VS Code Extension

Overview

Simplicity Studio Extension for VS Code

Leverage the power of Visual Studio Code while staying fully integrated with Simplicity Studio v6. The Simplicity Studio extension enables seamless build, flash, and debug operations directly within VS Code, while allowing you to continue using familiar Simplicity Studio project configurator GUIs.

Key Features

- Create and configure projects in Studio, then hand off to VS Code for editing, building, flashing, and debugging
- Works with CMake-generated projects and Ninja build tools for efficient compile cycles
- Easily identify and connect to local and remote Silicon Labs 32-bit kits and SoCs from within VS Code
- Install directly from the VS Code Marketplace to effortlessly integrate with your development workflow

Release Notes

Simplicity Studio VS Code Extension 2.0.0 (Jan 22, 2026) Release Notes

Leverage the power of Visual Studio Code while staying fully integrated with Simplicity Studio 6. The Simplicity Studio extension enables seamless build, flash, and debug operations directly within VS Code, while allowing you to continue using familiar Simplicity Studio project configurator graphical user interface.

Release Summary

Key Features

Added in 2.0.0

- The extension now works exclusively with Simplicity Studio 6. Support for Simplicity Studio 5 has been removed.
- Verified with Simplicity Studio 6.1.0 and remains backward compatible with earlier Simplicity Studio 6 versions.
- Improved Simplicity Studio path detection. The extension now automatically synchronizes with the latest Simplicity Studio 6 installation, making upgrades seamless.
- Overhauled the debug launch flow to improve debug session reliability. The extension now uses Simplicity Commander to flash and halt the target device before handing control to J-Link, ensuring a more predictable and stable debug workflow.
- Enhanced debug configuration management so that custom settings in `launch.json` are preserved rather than overwritten by the extension. This allows developers to retain advanced configurations (for example, Live Watch) when starting debug sessions from either the extension or the VS Code Run and Debug view.
- Simplified debug session startup. The extension no longer prompts users to select a binary file when starting a debug session. Instead, it automatically selects a `.rps`, `.hex`, or `.s37` file, in that priority order, based on availability.
- During debugging, the preferred file is used by default (`.rps`, `.hex`, or `.s37`). For EFR32xG917 devices, the `.rps` file is used. To flash a different file, select the file and use the Flash link in the Binaries section.
- Updated the Home and Welcome pages with improved getting-started guidance.
- Verified with Simplicity Studio 6.1.0 and remains backward compatible with earlier Simplicity Studio 6 versions.
- Added a Simplicity Studio launch link to the Tools view for quick access.
- Simplicity SDK header files are now displayed in the project file tree for direct access.

Bug Fixes

Fixed in 2.0.0

- Fixed issue with RTT terminal not launching from a custom board debug adapter.
- Fixed issue in the Build Configurator project for setting Compilers flags to keep Studio from overwriting the custom changes.
- Resolved an issue where connected debug adapters were not detected or listed in the VS Code Devices view.

Removed/Deprecated Features

- Removed support for Simplicity Studio 5 Setting that allowed the user to switch between Simplicity Studio and Simplicity Studio V6 features.

Known Issues and Limitation

UID	Issue or Limitation Descripton		Workaround (if any)	Affected Tools, SDK, Harware
??	Project Configurator Additional Sources menu does not support 'Solution' projects for adding customer source files.	None	Add customer source files in `CMakeLists.txt`.	Series 2 and 3 Devices

UID	Issue Description
1528904	Fixed compiler flag overrides so C optimization settings apply correctly during VS Code builds.
1542834	Resolved an issue where connected debug adapters were not detected or listed in the VS Code Devices view.
1534663	Fixed RTT Terminal launch failures when using custom boards or Mini debug adapters requiring explicit device IDs.
1557571	Fixed workspace import issues on macOS where v6 workspaces failed to load in the VS Code extension.
1563784	Fixed an issue where debugging custom board projects failed or did not stop at the application entry point.
1565358	Fixed an issue where Wi-Fi projects imported from Simplicity Studio 5 could not be debugged in VS Code.
1557531	Improved flash and debug compatibility handling to allow user override for compatible but mismatched parts.

Getting Started

Installation and Configuration

The Simplicity Studio VS Code extension lets developers build, flash, and debug projects in the Visual Studio Code environment while continuing to leverage Simplicity Studio (version 5 or 6) for project configuration and creation.

This hybrid workflow provides the best of both tools: Use Simplicity Studio to set up your project and VS Code for daily development.

> Note: If you have an earlier version of the Simplicity Studio extension installed in VS Code, uninstall it following the steps below before proceeding to ensure a clean installation. Otherwise, continue with the download and installation steps below.

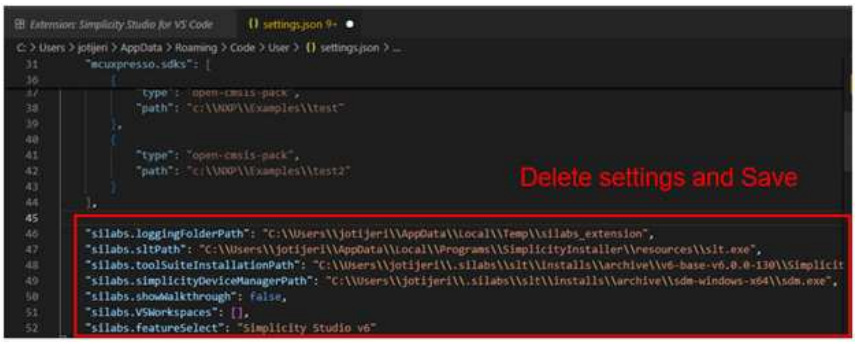
Uninstall the VS Code Extension

In the Extension search bar, type "Simplicity Studio" to locate the extension and click the Uninstall button.

Optional: To ensure there are no leftover settings from a previous installation that could cause issues, you can clear the Silabs settings in the *settings.json* file by following the steps below.

1. Press Ctrl+Shift+P to open the Command Palette, and then type and select Preferences: Open User Settings (JSON).

2. Find and remove the extension settings, and then save the changes.



3. Close and restart VS Code.

Download and Install VS Code

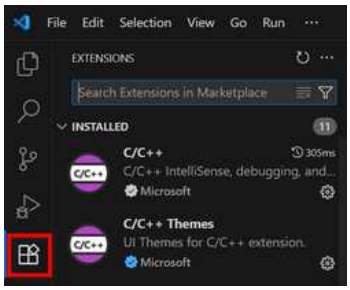
Download the latest version of the VS Code application at https://code.visualstudio.com/download and follow the installation instructions provided there.

Install the Simplicity Studio VS Code Extension

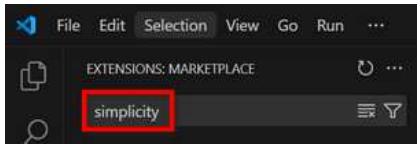
Install the Simplicity Studio VS Code Extension directly from the Extensions view in VS Code following the steps below:

> Important: Simplicity Studio VS Code extension v2.0.0 or later is recommended for compatibility with Simplicity Studio 6.

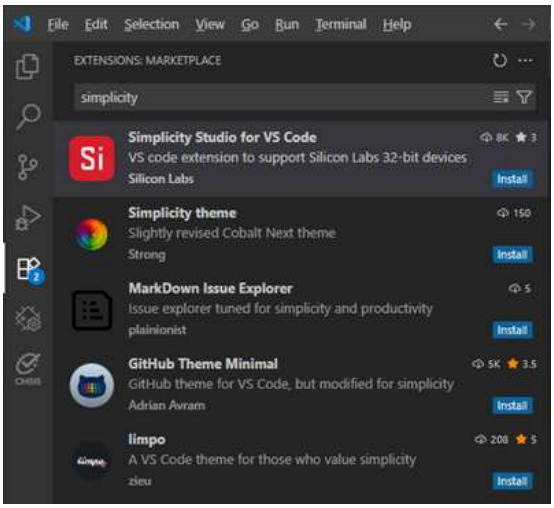
1. Open VS Code.
2. Click the Extensions icon on the left navigation bar. A list of installed extensions displays in the EXTENSIONS panel at the right of the navigation bar.



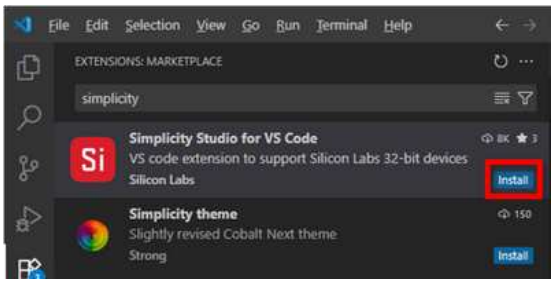
3. In the Search field at the top of the EXTENSIONS panel, type simplicity.



The extensions display in the search results below the panel.



4. At the bottom right of the Simplicity Studio for VS Code extension panel, click the Install button.



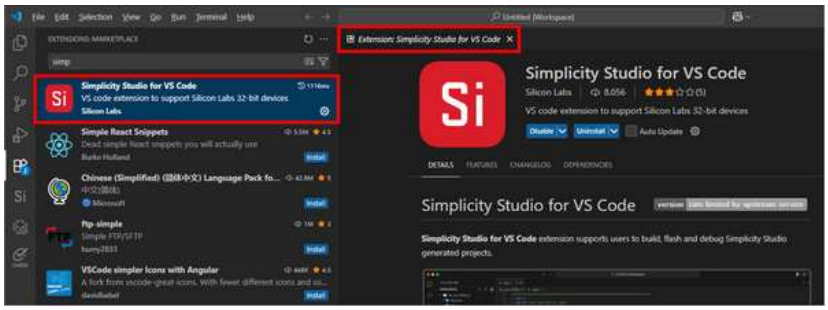
A status bar briefly displays at the top left of your screen while the extension is installed. When the installation is completed, the status bar disappears from the screen.

Verify the Simplicity Studio VS Code Extension Configuration

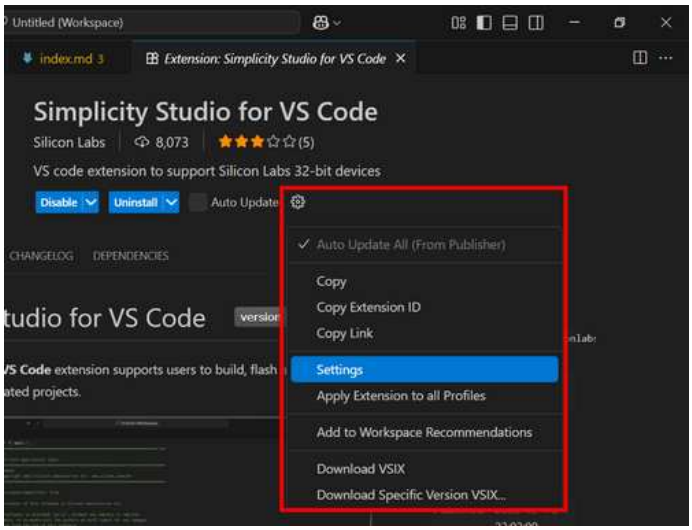
A white circle should appear in the lower-right corner, indicating that the extension is connected to Studio 6 and ready to open projects generated from Simplicity Studio 6. For more information on how to start a project, see Create a Project in the Simplicity Studio v6 User Guide

If you do not see the indicator, or if it appears red, VS Code cannot locate Simplicity Studio. In that case, complete the following steps to verify that the path to the Simplicity Studio tool suite is set correctly, or try closing and restarting VS Code.

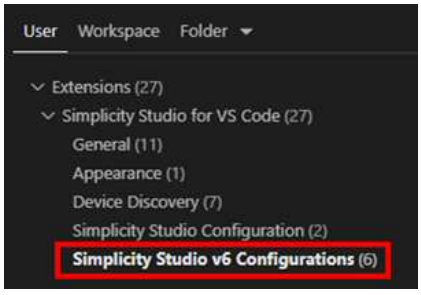
1. In the EXTENSIONS MARKETPLACE panel, click Simplicity Studio for VS Code extension. The extension displays in a tab on the right of your screen.



2. In the extension tab, click the Settings (cog) icon and select Settings from the pop-up menu.



3. Verify the path for the Simplicity Studio Tool Suite is correctly set under Simplicity Studio Configurations. This path should be automatically synchronized with the latest Simplicity Studio 6 installation.



For example, for Simplicity Studio v6, the paths are below:

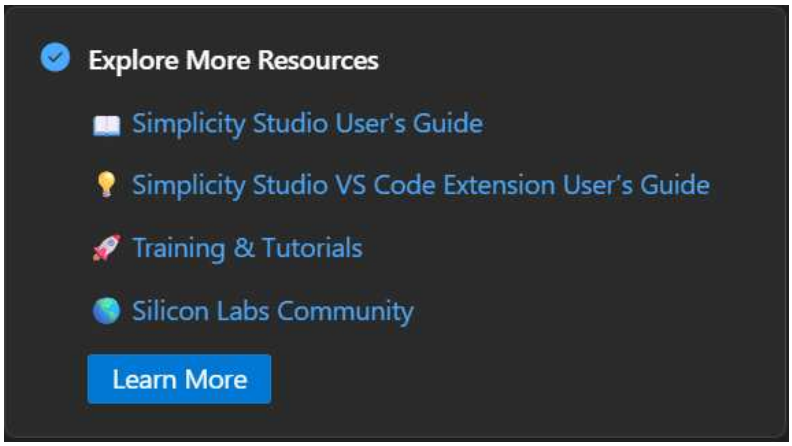
- Windows: `c:\Users\<username>.silabs\slt\installs\archive\v6-base-v6.0.0\SimplicityStudio-6`
- macOS: `/Users/<username>/.silabs/slt/installs/archive/v6-base-v6.0.0/SimplicityStudio-6.app`
- Linux: `/home/<username>/.silabs/slt/installs/archive/v6-base-v6.0.0/SimplicityStudio-6`

Explore the VS Code Extension

Explore the VS Code Extension

After you install the Simplicity Studio for VS Code extension, a Welcome screen displays with getting started instructions for using the extension.

- Download Simplicity Studio: Because the extension relies on projects generated and configured by Simplicity Studio, your first task is to download and install Simplicity Studio 6.
- Generate a VS Code Project: Indicates a project must be generated in Simplicity Studio 6 to use the project in the extension.
- Explore More Resources: Links are provided for the online user's guide, training, and tutorials.



- Ready to Start: Click the Si icon in the left navigation bar to find the heart of the extension functionality.
- Mark Done: Click this link to stop the Welcome screen from displaying when you launch the Simplicity Studio VS Code extension.

Dependencies for the Simplicity Studio for VS Code Extension

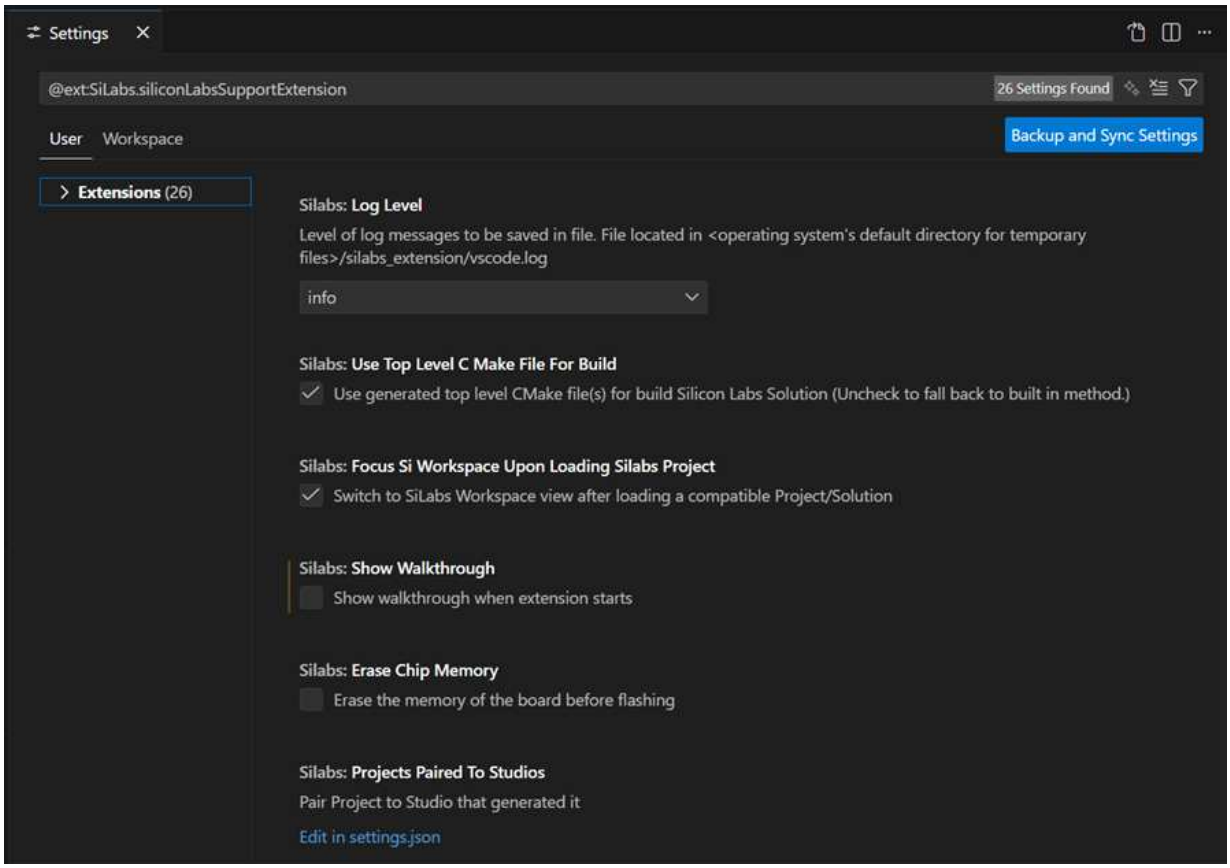
You can view the dependencies for the Simplicity Studio VS Code extension by selecting the extension in VS Code and then clicking the DEPENDENCIES tab. The current dependencies are:

- Cortex-Debug
- C/C++
- C/C++ Themes
- CMake
- YAML

Settings for the Simplicity Studio for VS Code Extension

You can explore the extension settings by:

1. Selecting the extension.
2. Clicking the Settings (gear) icon and selecting Settings from the pop-up menu.
   The Settings tab displays.



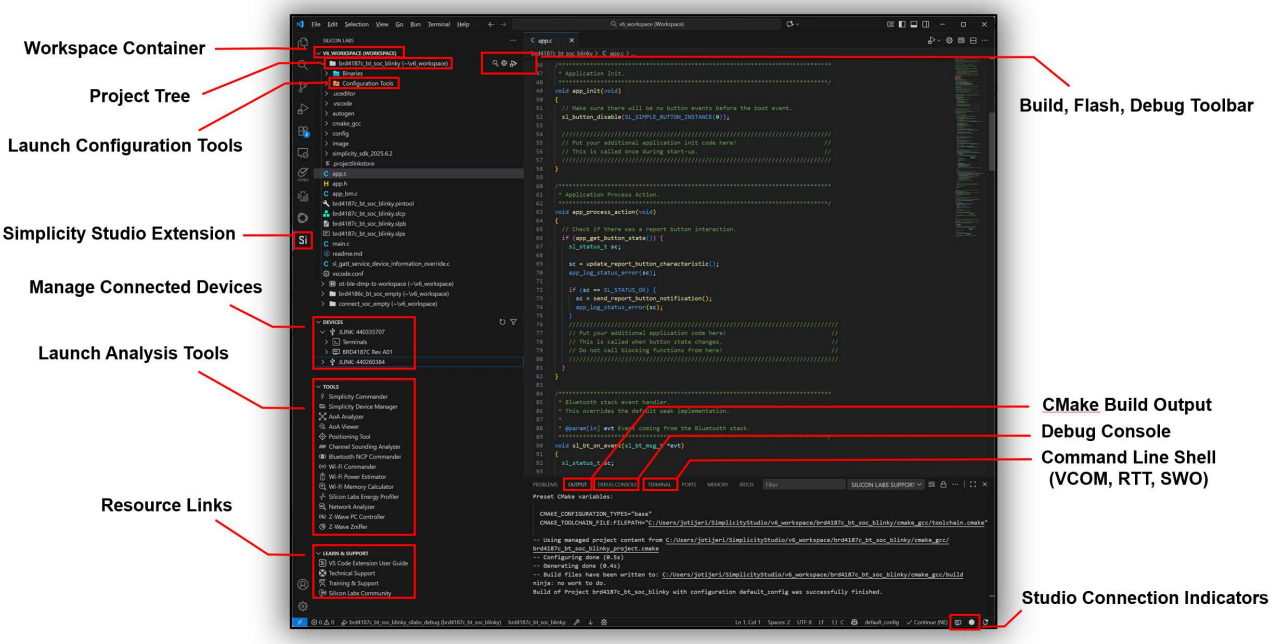
Each setting has a description on the Settings tab, but only a few important settings are listed below:

- Build Before Flash: This setting controls whether a project is built before flashing the target. If you always manually build the project to check for errors before flashing, you can deselect this setting.
- Device Name: A custom board setting that lets you override the target part. This is an important setting if the other device detection and configuration options are not working.

Si View

The Si View has four possible containers:

- Workspace
- Devices
- Tools
- Learn and Support



Workspace Container

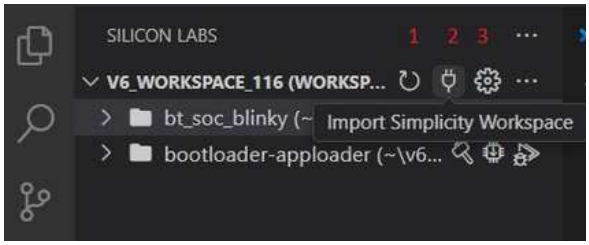
When you select Si view for the first time, no items have been added to the Workspace container, and so links display to either Open Simplicity Studio or Import Workspace. If you have not created any projects in Simplicity Studio, click Open Simplicity Studio to create one or more projects and.or solutions.

> Note: In this context, the term, *project*, refers to projects and solutions, except when discussing explicit solution functionality.

After you create one or more projects, click Import Workspace to add the project(s) to the Si Workspace container. Any recently opened workspaces are displayed in the dropdown list. If you do not see them, click Search to find your Simplicity Studio projects or workspaces. A Simplicity Studio generated project is considered compatible, if its generator list contains Visual Studio Code. This is the default for all Simplicity Studio 6 projects.

Workspace Toolbar

The Workspace toolbar has three icons:



1. Refresh Workspace: Click to refresh your workspace when you have made changes to the workspace outside VS Code.
2. Import Workspace: Click to import a workspace as described above.
3. Settings: Click to display the settings for the Simplcity Studio VS Code extension.

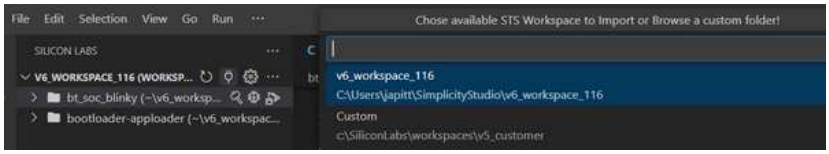
Connect and Select Previous Workspaces

Follow these steps to connect and select previous workspaces:

1. Import the Simplicity Studio Workspace.
2. Select a previously connected workspace.
   - Selecting a previously connected workspace automatically loads the folder content into the workspace view.
   - Edit the list using settings.

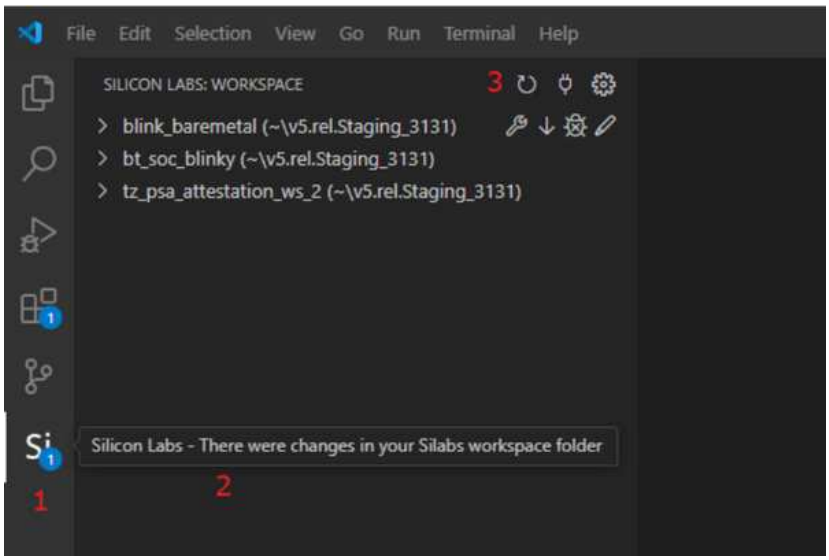
> Important: Only projects and solution generated with *Visual Studio Code* generator are displayed.

3. Click the Browse button to browse the folders for a new workspace.
   - Selecting a new workspace automatically loads the folder content into the workspace view.



Workspace Details

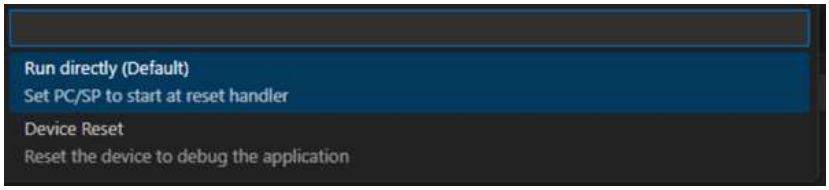
1. Si icon: The number in the circle at the lower right corner of this icon shows how many notifications are waiting. These are listed in the tooltip when you hover your pointer over the Si icon.
2. Tooltip: This displays when the workspace has changes.
3. Refresh icon: Click to update the content in your workspace.



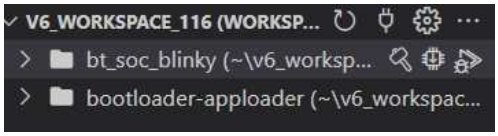
Actions for a Displayed Project

The following toolbar icons are available for a displayed project:

- Build: If multiple build configurations are present for a project, you must select one build configuration before you build a project.
- Flash:
  - Before executing a flash, you must choose a binary file to flash. Select from .hex, .bin, or .rsp. For binary files that do not contain address information, such as .bin, you should also add a starting memory address.
  - When multiple compatible devices are connected (see settings: *Board display*), choose one.
- Debug:
  - When multiple compatible devices are connected (see settings: *Board display*), choose one.
  - When the debug section starts successfully, the Run and Debug view displays.
  - When starting a debug session for the first time, you'll be prompted to choose one of two startup modes: Run Directly (default) or Device Reset.
    - Run Directly: Sarts code execution from the application's reset handler, initializing the stack pointer (SP) and program counter (PC) directly from the application's vector table. This mode does not perform a full hardware reset and is recommended for applications with a non-zero boot address, such as those running alongside a bootloader.
    - Device Reset - Performs a full hardware reset before starting the debug session, ensuring the device begins execution from its primary reset vector. This mode is appropriate for standalone applications or when a clean hardware state is required.



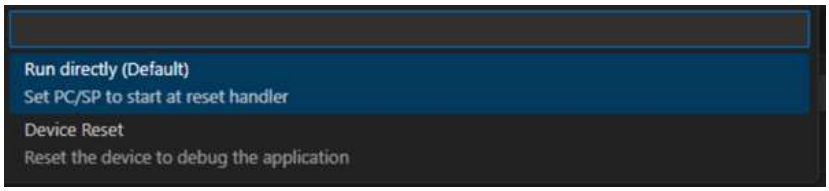
  - When Debug is launched, a `launch.json` file is created or updated in the `.vscode` folder with information about the Silicon Labs debug session. If you have your own `launch.json` file and want to keep it, the corresponding Overwrite launch file setting should be turned off.



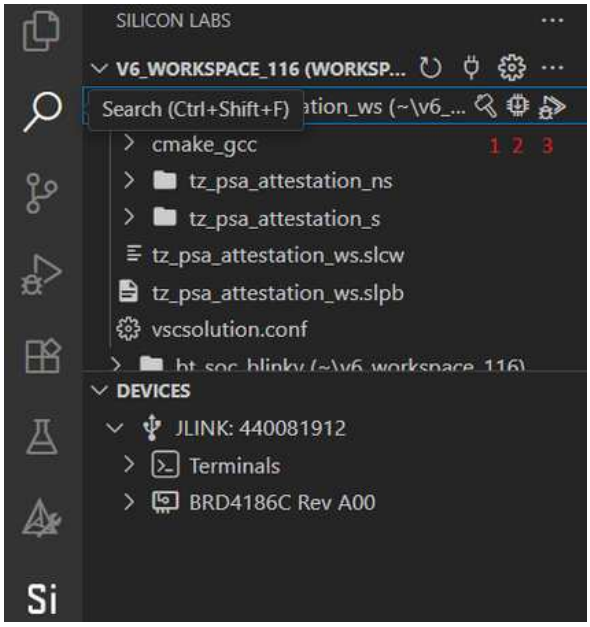
Actions for a Displayed Solution

The following actions are available for a displayed solution:

- Build: In this version only, one build configuration is available for a solution.
- Flash:
  - Before executing a flash, you must choose a binary file to flash.
  - When multiple compatible devices are connected (see settings: *Board display*), choose one.
- Debug:
  - When multiple compatible devices are connected (see settings: *Board display*), choose one.
    - When starting a debug session for the first time, you'll be prompted to choose one of two startup modes:Run Directly (default) or Device Reset.
      - Run Directly: Sarts code execution from the application's reset handler, initializing the stack pointer (SP) and program counter (PC) directly from the application's vector table. This mode does not perform a full hardware reset and is recommended for applications with a non-zero boot address, such as those running alongside a bootloader.
      - Device Reset - Performs a full hardware reset before starting the debug session, ensuring the device begins execution from its primary reset vector. This mode is appropriate for standalone applications or when a clean hardware state is required.

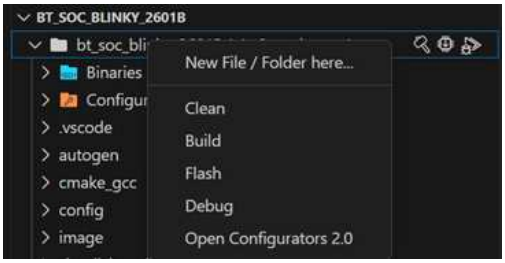


- When the debug section starts successfully, the Run and Debug view displays.
- When Debug is launched, a `launch.json` file is created or updated in the `.vscode` folder with information about the Silicon Labs debug session. If you have your own `launch.json` file and want to keep it, the corresponding Overwrite launch file setting should be turned off.



Context Menu for a Project Folder

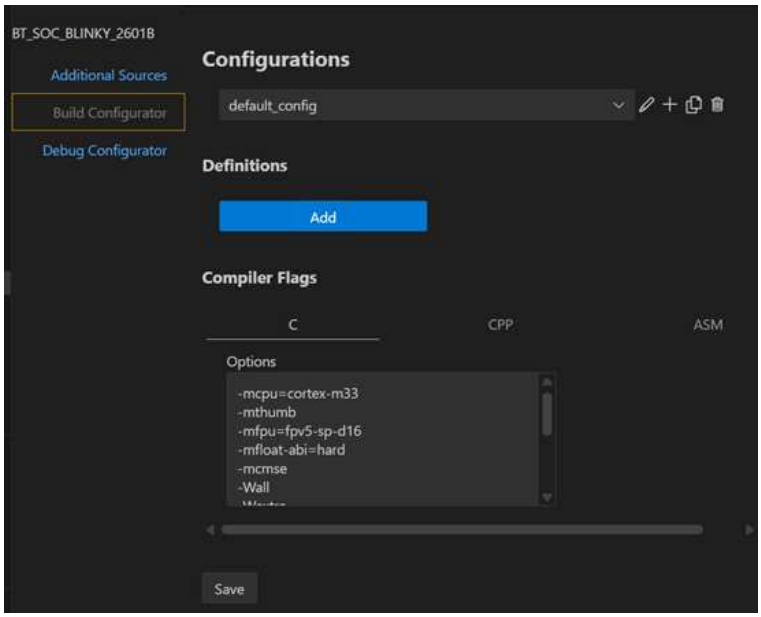
Right-click a project folder to display a context menu with the following actions:



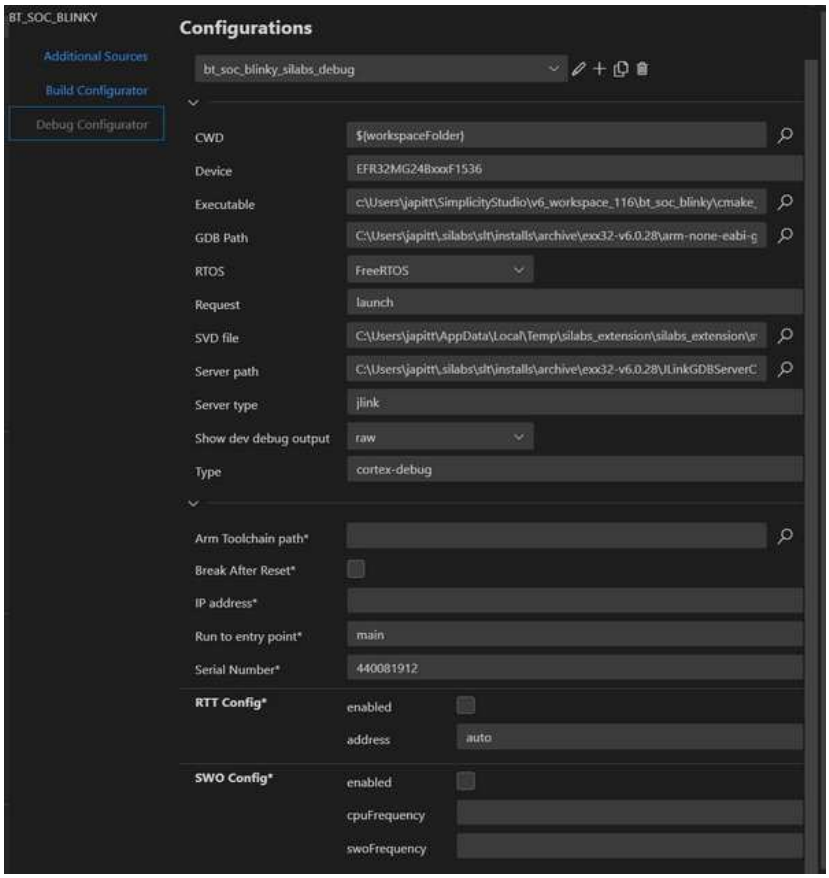
1. Add new file or folder to the selected folder
   - When you select this action, an additional input dialog displays.
   - If the text input contains a file extension, a file is created; otherwise, a folder is added.
   - You cannot add a folder or file to the top-level folder of a solution.
2. Clean: Does a clean build of the project.
3. Build: Builds the project.
4. Flash: Flashes the project binary to the target.
5. Debug: Launches the debugger for the project.

- Additonal Sources: Adds files and folders previously added to the project to the build configuration.

  - Folder Name: Adds a source file folder to the build configuration.
  - File Path: Adds include file paths to the build configuration. The added path is searched for header files included by source files.
- Build Configurator: Adjusts the build configurations.



  - Configurations: Selects an existing build configuration to edit or rename, add, copy, or delete a configuration.
  - Definition (build symbols)
    - Edit or delete an existing definitions.
      - Add new definitions.
  - Compiler Flags: Add, edit or delete compiler flags for C, CPP, or ASM.
  - Save your changes.
- Debug Configurator: Adjusts the debugger configuration.
  The debug configuration is created automatically with default settings for the project. The Debug Configurator lets you change the settings manually.
  - Select from the available debug configurations to edit.
  - Rename, add, copy, or delete a debug configuration.



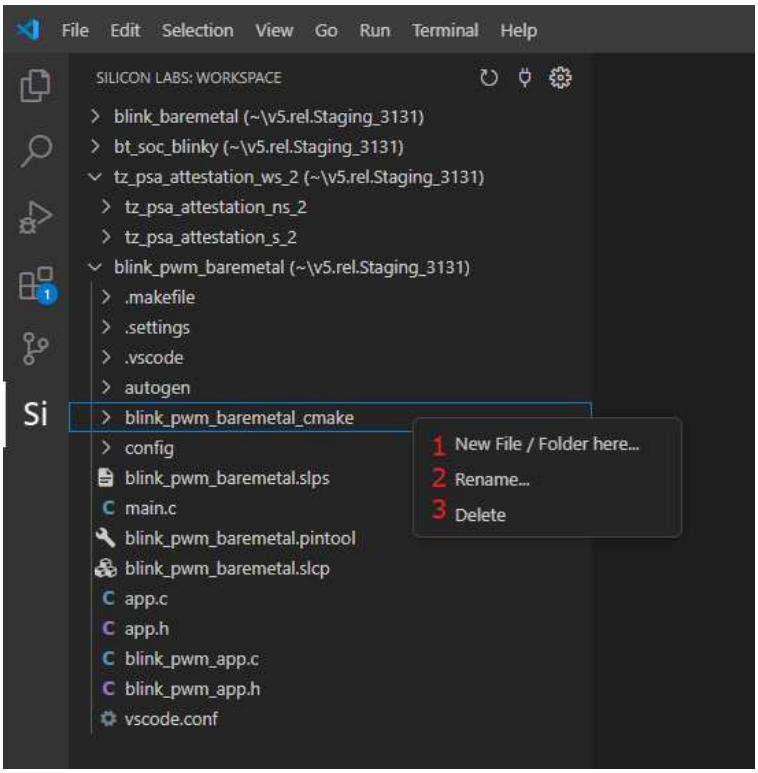
- Debug Configurator Fields
  - CWD: The current working directory or workspace folder.
  - Device: The debug device name, it may be different than the actual device OPN depending on currently supported J-Link support. The default setting should be compatible with J-Link.
  - Executable: The debugger-compatible project binary file [i.e., a binary file with executable and linking format (ELF) debug information].
  - GDB Path: The path to the arm gdb client.
  - RTOS: The real-time operating system (RTOS) used by the project, if any. This loads the GDB RTOS plugin for the debug session.
  - Request: Launch
  - SVD File: The special function register (sfr) file for the debugger debug device (target). This file should be downloaded automatically by the extension based on the project properties.
  - Server path: The path to the GDB server.
  - Server type: J-Link, the only server type currently supported.
  - Show dev debug output: The format to use in the output in the Packet Trace Interface (PTI) debug console.
  - Type: The debug configuration type. Currently `cortex-debug` is the only supported debug configuration type.
  - Arm Toolchain path: Use to override the default toolchain path.
  - Break After Reset: If Run to entry point is blank, select this option to stop the debugger after a device is reset.
  - IP address: The IP address for the networked J-Link debug adapters.
  - Run to entry point: The initial breakpoint for the debug session. The default is run to main.
  - Serial Number: The serial number for the USB-connected debug adapters. This is when multiple debug adapters are connected to the computer.

7. Save your debug configurator changes.

Context Menu Within a Project

When you are in a project, right-click a folder to display a context menu with the following actions:

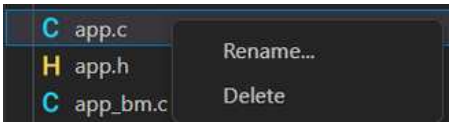
1. New File / Folder here: Adds a new file or folder to the selected folder.
    - When you choose this, an additional input dialog displays.
    - If a text input contains a file extension, a file is created; otherwise, a folder is added.
    - You cannot add a folder or file to the top-level folder of a solution.
2. Rename: Changes the name of a selected folder.
    - You can rename projects and solutions in Simplicity Studio only.
    - You can rename sub-projects of a solution in Simplicity Studio only.
    - Renaming a folder generated in Simplicity Studio may cause the project to malfunction.
3. Delete: Deletes a selected folder.
    - You can delete projects and solutions in Simplicity Studio only.
    - You can delete sub-projects of a solution in Simplicity Studio only.
    - Deleting a folder generated in Simplicity Studio may cause the project to malfunction.



Context Menu for a Source File

Right-click a source file to open a context menu with the following actions:

1. Rename: Changes the name of the selected file.
   Renaming a file generated in Simplicity Studio may cause the project to malfunction.
2. Delete: Deletes the selected file.
   Deleting a file generated in Simplicity Studio may cause the project to malfunction.



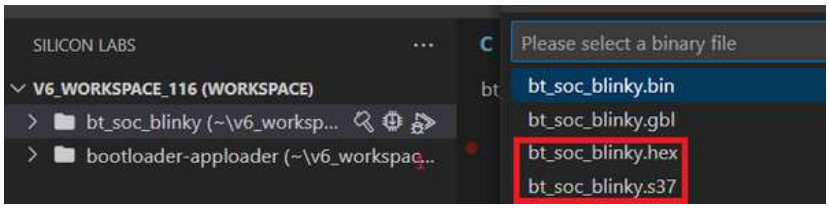
File-specific Context Menu

Right-click a Simplicity Studio 6 tool or project file to open a file-specific context menu that may include opening it in Simplicity Studio 6 or in a text editor.pintool file.

Show the selected file in VSC Explorer by moving the focus to the selected file in the VSC Explorer view.

Successfully built configuration under the project

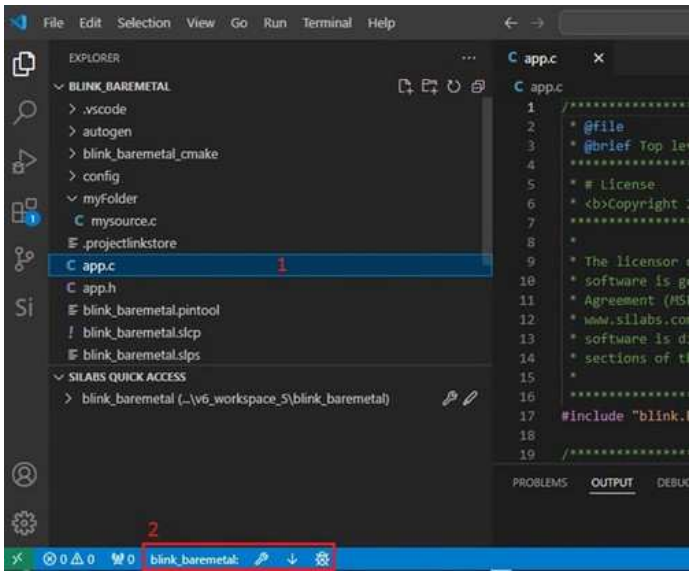
1. From the project folder toolbar, click the Flash icon.
   - Before executing a flash, you must choose a binary file to flash. Select from .hex, .bin, or .rsp. For binary files that do not contain address information, such as .bin, you should also add a starting memory address.
   - When multiple compatible devices are connected (see settings: *Board display*), choose one.
2. From the project folder toolbar, click the Debug icon.
   - When multiple compatible devices are connected (see settings: *Board display*), choose one.
   - When the debug section starts successfully, the Run and Debug view displays.
   - When Debug is launched, a `launch.json` file is created or updated in the `.vscode` folder with information about the Silicon Labs debug session. If you have your own `launch.json` file and want to keep it, the corresponding Overwrite launch file setting should be turned off.



Status Bar

When you select a file from an opened project in VS Code explorer, the shortcuts to build, flash, and debug display in the status bar with the project name that contains the file you selected.

1. A file selected in the EXPLORER panel of VS Code
2. Shortcuts to build, flash and debug



You can initiate these by one or more additional input steps, depending on the number of opened projects and the number of connected devices.

Projects Guide

Projects Guide

The Simplicity Studio VS Code extension supports users to build, flash, and debug Simplicity Studio v6 generated projects.

This section provides details on adding a project to VS Code and various project configurations.

Topics include creating and importing projects, building and debugging, terminal support, and custom board setup.

Create a New Project

Create a New Project

Start by creating a project in Simplicity Studio.

1. Follow the steps in the Studio v6 Project Generation guide for VS Code IDE
2. When prompted, target the project for VS Code.

After creation, the project will auto-load in VS Code under the Simplicity Workspace (Si) extension view.

Import/Open an Existing Project

Import/Open an Existing Project

You can add a project to VS Code in three ways.

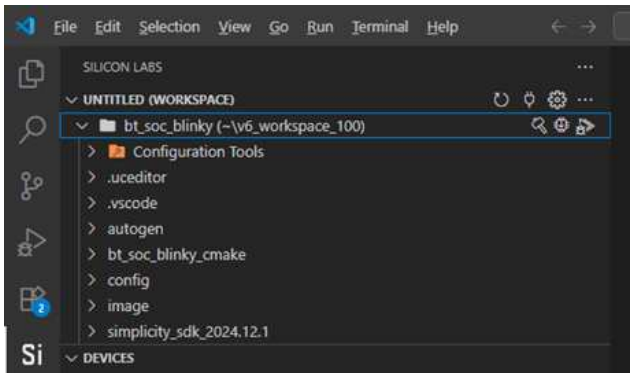
1. Use File /> Open Folder…. Browse to the Simplicity Studio project folder.
2. Use File /> Add Folder to Workspace…. Recommended for managing multiple projects. Then save the workspace using File > Save Workspace As… to a non-Studio folder.
3. Use Silabs Workspace /> Connect to Simplicity Workspace.

Build, Flash, or Debug a Project

Build, Flash, or Debug a Project

In the Si extension view, hover over your project to reveal the following action buttons:

- Build: Compiles your project
- Flash: Programs your target device
- Debug: Launches the debugger



Attach to a Running Target

The Simplicity Studio VS Code extension does not currently provide a built-in option to attach to a running target. However, this is supported through the Cortex-Debug extension by adding an attach configuration to the project's `launch.json` file.

This procedure assumes you have debugged the project at least once so that `.vscode/launch.json` exists.

Attach the Configuration

1. In the Si Activity bar, open your project's `.vscode/launch.json` file.
2. In the `configuration[]` array, select and copy the existing launch configuration (typically from the opening bracket (`{`) on line 3 through the closing bracket (`}`) for that block).

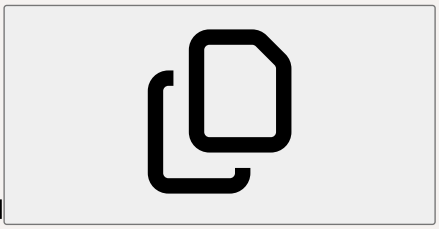
launch.json

bt_soc_blinky_freertos_2 > .vscode > launch.json > JSON Language Features > [ ] configurations

```
{
    "configurations": [
        {
            "name": "bt_soc_blinky_freertos_2_silabs_debug:default_config",
            "cwd": "${workspaceFolder}",
            "type": "cortex-debug",
            "request": "launch",
            "serverpath": "C:\\Users\\japitt\\.silabs\\slt\\installs\\archive\\exx32-v6.0
            "servertype": "jlink",
            "gdbPath": "C:\\Users\\japitt\\.silabs\\slt\\installs\\archive\\exx32-v6.0.32
            "device": "EFR32MG24BxxxF1536",
            "svdFile": "C:\\Users\\japitt\\AppData\\Local\\Temp\\silabs_extension\\silabs
            "runToEntryPoint": "main",
            "rtos": "FreeRTOS",
            "showDevDebugOutput": "raw",
            "executable": "c:/Users/japitt/SimplicityStudio/v6_workspace/bt_soc_blinky_fr
            "preLaunchTask": "bt_soc_blinky_freertos_2_silabs_debug:default_config_combin
            "breakAfterReset": true,
            "serialNumber": "440081918",
            "overrideLaunchCommands": [
                "exec-file c:/Users/japitt/SimplicityStudio/v6_workspace/bt_soc_blinky_fr
                "symbol-file c:/Users/japitt/SimplicityStudio/v6_workspace/bt_soc_blinky_
            ],
            "overrideResetCommands": [
                "monitor reset",
                "set $pc = __Vectors[1]",
                "set $sp = __Vectors[0]"
            ]
        },
```

3. Paste the copy after the existing configuration. Use it as the starting point for your attach configuration.
4. In the copied configuration block, change `debug` in the name to `attach`. You can use any name, but it should show that this entry is for an attach configuration.
5. Change `request` from `launch` to `attach`.
6. Delete the `runtoEntryPoint` line.
7. Delete the `preLaunchTask` line.
8. Change `breakAfterReset` from `true` to `false`.
9. If they are present, delete the `overrideLaunchCommands` and `overrideResetCommands` sections.
10. By default, attach halts the processor so you can inspect the target state and set breakpoints. To attach without halting, add a `postAttachCommands` entry:

```
"postAttachCommands": ["continue"]
```



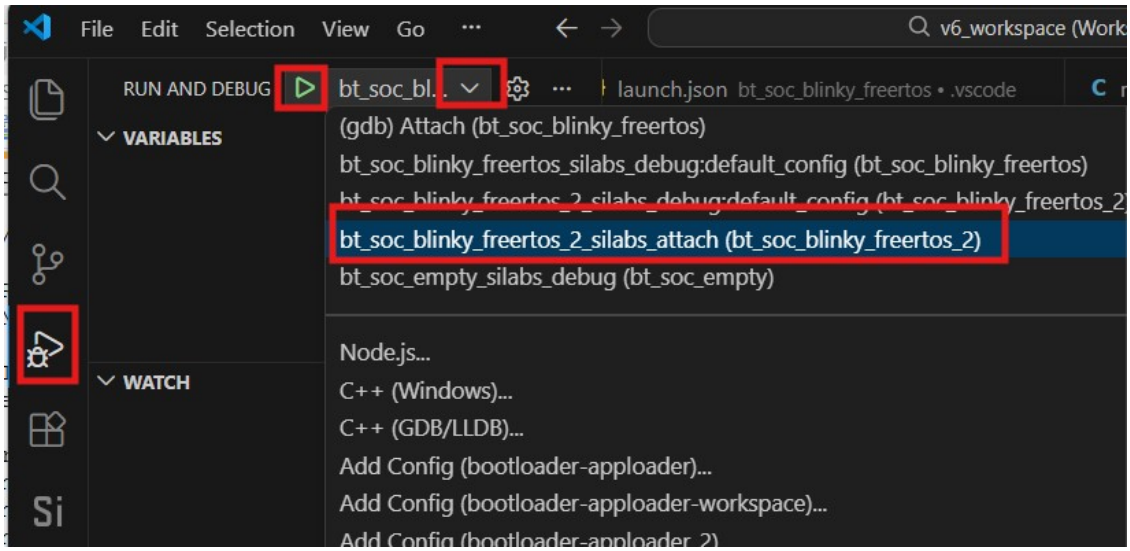
> Note: If you use `continue`, the processor does not stop at attach. The Pause button does not work until you set a breakpoint and hit it. After that, Pause works when you resume.

11. Save the file.

Use an Attach Configuration

From the Silicon Labs Activity Bar (Si), select the debug icon. A drop-down lists every configuration in `launch.json`, including your attach configuration.

Do not use the Si extension toolbar debug icon to start an attach session. Instead, open the Run and Debug view from the activity bar and select your attach configuration from the dropdown.

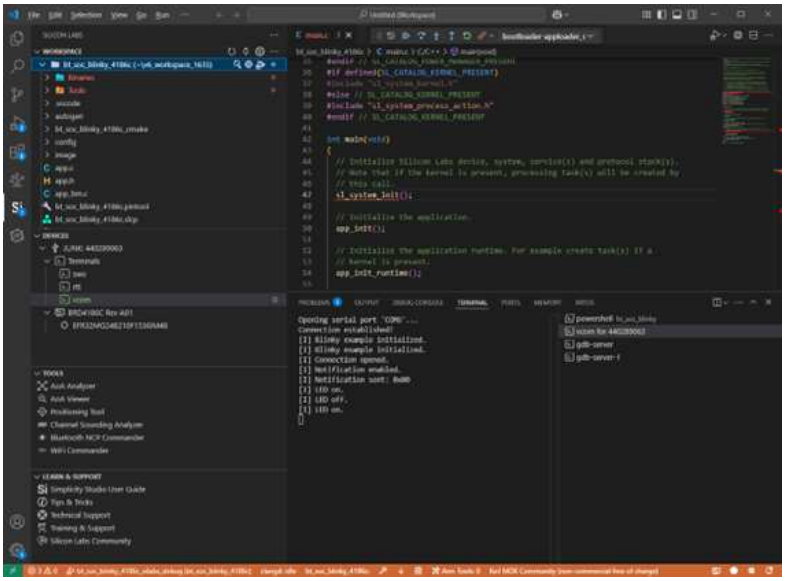


Select Start Debugging (F5) (green arrow) to attach to the running target.

Terminal Support

Terminal Support

The debugger supports multiple device interfaces, including SWO, RTT, and VCOM for terminal.



Set Up Custom Board

Set Up Custom Boards

There are two recommended workflows for developing with custom boards:

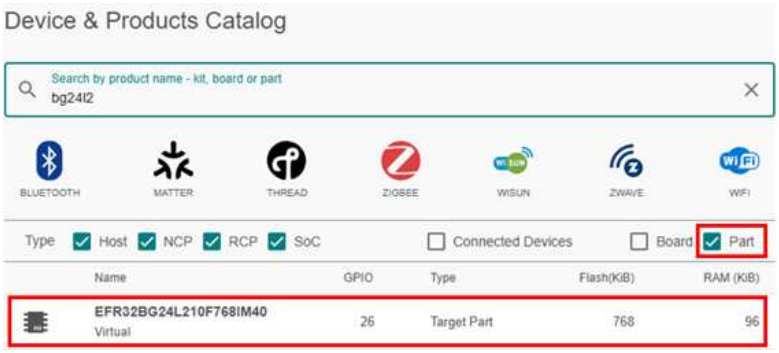
- Start with a device-compatible "Empty" sample application project.
- Start with a sample application project for a Silicon Labs development board, and retarget it to the device on your custom board.

When developing a project for a custom board, you must update settings in Simplicity Studio 6 Project Configurator and Simplicity Device Manager (SDM). In some cases, you may also need to adjust the debug configuration in the Visual Studio Code extension to ensure correct debug sessions.

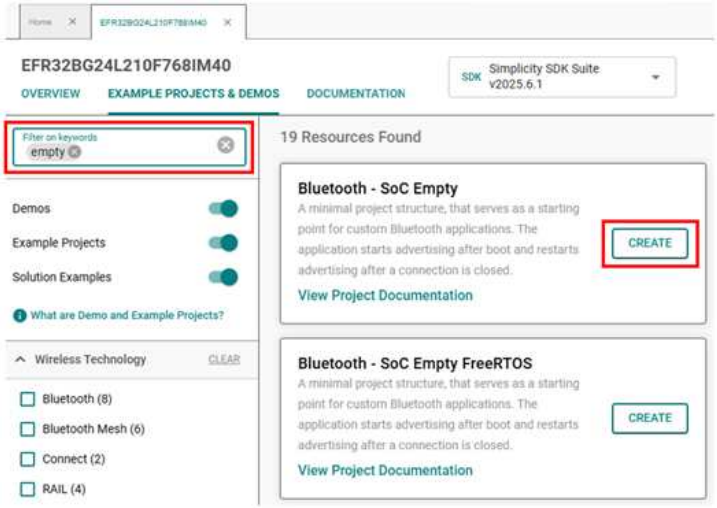
The following sections describe the required changes for each workflow.

Device-Compatible Example for a Custom Board

To create a project for a custom board, start by creating a project for the target device on your board.



Use one of the board-independent "Empty" SiSDK sample applications, such as Bluetooth – SoC Empty or Empty C Project.



This approach configures your project with the correct memory map, clock settings, peripheral set, and device constraints from the start. You can begin application development immediately, and the "Empty" sample application project should build without errors.

From this baseline, add your own configurations to support your custom hardware and application requirements.

Project Retargeting for a Custom Board

To create a project for a custom board using Project Retargeting, start by creating an example project for a Silicon Labs development board (such as a Radio Board or Explorer Kit) that uses a device similar to the target part on your custom board. You can then retarget the project to your custom board.

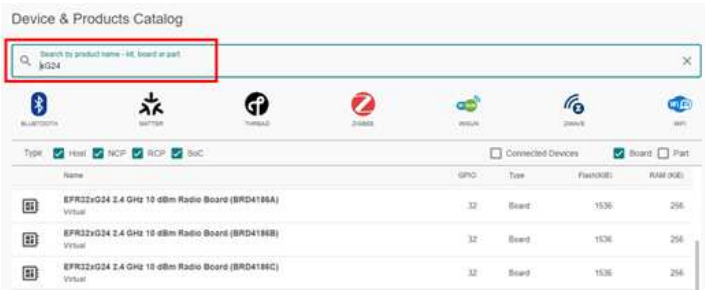
1. Add a Virtual Device

You don't need to purchase a physical board—Silicon Labs boards can be added as virtual devices. These boards typically use superset target devices to maximize feature coverage, including additional pins, increased memory, enhanced peripherals, and expanded system capabilities.

For custom boards using devices from with the same device family, the retargeting workflow supports transitions between superset and subset devices.

To find a compatible board:

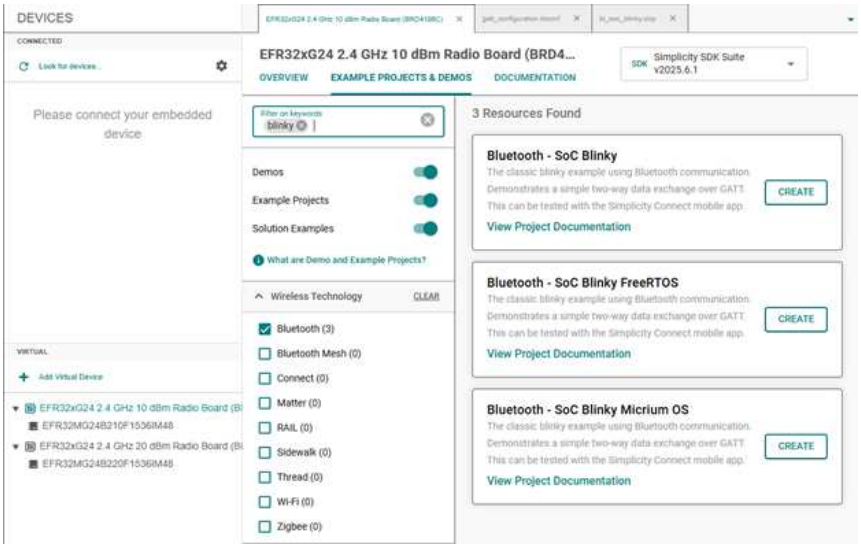
1. Go to Devices and click + Add Virtual Device.
2. Ensure both Board and Part checkboxes are selected.
3. In the Search by product name box, begin typing the target device number.
    - If no boards match, delete the last character to broaden the search.
    - If still no matches appear, try selecting a more general device family rather than one tied to a specific wireless protocol.



2. Select an Example Project

Once the virtual device is added:

1. Open the Example Projects & Demos tab to view examples projects for that board.
2. If few examples are listed, try selecting a different board.
3. Use filters to narrow down the list.
4. Click Create for a project that closely matches the application you are developing or uses similar peripherals to your project.
5. Review the project documentation to understand its functionality before proceeding.



3. Retarget the Project

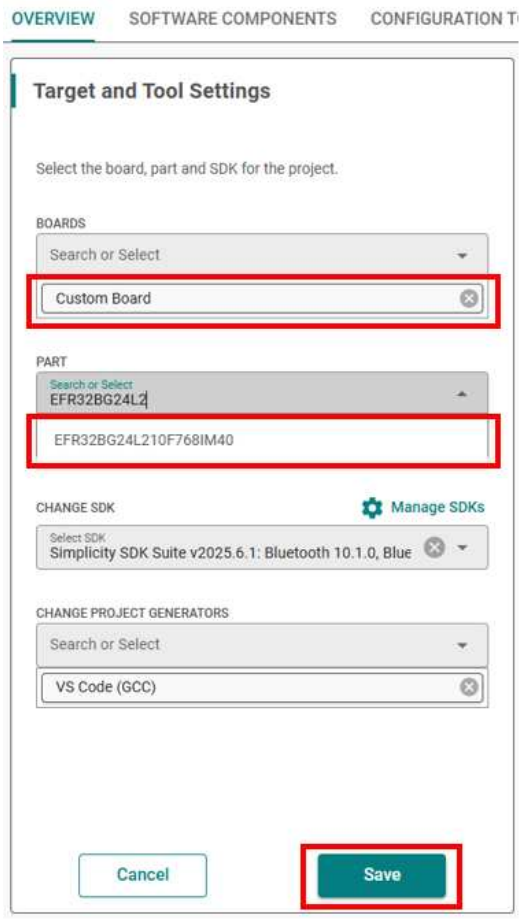
After creating the project, ensure it builds successfully in Visual Studio Code.

You have flexibility on when to retarget the project to the actual device; however, retargeting early helps minimize rework, especially if the target part lacks resources (such as pins or peripherals) as the part on the original board.

Retargeting requires removing the board and selecting the actual target part from the project configurator (.slcp file). This process involves two separate save operations: one after removing the board, and another after setting the desired target part.

To retarget:

1. Open the Overview tab and locate the Target and Tool Settings card.
2. Select Change Target/SDK/Generators.
3. Enter `custom board` in the boards Search or Select field.
4. In the Part Search or Select field, start entering the desired target part and select it from the drop-down list. Remove any listed boards by clicking the small “x” next to each.
5. Click Save.



The project will regenerate for the selected part.

Retargeting from a Silicon Labs board example carries over board configurations, software components, and drivers (such as LEDs, buttons, HFXO, LFXO, RF front-end, and sensors) from the original board. This eliminates the need to manually create a Board Support Package (BSP) using the Pintool—provided the custom board's pin configuration matches the Silicon Labs board.

If the pin configuration differs (e.g., an LED is assigned to a different pin), you must update the configuration using the Pintool based on your custom board's schematics.

4. Resolve Misconfigurations

After retargeting, address any configuration issues:

- Pin differences: Update pin definitions based on your custom board's schematic.
- Hardware features: When retargeting to a target device with the same hardware features (e.g., peripherals, power modes, and memory), the configuration from the original example is retained. However, when retargeting to a device with fewer hardware features, resulting in the loss of a peripheral, remove or replace unsupported components to prevent build failures.
- Memory density: The project configurator modify and define the Linker file to reflect changes in memory.

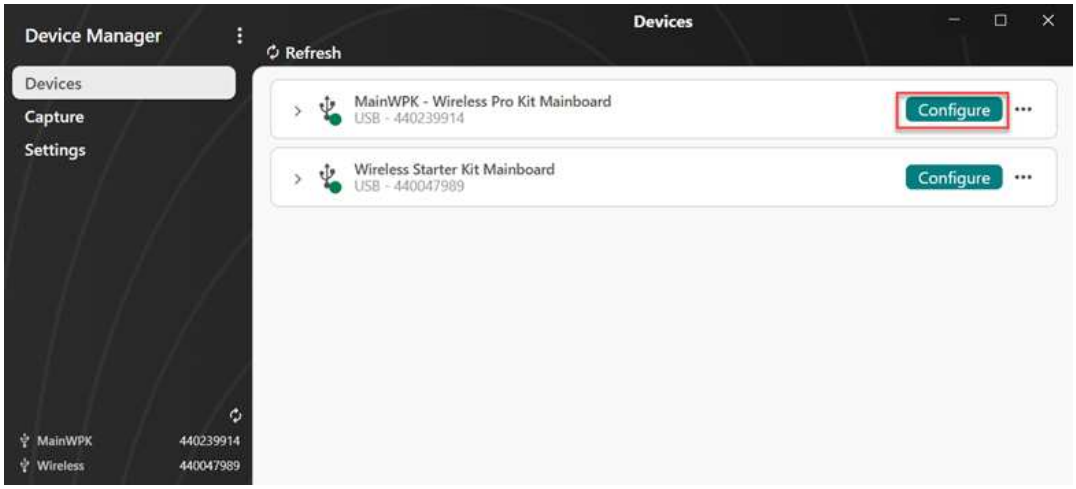
If your custom board's hardware matches that of the Silicon Labs board, system behavior and performance should remain consistent.

To verify configuration, generate a Project Configuration Report. This report summarizes installed board components and helps verify the project setup.

Simplicity Device Manager (SDM)

Simplicity Device Manager (SDM) is the central access point for debug adapters across all Silicon Labs software tools. It provides the current debug adapter configuration to any tool that requires access, making it the recommended location for configuring debug adapters for use with customer board designs.

You can launch SDM from the TOOLS menu in either Studio 6 or the VS Code extension. In SDM, select Devices, then click the Configure button for the desired debug adapter.



Debug Adapters

The following Silicon Labs debug adapters are supported for programming and debugging custom boards and hardware:

- Wireless Pro Kit Mainboard (WPK) – *Si-MB4002A*
- Wireless Starter Kit Mainboard (WSTK)
- Simplicity Link Debugger – *Si-DBG1015A*

Details about each debug adapter are shown in SDM, along with options for various operations.

- When using a Wireless Starter Kit Mainboard (WSTK BRD4001A) with a custom board, set the Adapter Debug Mode to OUT using the dropdown menu.
- When using a Wireless Pro Kit Mainboard (WPK BRD4002A, OPN: Si-MB4002A) with a custom board, set the Adapter Debug Mode to either:
  - MINI – to use the built-in mini-Simplicity connector located on the corner of the board.
  - OUT – to use the debug connectors on the side of the board.

In OUT mode for either WSTK or WPK, the standard method is to insert the Simplicity Debug Adapter Board (OPN: SLSDA001A) into the Debug Connector and Simplicity Connector, then use a 10-pin ribbon cable to connect to the debug connector on the custom board. See AN958 for more details and additional debug connector options.

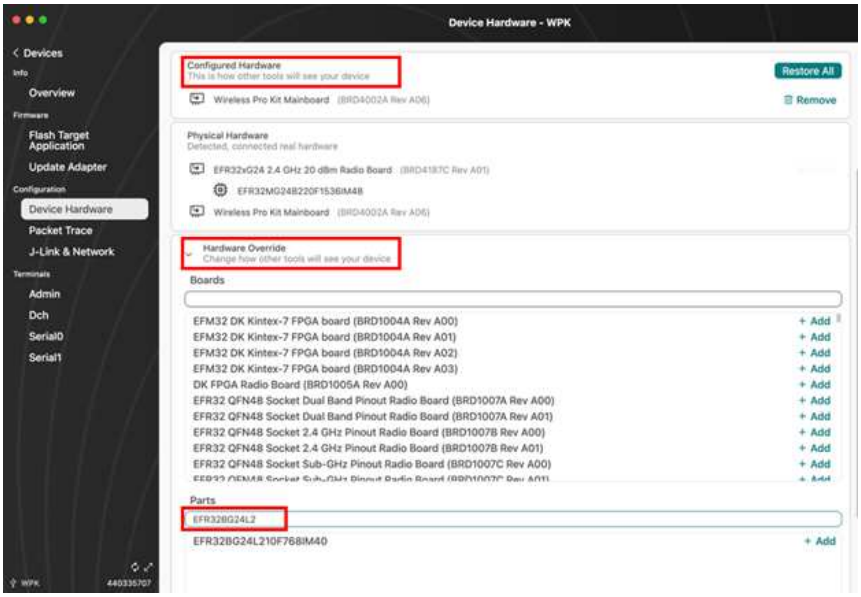
When using the Simplicity Link Debugger (Si-DBG1015A), set the Adapter Debug Mode to OUT, then connect your custom board either via the mini-Simplicity connector or through the header pins using jumper wires.

Custom Board Setup

In the Configuration section:

- The Device Hardware area displays the hardware that has been automatically detected.

- The Configured Hardware section shows the debug adapter configuration that is shared with all other software tools.

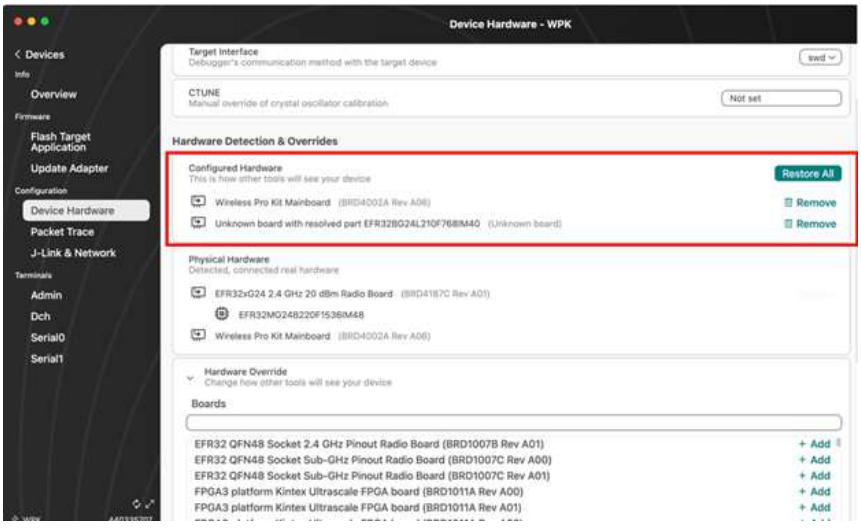


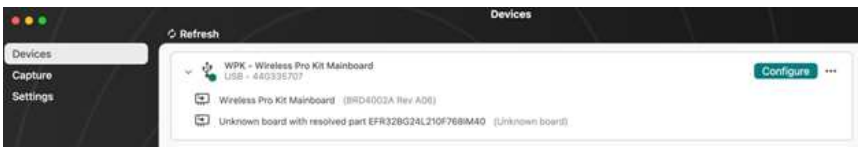
When using a debug adapter with a custom board, set the target part in the Hardware Override section:

1. Start typing the part number in the search box.
2. When the correct part appears, click Add to include it in the configured hardware.

This ensures the debug adapter is correctly configured for all tools.

If the target part changes in the future, click Restore All to reset the debug adapter to the physical hardware configuration. You can then apply new overrides as needed.



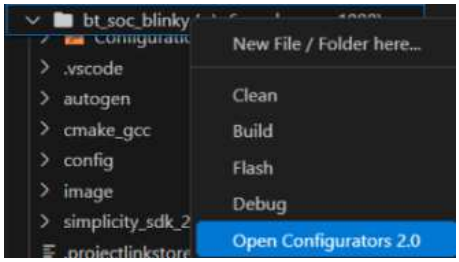


VS Code Extension Debug Configuration

The VS Code extension automatically retrieves the debug adapter configuration from SDM, so no additional setup is typically required to flash and debug a custom board. The debug adapter configured for the custom board will appear under Devices in the Simplicity VS Code extension.

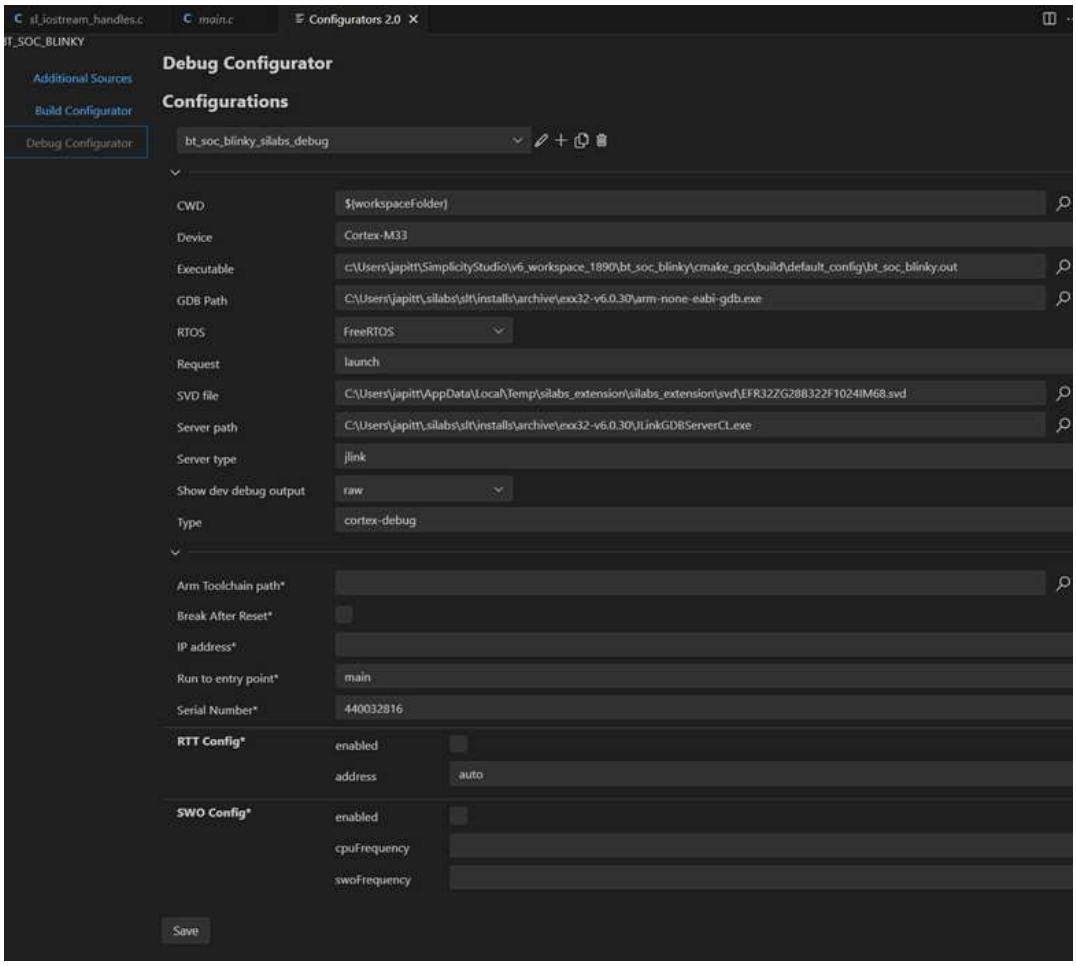
If the default settings are not sufficient, use the project-specific Debug Configurator:

1. Right-click the project folder.
2. Select Open Configurators 2.0.



3. Choose Debug Configurator.

The standard debug configuration fields will be pre-filled. Expand the bottom section of the dialog to view optional fields for the debug session, including configuration options for RTT and SWO terminals.



Common Settings

- Break after Reset – Stops code execution after a reset instead of running to `main`.

- Run to entry point – Defaults to `main`, but can be changed for special debugging scenarios.

Enable the RTT Config and/or SWO Config sections to apply custom settings for those terminals. The default settings are typically sufficient.

Project Configuration and Management

Project Configuration and Management

Create a GNU Arm Static Library Project

Simplicity Studio 6 does not provide a built-in option for generating static libraries. However, for projects using the `arm-none-eabi-gcc` GNU Arm Embedded Toolchain, you can modify the generated `CMakeLists.txt` to produce a `.a` static library instead of an executable.

1. Open the `CMakeLists.txt` file.
2. Change the `add_executable()` section to an `add_library()` section with the `STATIC` keyword:
   Before change:

```
add_executable(blink_pwm_baremetal
add_executable(blink_pwm_baremetal
  # Add additional sources here
)
```

   After change:

```
add_library(blink_pwm_baremetal STATIC
add_library(blink_pwm_baremetal STATIC
  # Add additional sources here
)
```

3. Comment out or remove the sections that copy the executable into other formats and run post-build commands.
   Commented-out sections:

```
# Create .bin, .hex and .s37 artifacts after building the project
#add_custom_command(TARGET blink_pwm_baremetal
   # POST_BUILD
   # COMMAND ${CMAKE_OBJCOPY} ${OBJCOPY_SREC_CMD}"$<TARGET_FILE:blink_pwm_baremetal>"
"$<TARGET_FILE_DIR:blink_pwm_baremetal>/$<TARGET_FILE_BASE_NAME:blink_pwm_baremetal>.s37"
   # COMMAND ${CMAKE_OBJCOPY} ${OBJCOPY_IHEX_CMD} "$<TARGET_FILE:blink_pwm_baremetal>"
"$<TARGET_FILE_DIR:blink_pwm_baremetal>/$<TARGET_FILE_BASE_NAME:blink_pwm_baremetal>.hex"
   # COMMAND ${CMAKE_OBJCOPY} ${OBJCOPY_BIN_CMD}  "$<TARGET_FILE:blink_pwm_baremetal>"
"$<TARGET_FILE_DIR:blink_pwm_baremetal>/$<TARGET_FILE_BASE_NAME:blink_pwm_baremetal>.bin"
#)
# Run post-build pipeline to perform additional post-processing
#if(post_build_command)
#add_custom_command(TARGET blink_pwm_baremetal
   # POST_BUILD
   # WORKING_DIRECTORY ${CMAKE_CURRENT_LIST_DIR}/..
   # COMMAND ${post_build_command}
#)
#endif()
```

4. Save the file.

You can now build the project to create the static library object of the form `libPROJECTNAME.a`.