data | Disabled |
| SMBUS0 | |
| Clock / Data | Disabled |
| Comparator0 | |
| Sync Output | Disabled |
| Async Output | Disabled |
| Comparator1 | |
| Sync Output | Disabled |
| Async Output | Disabled |
| SYSCLK | |
| Clock | Disabled |
| PCA0 | |
| CEX | Disabled |
| ECI0 | |
| PCA0 ECI | Disabled |
| Timer0 | |

**Figure 2.1. New Simplicity Configurator Project**

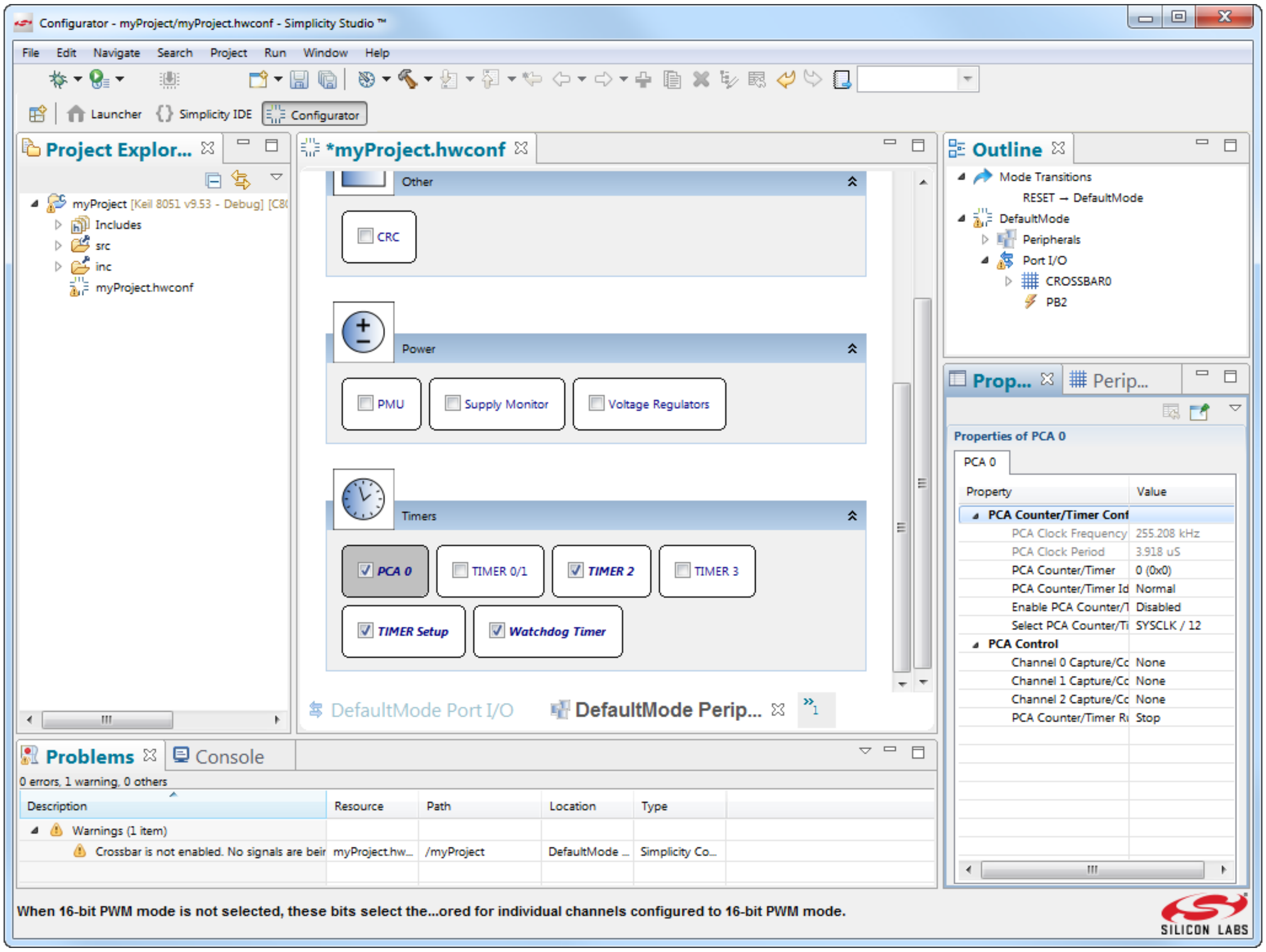
## 3. Configuring Peripherals

The [**Peripherals**] tab enables configuration of the hardware peripherals on a device like the ADC, comparators, Timers, etc. The peripherals are organized into logical groups: Analog, Communications, Core, Clocking, Power, Timers, and Other. These groups can be collapsed or expanded using the arrow icon in the upper right of the group.

To configure a peripheral, click on the peripheral box. This will select the peripheral and open the peripheral in the [**Properties**] view. Properties can either have a drop-down menu with the available selections or a text input box for a numeric or text field. Properties that are grayed out are read-only, and these properties are typically to provide more information about the configuration of a peripheral. After making a selection for a property, click away from the property or press [**Enter**] to ensure the property value change occurs.

For example, clicking the Timers peripheral opens the timers in different tabs in the [**Properties**] view. The Timer 3 tab has several properties that can be modified, like [**Clock Source**], [**Mode**], and [**Target Overflow Frequency**]. Modifying any of these properties will update the [**Timer Reload Overflow Period**] and [**Timer Reload Overflow Frequency**] read-only properties, which display the calculated overflow time based on the [**Target Overflow Frequency**] and the [**Clock Source**] settings. These read- only fields and calculators depend on the [**Mode**] and [**Run Control**] state. Calculators for reload value are for auto-reload mode.

**Note:** Peripherals will not have code generation enabled unless the peripheral checkbox is active.



Properties of PCA 0

PCA 0

| Property | Value |
|---|---|
| PCA Counter/Timer Conf | |
| PCA Clock Frequency | 255.208 kHz |
| PCA Clock Period | 3.918 uS |
| PCA Counter/Timer | 0 (0x0) |
| PCA Counter/Timer Id | Normal |
| Enable PCA Counter/T | Disabled |
| Select PCA Counter/Ti | SYSCLK / 12 |
| PCA Control | |
| Channel 0 Capture/Cc | None |
| Channel 1 Capture/Cc | None |
| Channel 2 Capture/Cc | None |
| PCA Counter/Timer Ru | Stop |

0 errors, 1 warning, 0 others

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|
| Warnings (1 item) | | | | |
| Crossbar is not enabled. No signals are beir | myProject.hw... | /myProject | DefaultMode ... | Simplicity Co... |

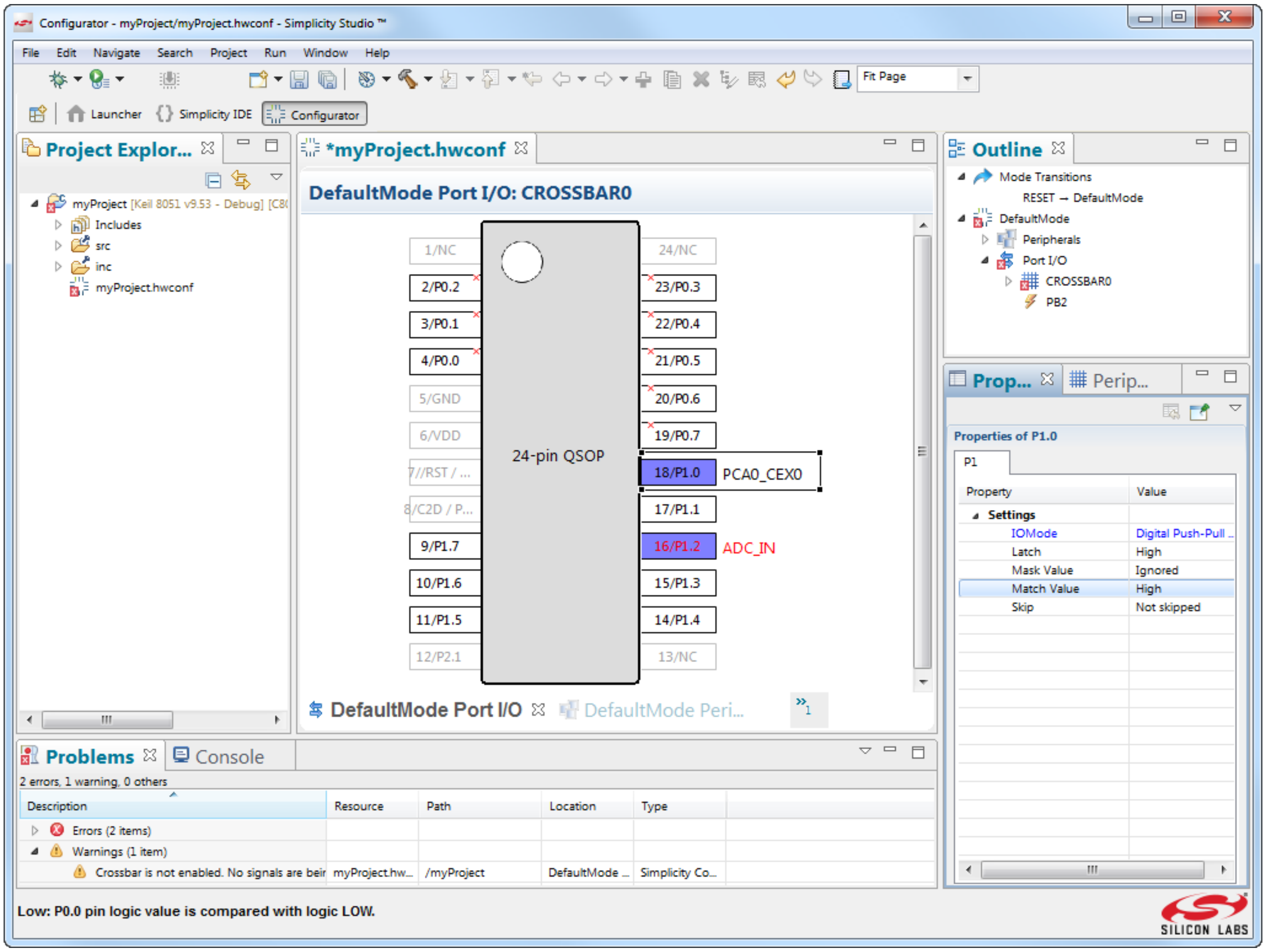
When 16-bit PWM mode is not selected, these bits select the...ored for individual channels configured to 16-bit PWM mode.

Figure 3.1. Configuring a Peripheral

# 4. Configuring I/O

The [**Port I/O**] tab displays a package drawing for the selected device. This drawing updates to display pin assignments as peripherals are enabled on the crossbar or peripherals with fixed pin assignments are enabled. For fixed pin assignments, the associated peripheral must be enabled in order for the fixed pin function to appear on the Port I/O tab. For example, enabling the ADC0 module on an EFM8 device ([**ADC 0**]>[**Enable ADC**]>[**Enabled**]) displays the ADC_IN fixed signal on the selected pin ([**ADC 0**]>[**Positive Input Selection**]>[**ADC0.0**]).

To enable a peripheral on the crossbar, enable a peripheral for code generation, click [**CROSSBAR0**] in the [**Outline**] view, and enable the checkbox next to the peripheral crossbar signals.



Properties of P1.0

P1

| Property | Value |
|---|---|
| Settings | |
| IOMode | Digital Push-Pull ... |
| Latch | High |
| Mask Value | Ignored |
| Match Value | High |
| Skip | Not skipped |

2 errors, 1 warning, 0 others

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|
| Errors (2 items) | | | | |
| Warnings (1 item) | | | | |
| Crossbar is not enabled. No signals are beir | myProject.hw... | /myProject | DefaultMode ... | Simplicity Co... |

Low: P0.0 pin logic value is compared with logic LOW.

**Figure 4.1. Configuing the I/O**

To configure a pin's properties, click on the pin on the package drawing and select the desired pin property settings. If configuing a device with a crossbar, a pin can also be skipped by right-clicking on the pin and selecting [**Skip**]. On all devices, multiple pins can be selected at once by holding [**Ctrl**] or [**Shift**] and selecting the desired pins or holding left mouse click and dragging over a group of pins. With multiple pins selected, the property changes made in the [**Properties**] view or right-click menu will apply to all of the pins. Pins can be reset to their default state by right-clicking on the pin and selecting [**Reset**].

The EFM32 products have multiple possible locations for peripheral assignments. These locations are selectable using a drop-down menu next to the corresponding set of signals in the peripheral. Any conflicts with other peripherals will be highlighted in red.

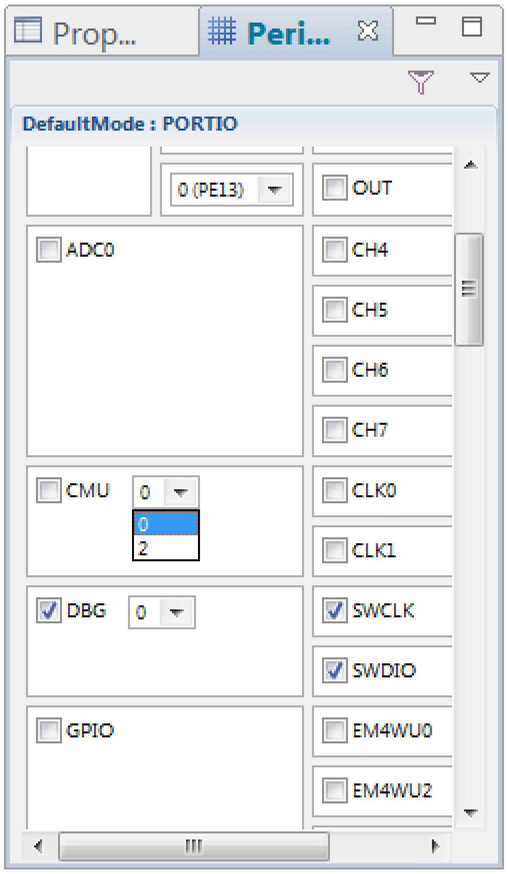
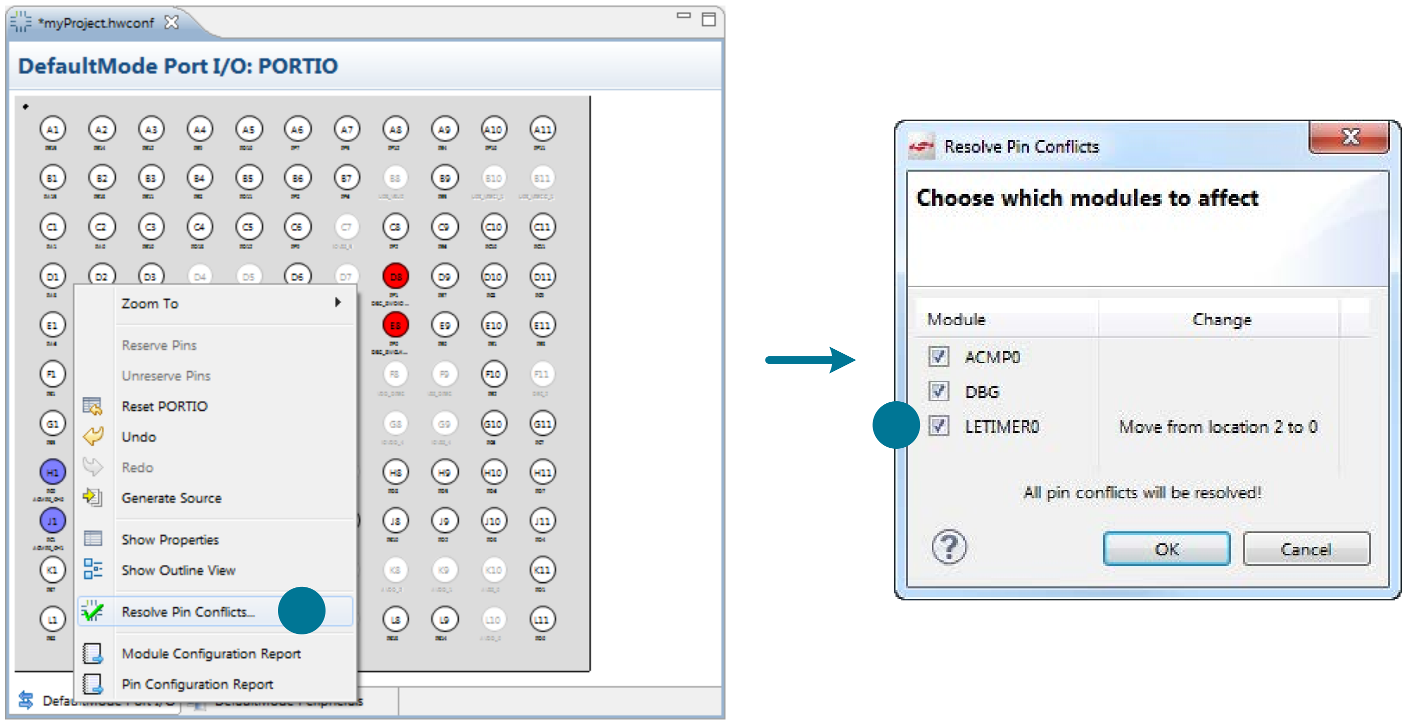


**Figure 4.2. EFM32 Signal Locations**

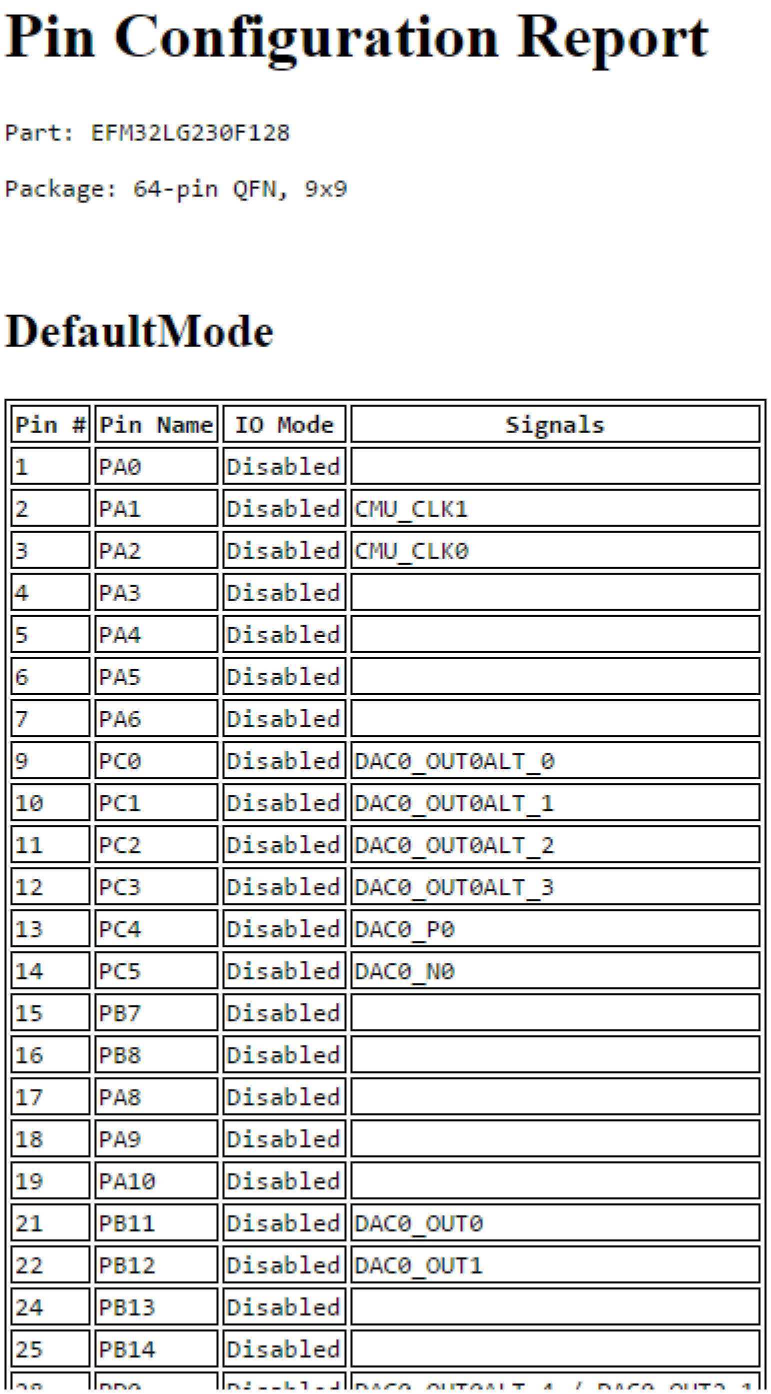
Pin locations can be selected manually or resolved using the [**Resolve Pin Conflicts...**] option in the right-click menu. This will run a pin resolver algorithm that provides the options resolving all pin conflicts, if any are available. Peripherals can be selectively removed from this algorithm by unchecking the boxes in the [**Resolve Pin Conflicts**] dialog.



| Module | Change |
|---|---|
| ACMP0 | |
| DBG | |
| LETIMER0 | Move from location 2 to 0 |

**Figure 4.3. Resolving Pin Conflicts**

Once pins have been configured for any device, a printable report can be generated by right-clicking on the pinout diagram and selecting [**Pin Configuration Report**]. This will open a report in 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.



# Pin Configuration Report

Part: EFM32LG230F128

Package: 64-pin QFN, 9x9

## DefaultMode

| Pin # | Pin Name | IO Mode | Signals |
|---|---|---|---|
| 1 | PA0 | Disabled | |
| 2 | PA1 | Disabled | CMU_CLK1 |
| 3 | PA2 | Disabled | CMU_CLK0 |
| 4 | PA3 | Disabled | |
| 5 | PA4 | Disabled | |
| 6 | PA5 | Disabled | |
| 7 | PA6 | Disabled | |
| 9 | PC0 | Disabled | DAC0_OUT0ALT_0 |
| 10 | PC1 | Disabled | DAC0_OUT0ALT_1 |
| 11 | PC2 | Disabled | DAC0_OUT0ALT_2 |
| 12 | PC3 | Disabled | DAC0_OUT0ALT_3 |
| 13 | PC4 | Disabled | DAC0_P0 |
| 14 | PC5 | Disabled | DAC0_N0 |
| 15 | PB7 | Disabled | |
| 16 | PB8 | Disabled | |
| 17 | PA8 | Disabled | |
| 18 | PA9 | Disabled | |
| 19 | PA10 | Disabled | |
| 21 | PB11 | Disabled | DAC0_OUT0 |
| 22 | PB12 | Disabled | DAC0_OUT1 |
| 24 | PB13 | Disabled | |
| 25 | PB14 | Disabled | |

Figure 4.4. Generating a Pin Report

## 5. Mode Transitions

For EFM8 and C8051 8-bit MCUs, modes define different states of peripheral configurations. For example, an application may want to use two configurations of the ADC and Timer 2 with different input pins and sampling rates. These two configurations can be treated as separate modes with separate initialization sequences. The movements between these modes are called transitions. Mode transitions are not supported on the EFM32 products.

By default, each project contains one [**DefaultMode**] mode. The reset state exists only to indicate the transition from [**RESET**] to [**DefaultMode**] and is not a full mode. To add a new mode, right-click on either the [**Mode Transitions**] tab or the [**Outline**] view and select [**Create Mode from Reset**]. Right-clicking on an existing mode also enables cloning that mode with the [**Clone Mode**] option. To add a transition from one mode to another, hover over one of the four white handles on a mode until the mouse cursor changes, hold down left mouse click, and drag to the end mode of the transition. A Peripherals and Port I/O pair of tabs will be added for each mode in the project.

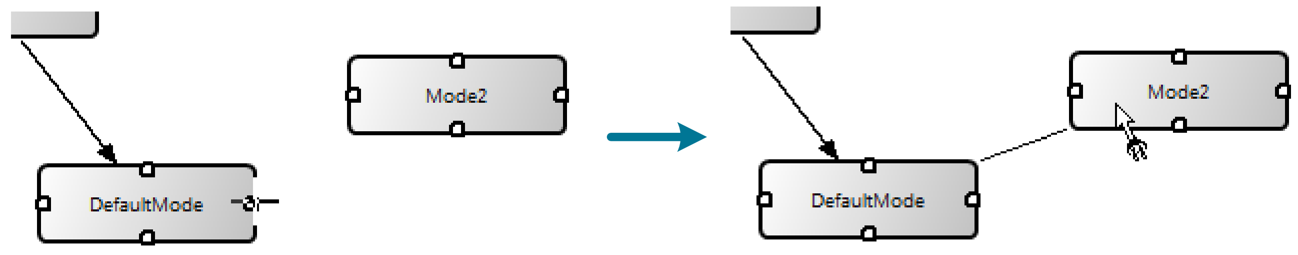


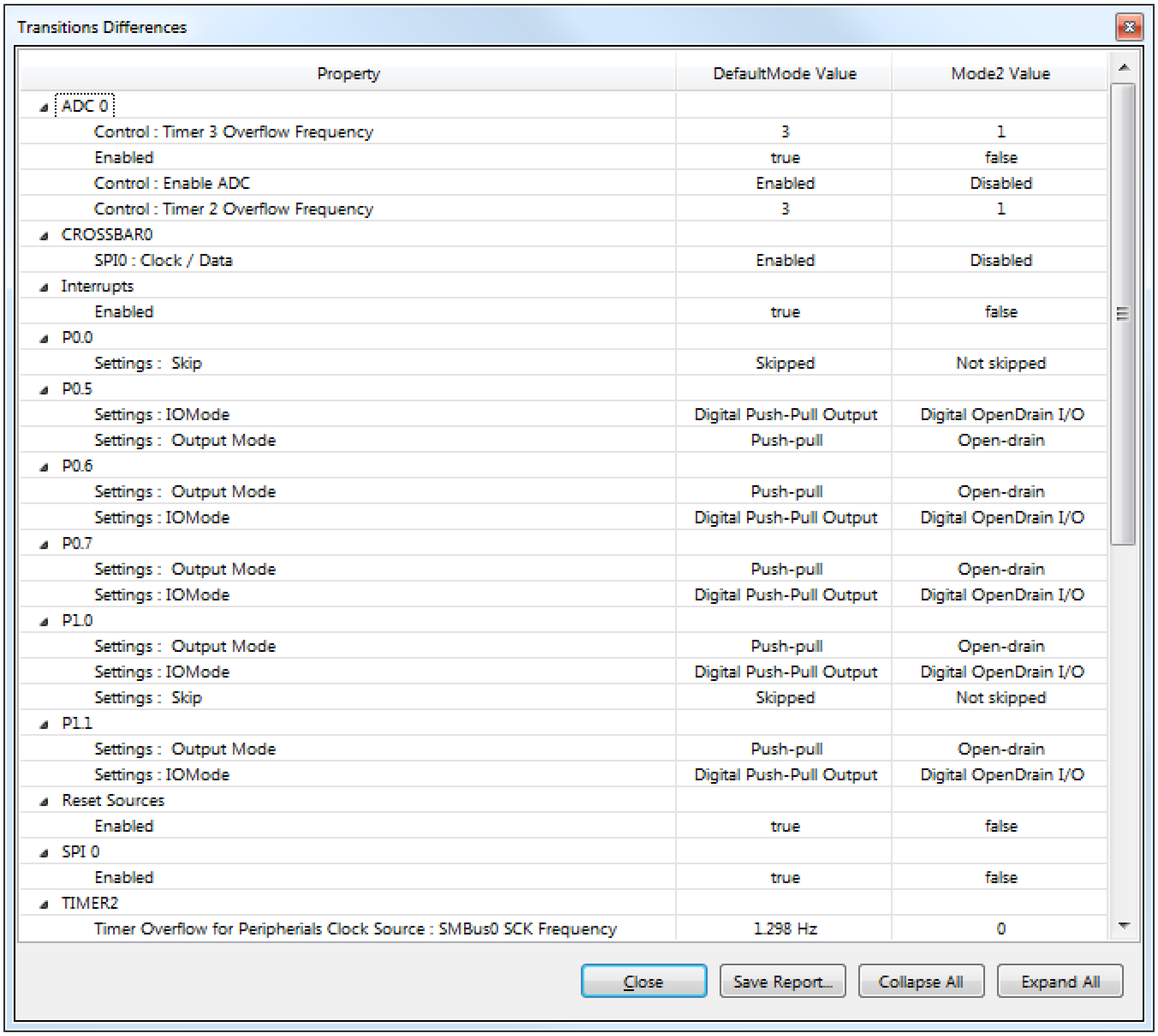
**Figure 5.1. Adding a Mode Transition**

Adding a transition causes the generation of a corresponding function that application firmware can call to initiate the transition between modes. For example, with DefaultMode as the originating mode going to Mode2:

```
enter_Mode2_from_DefaultMode(void)
```

This function then calls all of the functions that configure the peripherals for the mode.

Configurator generates only the transitional code from peripheral to peripheral between modes. For example, if only the timer reload value and ADC input mux selection differ, then those are the only two properties touched by the transition code. To see the differences between two modes, right-click on the mode transition and select [**Show Differences**]. This provides a report of all the different property values between the modes on a module-by-module basis.



| Property | DefaultMode Value | Mode2 Value |
|---|---|---|
| ADC 0 | | |
| Control : Timer 3 Overflow Frequency | 3 | 1 |
| Enabled | true | false |
| Control : Enable ADC | Enabled | Disabled |
| Control : Timer 2 Overflow Frequency | 3 | 1 |
| CROSSBAR0 | | |
| SPI0 : Clock / Data | Enabled | Disabled |
| Interrupts | | |
| Enabled | true | false |
| P0.0 | | |
| Settings : Skip | Skipped | Not skipped |
| P0.5 | | |
| Settings : IOMode | Digital Push-Pull Output | Digital OpenDrain I/O |
| Settings : Output Mode | Push-pull | Open-drain |
| P0.6 | | |
| Settings : Output Mode | Push-pull | Open-drain |
| Settings : IOMode | Digital Push-Pull Output | Digital OpenDrain I/O |
| P0.7 | | |
| Settings : Output Mode | Push-pull | Open-drain |
| Settings : IOMode | Digital Push-Pull Output | Digital OpenDrain I/O |
| P1.0 | | |
| Settings : Output Mode | Push-pull | Open-drain |
| Settings : IOMode | Digital Push-Pull Output | Digital OpenDrain I/O |
| Settings : Skip | Skipped | Not skipped |
| P1.1 | | |
| Settings : Output Mode | Push-pull | Open-drain |
| Settings : IOMode | Digital Push-Pull Output | Digital OpenDrain I/O |
| Reset Sources | | |
| Enabled | true | false |
| SPI 0 | | |
| Enabled | true | false |
| TIMER2 | | |
| Timer Overflow for Peripherials Clock Source : SMBus0 SCK Frequency | 1.298 Hz | 0 |

**Figure 5.2. Transition Differences Report**

Transitions can be added in both directions using the same method as the original mode transition.

To delete a mode transition, select the arrow and press the [**Delete**] key or right-click on the arrow and select [**Delete transition**].

The transition function will not modify the state of the global interrupt enable (EA) on EFM8 or C8051 devices. Transitions assume that interrupts are disabled (i.e., global interrupt enable EA = 0) or that enabled interrupts will not interact with any changes made during the transition. The transition function does not clear interrupt flags to ensure that no information is lost during reconfiguration. Firmware must clear these interrupt flags prior to calling the transition function if having these flags set will cause an issue in the firmware operation.

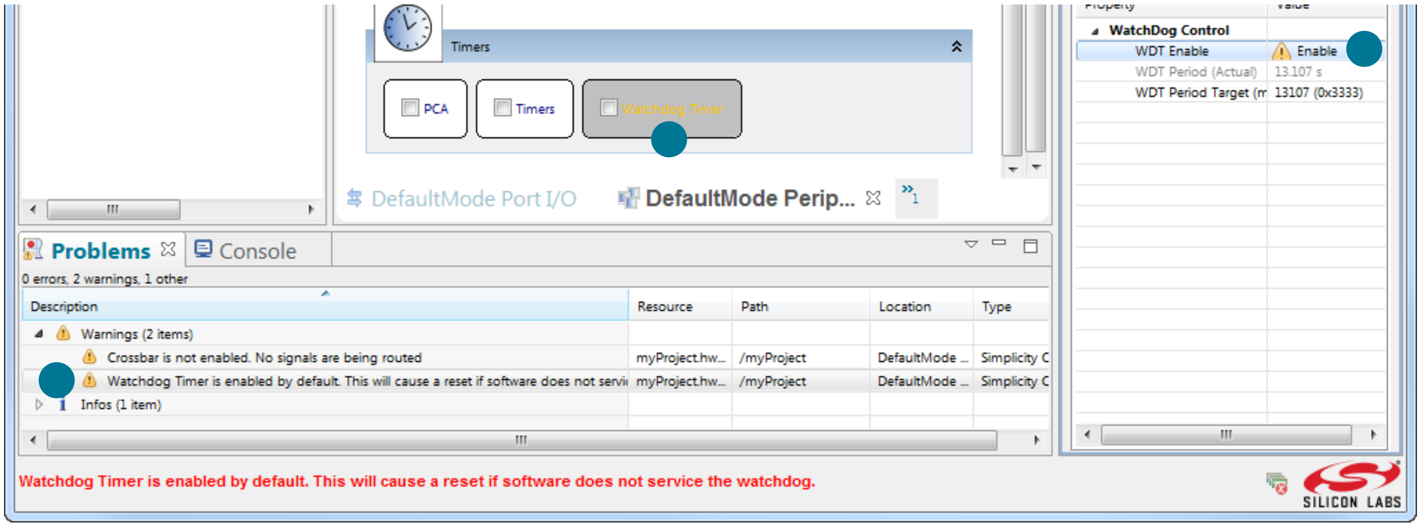
Transitions also assume that any peripherals being updated are not active. For example, firmware stops a timer before calling the transition function and restarts it after the transition completes.

It is possible to change the reconfiguration order of a set of peripherals by creating multiple modes with multiple transitions. For example, to configure Timer 2 before the ADC during a mode transition that would normally put the ADC first, create a mode to handle the Timer 2 changes and a mode to handle the ADC changes. Firmware can then call the Timer2 transition function first.

## 6. Errors and Warnings

Simplicity Configurator automatically validates register configuration values and displays any errors, warnings, or information items in the [**Problems**] view. Double clicking on an entry in the [**Problems**] view will automatically display the property that raised the problem in the [**Properties**] view. Any peripherals that contain a warning or error will be highlighted in yellow (warning) or red (error), and the associated property has a corresponding warning or error icon.

Once the highlighted property is updated, the associated warning or error will disappear from the [**Problems**] view, indicating that the problem has been solved.



0 errors, 2 warnings, 1 other

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|
| Warnings (2 items) | | | | |
| Crossbar is not enabled. No signals are being routed | myProject.hw... | /myProject | DefaultMode ... | Simplicity C |
| Watchdog Timer is enabled by default. This will cause a reset if software does not servi | myProject.hw... | /myProject | DefaultMode ... | Simplicity C |
| Infos (1 item) | | | | |

Watchdog Timer is enabled by default. This will cause a reset if software does not service the watchdog.

Figure 6.1. Errors and Warnings

All newly created projects will have a warning regarding the Watchdog Timer, since it’s enabled after a reset and must be handled in application code or disabled.

If a warning or error doesn’t apply to the project, it can be temporarily deleted from the project by right-clicking on the entry in the [**Problems**] view and selecting [**Delete**]. Closing and reopening the `*.hwconf` file will regenerate any deleted warnings or errors.

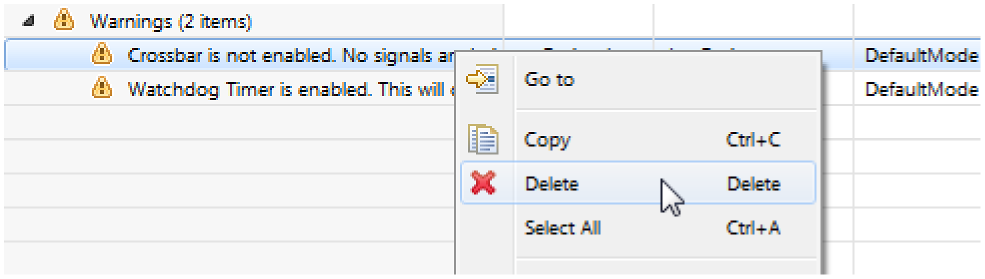


Figure 6.2. Deleting an Error or Warning

# 7. Code Generation

To generate code for a peripheral, enable the checkbox for that peripheral in the [**Peripherals**] tab. Saving the project will automatically generate code for all selected peripherals and port pins. This code generates into the `InitDevice.c`, `InitDevice.h`, and `Interrupts.c` files.

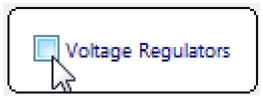


**Figure 7.1. Selecting a Peripheral for Code Generation**

Each hardware register has a section in the `InitDevice.c` file tagged with the register name and register description. For example, the P1 register area is as follows:

```
// $[P1 - Port 1 Pin Latch]
// [P1 - Port 1 Pin Latch]$
```

These tags indicate that everything in between is automatically generated, so any code added manually between these tags will be deleted on the next Configurator project save. Code can be added in between these tags, however, and this code will not be overwritten.

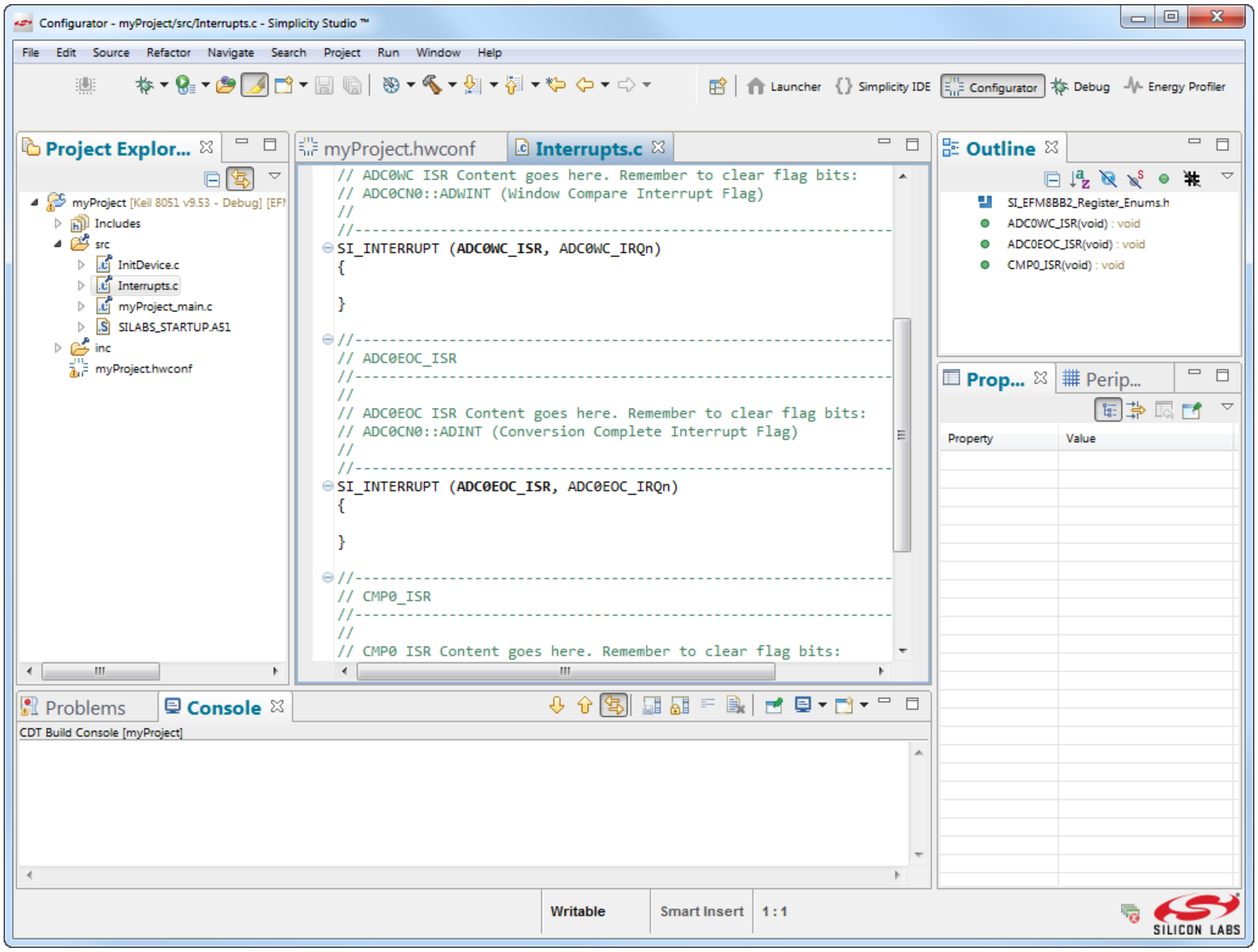
```
// $[P1 - Port 1 Pin Latch]
P1 = 0x00; // This code is not safe, and Configurator will overwrite it
// [P1 - Port 1 Pin Latch]$

P1 = 0x00; // This code is safe from Configurator automatic code generation

// $[P1MASK - Port 1 Mask]
// [P1MASK - Port 1 Mask]$
```

For registers that are modified from their reset value or the value in the previous mode, the tags include comments that update based on the settings for the register. For example, setting P1.0 to push-pull on an EFM8 or 8-bit MCU generates the following:

```
// $[P1MDOUT - Port 1 Output Mode]
/*
// B0 (Port 1 Bit 0 Output Mode) = PUSH_PULL
//    (P1.0 output is push-pull.)
// B1 (Port 1 Bit 1 Output Mode) = OPEN_DRAIN
//    (P1.1 output is open-drain.)
// B2 (Port 1 Bit 2 Output Mode) = OPEN_DRAIN
//    (P1.2 output is open-drain.)
// B3 (Port 1 Bit 3 Output Mode) = OPEN_DRAIN
//    (P1.3 output is open-drain.)
// B4 (Port 1 Bit 4 Output Mode) = OPEN_DRAIN
//    (P1.4 output is open-drain.)
// B5 (Port 1 Bit 5 Output Mode) = OPEN_DRAIN
//    (P1.5 output is open-drain.)
// B6 (Port 1 Bit 6 Output Mode) = OPEN_DRAIN
//    (P1.6 output is open-drain.)
// B7 (Port 1 Bit 7 Output Mode) = OPEN_DRAIN
//    (P1.7 output is open-drain.)
*/
P1MDOUT = P1MDOUT_B0__PUSH_PULL | P1MDOUT_B1__OPEN_DRAIN | P1MDOUT_B2__OPEN_DRAIN
   | P1MDOUT_B3__OPEN_DRAIN | P1MDOUT_B4__OPEN_DRAIN | P1MDOUT_B5__OPEN_DRAIN |
   P1MDOUT_B6__OPEN_DRAIN | P1MDOUT_B7__OPEN_DRAIN;
// [P1MDOUT - Port 1 Output Mode]$
```

## 8. Interrupts

Simplicity Configurator creates an `Interrupts.c` file containing interrupt prototypes whenever interrupts are enabled and the project is saved. Configurator generates the prototype if it's not present, but will otherwise do nothing. This protects any application code written in the prototypes from being accidentally modified or deleted. The comments above the prototypes list any flags in the modules that may need to be cleared by application code after the interrupt triggers.



```
// ADC0WC ISR Content goes here. Remember to clear flag bits:
// ADC0CN0::ADWINT (Window Compare Interrupt Flag)
//
//-----------------------------------------------------------------------
SI_INTERRUPT (ADC0WC_ISR, ADC0WC_IRQn)
{

}

//-----------------------------------------------------------------------
// ADC0EOC_ISR
//-----------------------------------------------------------------------
//
// ADC0EOC ISR Content goes here. Remember to clear flag bits:
// ADC0CN0::ADINT (Conversion Complete Interrupt Flag)
//
//-----------------------------------------------------------------------
SI_INTERRUPT (ADC0EOC_ISR, ADC0EOC_IRQn)
{

}

//-----------------------------------------------------------------------
// CMP0_ISR
//-----------------------------------------------------------------------
//
// CMP0 ISR Content goes here. Remember to clear flag bits:
```

**Figure 8.1. Generating Interrupt Prototypes**

# 9. Revision History

## 9.1 Revision 0.3

June 13th, 2016

Updated screenshots and descriptions for Simplicity Studio v4.

## 9.2 Revision 0.2

February 13th, 2015

Updated formatting.

Updated screenshots for Simplicity Studio v3.

Added EFM32 devices.

## 9.3 Revision 0.1

February 2014

Initial revision.

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

**http://www.silabs.com**