Developers Guide

Developers Guide

Machine Learning Developers Guide

The Developer's Guide content is organized as follows:

- Add Machine Learning to a New or Existing Project: Describes the process of adding a machine learning model to a new or an existing project.
- Update or Replace a .tflite File: Describes steps to update and replace a machine learning model in your project.
- Developing a Model: Explains the procedure to develop a machine learning model.
- AI/ML Extension Setup: In-depth explanantion of adding AI/ML extension to the setup.
- Flatbuffer Converter Tool: Shows how a flatbuffer is converted to bytes for including into header file.
- I2S Configuration for SiWx917: Provides details on configuring I2S for AI/ML audio apps on SiWx917.

Add Machine Learning to a New or Existing Project

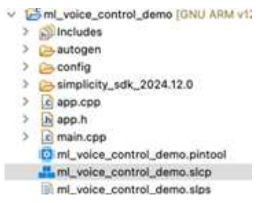
Add Machine Learning to a New or Existing Project

This guide provides details of adding Machine Learning to a new or existing project, making use of the wrapper APIs for TensorFlow Lite for Microcontrollers provided by Silicon Labs for automatic initialization of the TFLM framework.

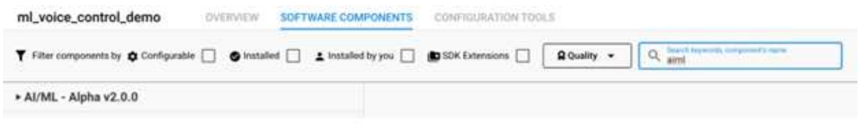
The guide assumes that a project already exists in the Simplicity Studio workspace and you have installed the AI/ML extension. If you're starting from scratch, you may start with any sample application or the Empty C++ application. Please refer to AI/ML Extension Setup for instructions to install the AI/ML extension. TFLM has a C++ API, so the application code interfacing with it will also need to be written in C++. If you're starting with an application that is predominantly C code, see the section on interfacing with C code for tips on how to structure your project by adding a separate C++ file for the TFLM interface.

Install the TensorFlow Lite Micro Component

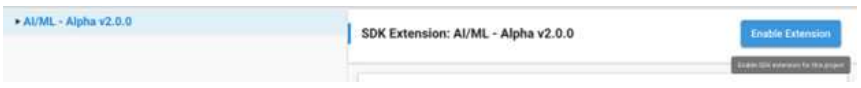
1. Open your project file (the one with the `.slcp` extension).


ml_voice_control_demo [GNU ARM v1
Includes
autogen
config
simplicity_sdk_2024.12.0
app.cpp
app.h
main.cpp
ml_voice_control_demo.pintool
ml_voice_control_demo.slcp
ml_voice_control_demo.slps

2. Under Software Components, search for "aiml".

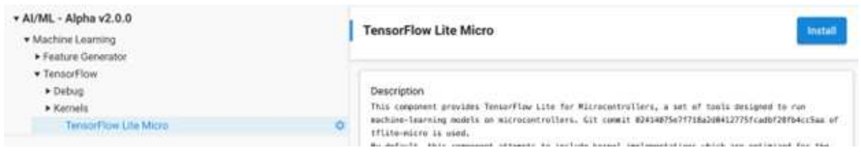


3. Enable the AI/ML extension by clicking Enable Extension.



4. Expand: AI/ML > Machine Learning > TensorFlow. Select TensorFlow Lite Micro and click Install.

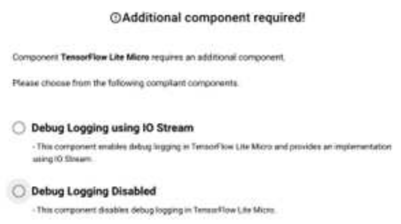
> Note: Skip this step for SiWx917 applications.



This component provides TensorFlow Lite for Microcontrollers, a set of tools designed to run machine-learning models on microcontrollers. Git commit 02414075e7f718a2d0412775fcadbf20fb4cc5aa of tflite-micro is used.

5. You will be prompted to select additional components:

> Note: Skip this step for SiWx917 applications.



Additional component required!

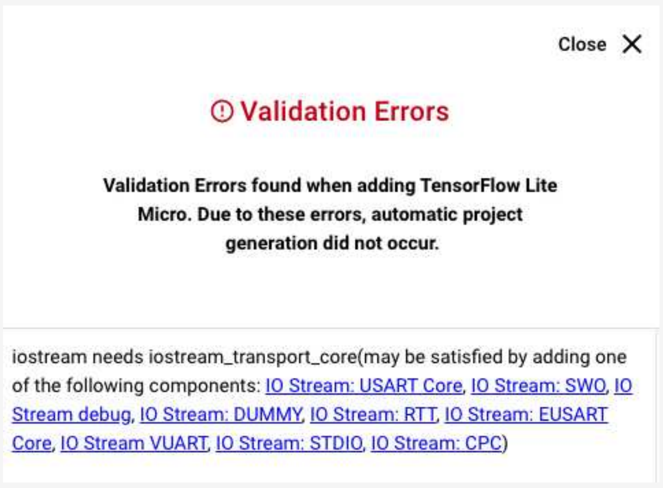
Component **TensorFlow Lite Micro** requires an additional component.

Please choose from the following compliant components.

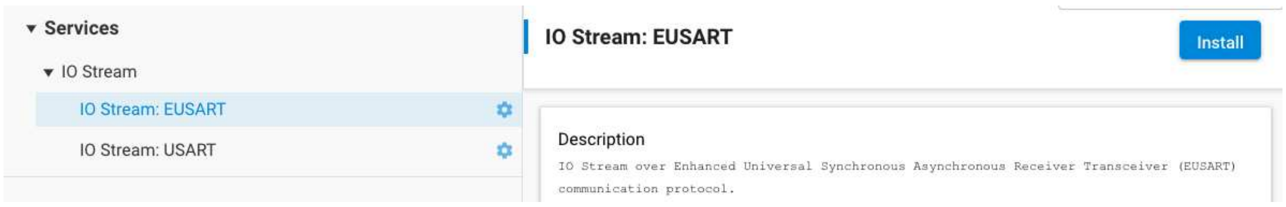
( ) **Debug Logging using IO Stream**
- This component enables debug logging in TensorFlow Lite Micro and provides an implementation using IO Stream.

( ) **Debug Logging Disabled**
- This component disables debug logging in TensorFlow Lite Micro.

- Debug Logging: Choose Debug Logging using IO Stream (if needed) or Debug Logging Disabled. Click Install.

  > NOTE: If your project didn't already contain an I/O Stream implementation, you may get a dependency validation warning. This is not a problem, but simply means that a choice of I/O Stream backend needs to be made. The USART or EUSART backends are the most common, as these can communicate with a connected PC through the development kit virtual COM port (VCOM).
  >
  > 
  >
  >
  > **Validation Errors**
  >
  > **Validation Errors found when adding TensorFlow Lite Micro. Due to these errors, automatic project generation did not occur.**
  >
  > iostream needs iostream_transport_core(may be satisfied by adding one of the following components: IO Stream: USART Core, IO Stream: SWO, IO Stream debug, IO Stream: DUMMY, IO Stream: RTT, IO Stream: EUSART Core, IO Stream VUART, IO Stream: STDIO, IO Stream: CPC)

- IO Stream: You may also need to install EUSART or USART component if you don't see output on your serial console.

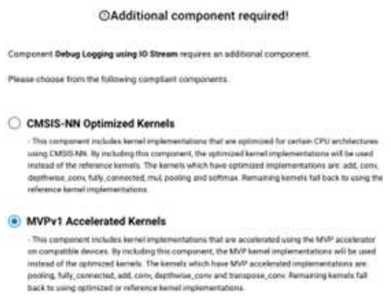


Description

IO Stream over Enhanced Universal Synchronous Asynchronous Receiver Transceiver (EUSART) communication protocol.

Accept the default suggestion of "vcom" as the instance name, which will automatically configure the pinout to connect to the development board's VCOM lines. If you're using your own hardware, you can set any instance name and configure the pinout manually.

- Kernels: Select MVPv1 Accelerated Kernels. Click Install.



Additional component required!

Component **Debug Logging using IO Stream** requires an additional component.

Please choose from the following compliant components.

( ) **CMSIS-NN Optimized Kernels**

- This component includes kernel implementations that are optimized for certain CPU architectures using CMSIS-NN. By including this component, the optimized kernel implementations will be used instead of the reference kernels. The kernels which have optimized implementations are: add, conv, depthwise_conv, fully_connected, mul, pooling and softmax. Remaining kernels fall back to using the reference kernel implementations.

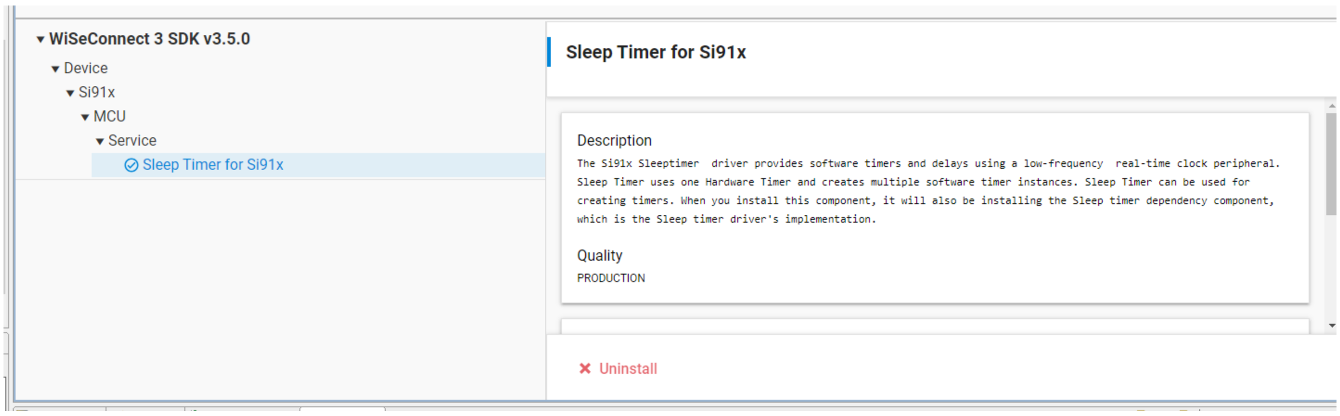
[✓ **MVPv1 Accelerated Kernels**]

- This component includes kernel implementations that are accelerated using the MVP accelerator on compatible devices. By including this component, the MVP kernel implementations will be used instead of the optimized kernels. The kernels which have MVP accelerated implementations are pooling, fully_connected, add, conv, depthwise_conv and transpose_conv. Remaining kernels fall back to using optimized or reference kernel implementations.

Additional Software Components and C++ Build Settings

> Note: Skip to Model Inclusion section for Series 2 devices. This section is only applicable for SiWx917 devices.

1. Ensure the following components are installed in your project.
   - WiseConnect SDK > Device > Si91X > MCU > Service > Power Manager > Sleep Timer for Si91x
   - Platform > Peripheral > Common Headers

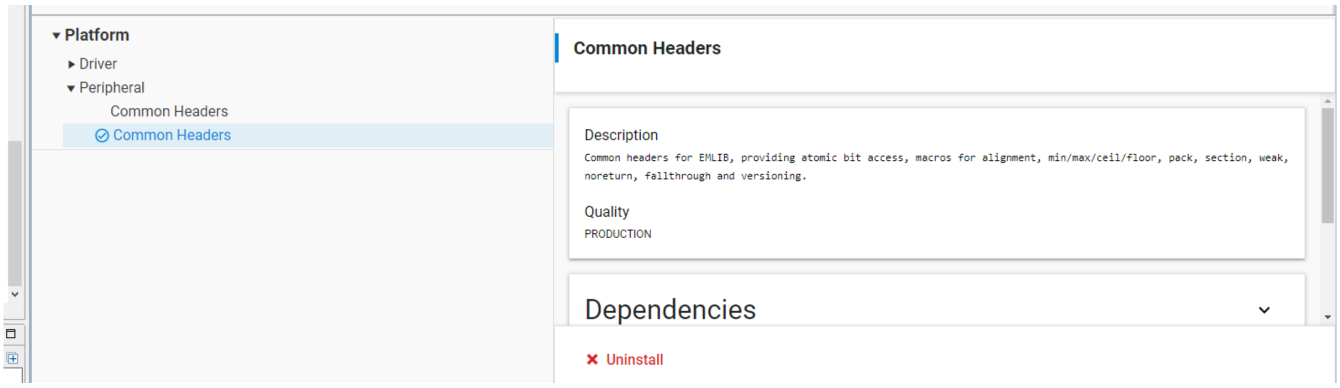


Description

The Si91x Sleeptimer driver provides software timers and delays using a low-frequency real-time clock peripheral. Sleep Timer uses one Hardware Timer and creates multiple software timer instances. Sleep Timer can be used for creating timers. When you install this component, it will also be installing the Sleep timer dependency component, which is the Sleep timer driver's implementation.

Quality

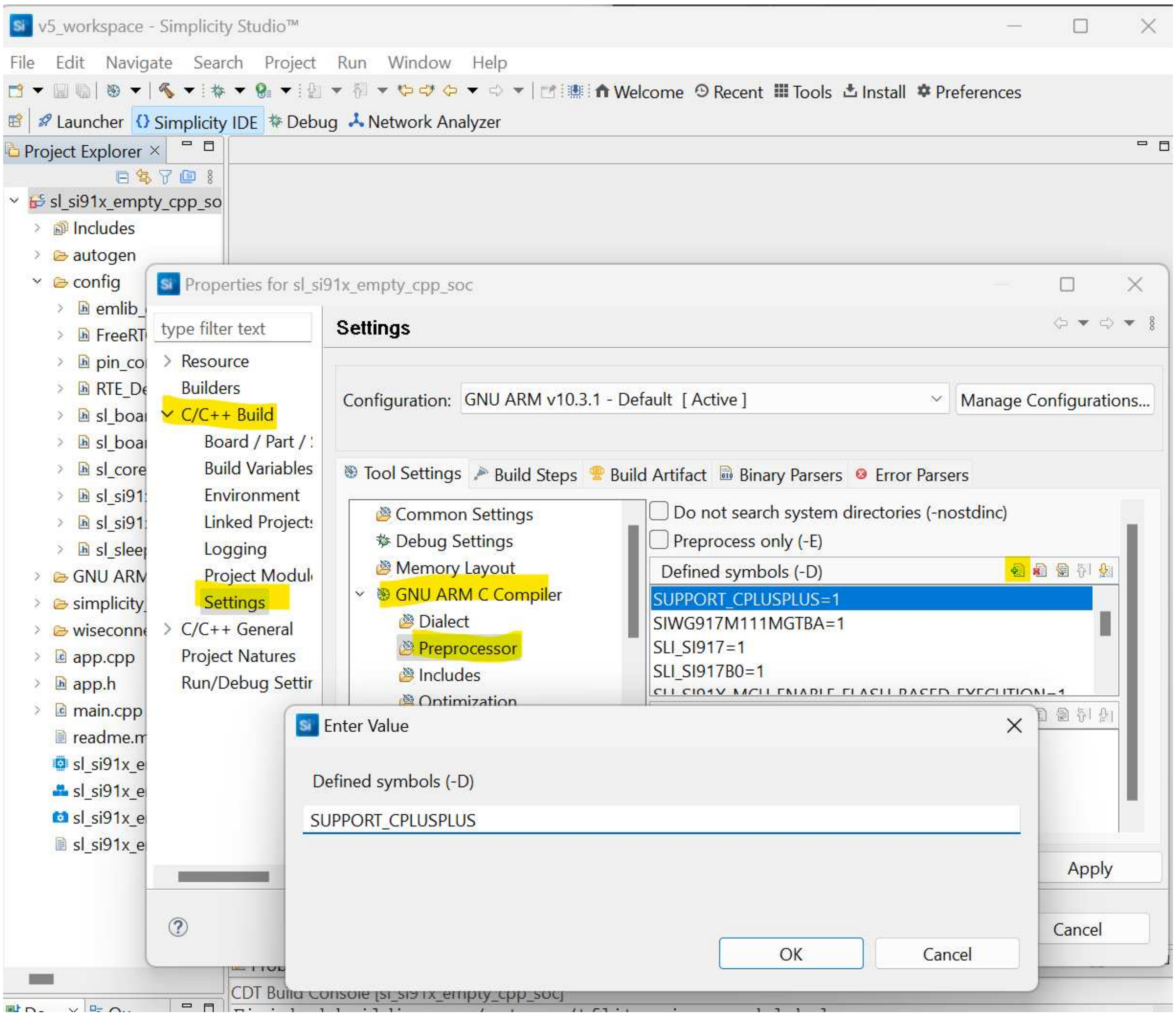
PRODUCTION



Description
Common headers for EMLIB, providing atomic bit access, macros for alignment, min/max/ceil/floor, pack, section, weak, noreturn, fallthrough and versioning.

Quality
PRODUCTION

2. Update your C++ build settings as follows:
   - In the C preprocessor defines, add: `SUPPORT_CPLUSPLUS` .
   - In GNU ARM C++ Compiler > Miscellaneous settings, add: `-mfp16-format=ieee` .



Settings

Configuration: GNU ARM v10.3.1 - Default [Active]

Manage Configurations...

☐ Do not search system directories (-nostdinc)

☐ Preprocess only (-E)

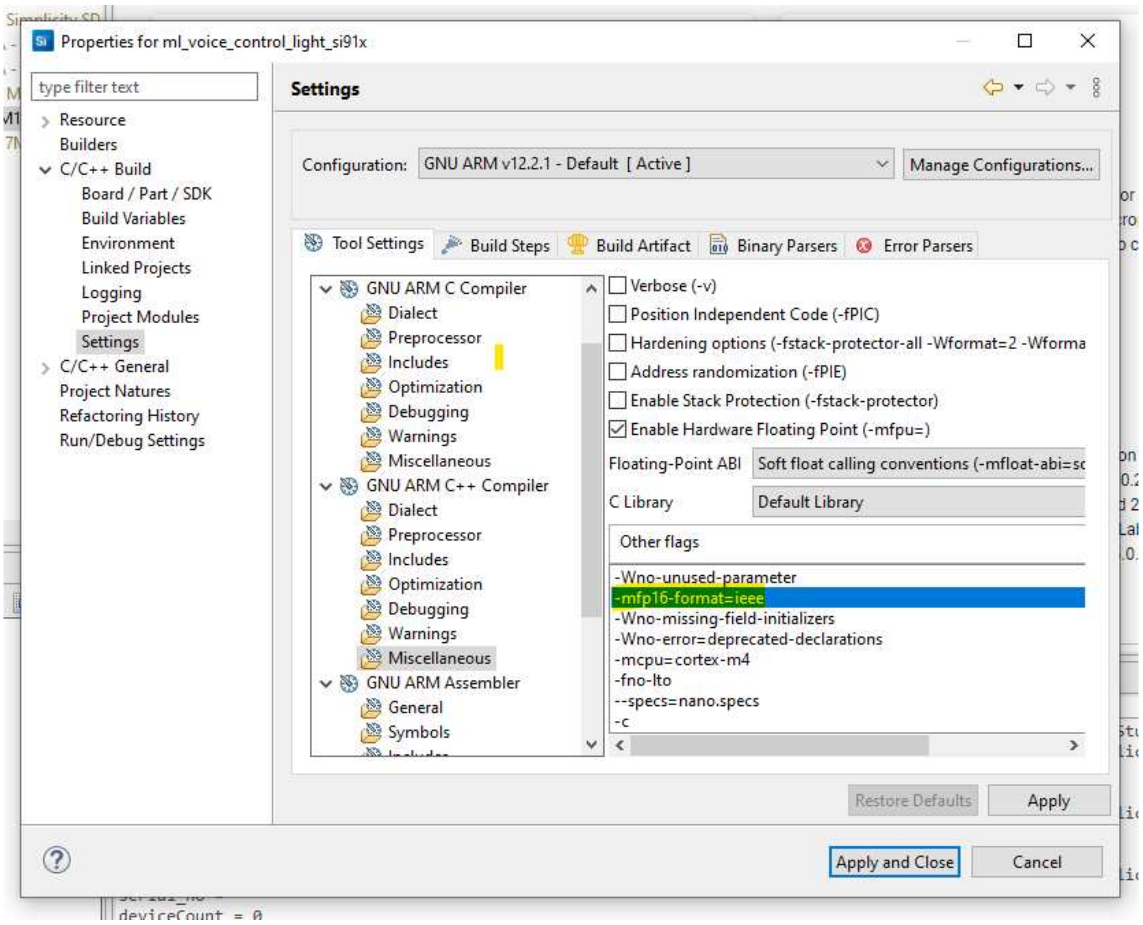
Defined symbols (-D)

```
SUPPORT_CPLUSPLUS=1
SIWG917M111MGTBA=1
SLI_SI917=1
SLI_SI917B0=1
```

Enter Value

Defined symbols (-D)

```
SUPPORT_CPLUSPLUS
```



Properties for ml_voice_control_light_si91x

Settings

Configuration: GNU ARM v12.2.1 - Default [Active]

Tool Settings: GNU ARM C++ Compiler > Miscellaneous

- [ ] Verbose (-v)
- [ ] Position Independent Code (-fPIC)
- [ ] Hardening options (-fstack-protector-all -Wformat=2 -Wforma
- [ ] Address randomization (-fPIE)
- [ ] Enable Stack Protection (-fstack-protector)
- [x] Enable Hardware Floating Point (-mfpu=)

Floating-Point ABI: Soft float calling conventions (-mfloat-abi=sc

C Library: Default Library

Other flags

```
-Wno-unused-parameter
-mfp16-format=ieee
-Wno-missing-field-initializers
-Wno-error=deprecated-declarations
-mcpu=cortex-m4
-fno-lto
--specs=nano.specs
-c
```

Alternate method:
You can also add these settings directly to your project's `.slcp` file:

- Open the `.slcp` file in a text editor.
- Add `SUPPORT_CPLUSPLUS` to the `define` section for C preprocessor defines. For example:

```
define:
  - name: SUPPORT_CPLUSPLUS
    value: 1
```

- Add `-mfp16-format=ieee` to the `toolchain_settings` section for C++ compiler flags, for example:

```
toolchain_settings:
  - option: gcc_compiler_option
    value: -mfp16-format=ieee
```

- Save the file and regenerate your project to apply the changes.

Model Inclusion

With the TensorFlow Lite Micro component added in the Project Configurator, the next step is to load the model file into the project. To do this, create a `tflite` directory inside the `config` directory of the project, and copy the `.tflite` model file into it. The project configurator provides a tool that will automatically convert `.tflite` files into `sl_tflite_micro_model` source and header files. The full documentation for this tool is available at Flatbuffer Converter Tool.

For SiWx917 devices, after copying the `.tflite` model file into the project, a header file named `sl_ml_model_<model_name>.h` is generated. This header provides access to the C array obtained from the converted `.tflite` file, as well as APIs to initialize and run the model.

Automatic Initialization

The TensorFlow framework is automatically initialized using the system initialization framework described in SDK Programming Model. This includes allocating a tensor arena, instantiating an interpreter and loading the model.

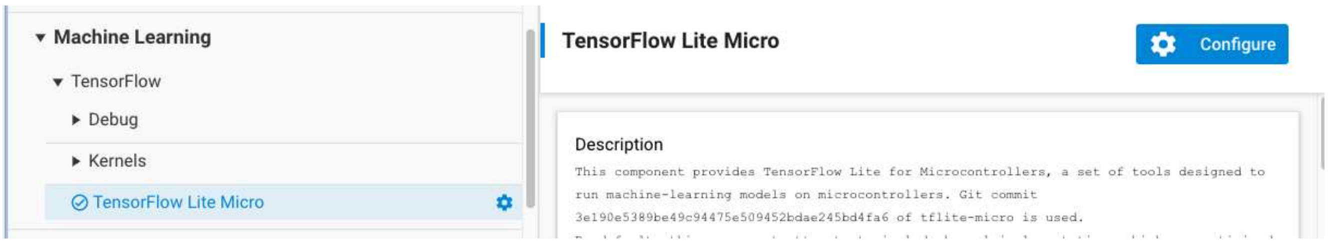
Configuration

> Note: All configuration for SiWx917 is taken care of automatically.

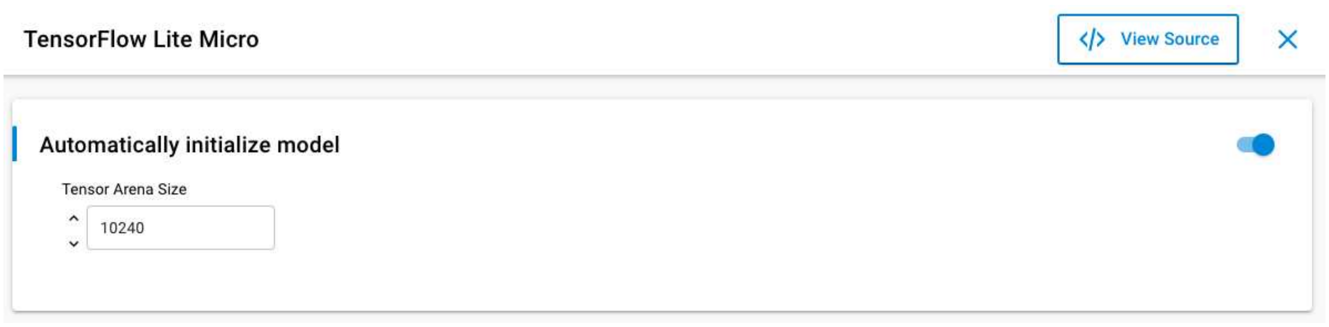
If the model was produced using the Silicon Labs Machine Learning Toolkit (MLTK), it already contains metadata indicating the required size of the Tensor Arena, the memory area used by TensorFlow for runtime storage of input, output, and intermediate arrays. The required size of the arena depends on the model used.

If not using the MLTK, the arena size needs to be configured. This can be done in two ways:

- [Automatic] Set the arena size to -1, and it will attempt to automatically infer the size upon initialization.
- [Manual] Start with a large number during development, and reduce the allocation until initialization fails as part of size optimization.



```
This component provides TensorFlow Lite for Microcontrollers, a set of tools designed to
run machine-learning models on microcontrollers. Git commit
3e190e5389be49c94475e509452bdae245bd4fa6 of tflite-micro is used.
```



Automatically initialize model: on

Tensor Arena Size: 10240

Run the Model for Series 2 Devices

Include the Silicon Labs TensorFlow Init API

```
#include "sl_tflite_micro_init.h"
```

For default behavior in bare metal application, it is recommended to run the model during `app_process_action()` in `app.cpp` to ensure that periodic inferences occur during the standard event loop. Running the model involves three stages:

Provide Input to the Interpreter

Sensor data is pre-processed (if necessary) and then is provided as input to the interpreter.

```
TfLiteTensor* input = sl_tflite_micro_get_input_tensor();
// stores 0.0 to the input tensor of the model
input->data.f[0] = 0.;
```

Run Inference

The interpreter is then invoked to run all layers of the model.

```
TfLiteStatus invoke_status = sl_tflite_micro_get_interpreter()->Invoke();
if (invoke_status != kTfLiteOk) {
TF_LITE_REPORT_ERROR(sl_tflite_micro_get_error_reporter(),
                "Bad input tensor parameters in model");
return;
}
```

Read Output for Series 2 devices

The output prediction is read from the interpreter.

```
TfLiteTensor* output = sl_tflite_micro_get_output_tensor();
// Obtain the output value from the tensor
float value = output->data.f[0];
```

At this point, application-dependent behavior based on the output prediction should be performed. The application will run inference on each iteration of `app_process_action()` .

Full Code Snippet (Series 2)

After following the steps above, the resulting `app.cpp` now appears as follows:

```
#include "sl_tflite_micro_init.h"

/***************************************************************************//**
 * Initialize application.
 ******************************************************************************/
void app_init(void)
{
  // Init happens automatically
}

/***************************************************************************//**
 * App ticking function.
 ******************************************************************************/
void app_process_action(void)
{
  TfLiteTensor* input = sl_tflite_micro_get_input_tensor();
  // stores 0.0 to the input tensor of the model
  input->data.f[0] = 0.;

  TfLiteStatus invoke_status = sl_tflite_micro_get_interpreter()->Invoke();
  if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(sl_tflite_micro_get_error_reporter(),
                       "Bad input tensor parameters in model");
  return;
  }

  TfLiteTensor* output = sl_tflite_micro_get_output_tensor();
  // Obtain the output value from the tensor
  float value = output->data.f[0];
}
```

Run the Model for SiWx917 Devices

For SiWx917 devices, the APIs differ from Series 2. Use the generated model C++ APIs to initialize and run your model. Below is a general guide for using these APIs:

1. Include Required Headers

```
//Provides _init() and _run() functions for this model
#include "sl_ml_model_<model_name>.h"
// ...other headers as needed...
```

2. System Initialization

```
sl_system_init();
```

3. Model Initialization

```
//Loads and Initializes the model with name <model_name>
//Initializes its parameters
//Initializes its interpreter
//Initializes its error reporter
//Allocates memory for model's tensors

sl_status_t status = slx_ml_<model_name>_model_init();
if (status != SL_STATUS_OK) {
   printf("Failed to initialize model\n");
   // Handle error
}
```

4. Access and Prepare Input Tensors

```
for (unsigned int i = 0; i < <model_name>_model.n_inputs(); ++i) {
  auto& input_tensor = *<model_name>_model.input(i);
  // Fill input_tensor.data with your input data
}
```

5. Run Inference

```
/// Executes the model with name <model_name> by invoking its interpreter for TFLite Micro and returns execution status.
/// The output is stored in the output tensor of the model itself.
status = slx_ml_model_<model_name>_run();
if (status != SL_STATUS_OK) {
  printf("Error while running inference\n");
  // Handle error
}
```

6. Access Output Tensors

```
for (unsigned int i = 0; i < <model_name>_model.n_outputs(); ++i) {
  auto& output_tensor = *<model_name>_model.output(i);
  // Read results from output_tensor.data
}
```

Full Code Snippet (SiWx917)

Below is a generic code snippet for running a model on SiWx917 devices. Replace `<model_name>` with your actual model name.

`app.h`

```
#ifndef APP_H
#define APP_H

#ifdef __cplusplus
extern "C" {
#endif

void app_init(void);
void app_process_action(void);

#ifdef __cplusplus
}
#endif

#endif  // APP_H
```

`app.cc`

```
#include "app.h"
#include "model_runner.h"

sl_status_t <model_name>_model_status = SL_STATUS_OK;

// Initialization logic
void app_init(void) {
  <model_name>_model_status = model_runner_init();
  if (<model_name>_model_status != SL_STATUS_OK) {
    // Print helpful error message or handle error
    return;
  }
}

void app_process_action(void) {
  if (<model_name>_model_status != SL_STATUS_OK) {
    // Print an error message or indicate error
    return;
  }
  model_runner_loop();
}
```

`model_runner.cc`

```
#include "sl_ml_model_<model_name>.h"

sl_status_t model_runner_init(void) {
  // Peripheral and other initialization logic

  <model_name>_model_status = slx_ml_<model_name>_model_init();

  if (<model_name>_model_status != SL_STATUS_OK) {
    // Peripheral deinit or cleanup logic
    return SL_STATUS_FAIL;
  }
  return <model_name>_model_status;
}

sl_status_t model_runner_loop(void) {
  // Data capture and pre-processing logic

  <model_name>_model_status = slx_ml_<model_name>_model_run();

  if (<model_name>_model_status != SL_STATUS_OK) {
    // Peripheral deinit or cleanup logic
    return SL_STATUS_FAIL;
  }

  // Post-processing logic

  return <model_name>_model_status;
}
```

Addendum: Interfacing with C Code

If your project is written in C rather than C++, place the code interfacing with TFLM into a separate file that exports a C API through an interface header. For this example, a filename `app_ml.cpp` is assumed that implements the function `ml_process_action()` with the same content as in the example above.

`app_ml.h`

```
#ifdef __cplusplus
extern "C" {
#endif

void ml_process_action(void);

#ifdef __cplusplus
}
#endif
```

`app_ml.cpp`

```
#include "app_ml.h"
#include "sl_tflite_micro_init.h"

extern "C" void ml_process_action(void)
{
  // ...
}
```

`app.c`

```
#include "app_ml.h"
// ...
void app_process_action(void)
{
  ml_process_action();
}
```

Addendum: Series 2 to SiWx917 (and vice versa) App Conversion

If you need to port an application from Series 2 to SiWx917, first remove the TensorFlow Lite Micro component from your project. Then, follow these steps for adding the required software components and C++ build settings. For SiWx917 devices, the APIs and workflow differ from Series 2. Use the generated model C++ APIs as described in the Run the Model for SiWx917 Devices section above. Refer to the provided code snippets and step-by-step instructions on this page for details on initialization, running inference, and accessing input/output tensors. For a complete example, see the Full Code Snippet (SiWx917) section. After updating your project, regenerate it to ensure all dependencies are correctly configured.

Update or Replace a .tflite File

Updating or Replacing the .tflite File in a Project

This guide describes how to swap out the model in an existing project. It assumes that the project uses the Flatbuffer Converter Tool.

Replace the Model

To replace the model in an existing project, drag-and-drop the new model file into the `config/tflite/` directory of the project. If the new model has the same file name as the previous model, accept the prompt to overwrite the file. If the new model has a different name, delete or rename the old `.tflite` file such that it no longer has the `.tflite` extension. The Flatbuffer Converter Tool will automatically execute when the directory watcher notices that a different `.tflite` file is present.

> Note: If you have multiple `.tflite` files in the `config/tflite/` directory, the converter tool will pick the first file in alphabetical order. Hence the recommendation to rename or delete any old models to ensure that the tool uses the correct file.

Inspect the Model

You can take steps to ensure that the automatic regeneration of the C arrays and headers from the `.tflite` file executed as expected. Any of the below steps can be taken if you are ever unsure of what model is part of your application binary.

Check the Model Size

The generated file `autogen/sl_tflite_micro_model.c` defines a size variable `sl_tflite_model_len`. This number can be compared to the file size of the `.tflite` file in bytes, for example, by right-clicking the `.tflite` file and looking at the value of Properties > Resource > Size.

Check the Operators Used by the Model

The generated file `autogen/sl_tflite_micro_opcode_resolver.h` contains multiple calls to `tflite::MicroMutableOpResolver::AddXXX`, where `XXX` is the name of operators used by the model. This can be compared to the operators you know the model *should* use.

Check the Model Parameters

If the `.tflite` file was generated using the Silicon Labs Machine Learning Toolkit, it contains metadata that is generated into `autogen/sl_tflite_micro_model_parameters.h`. These parameters can be compared to the expected parameters.

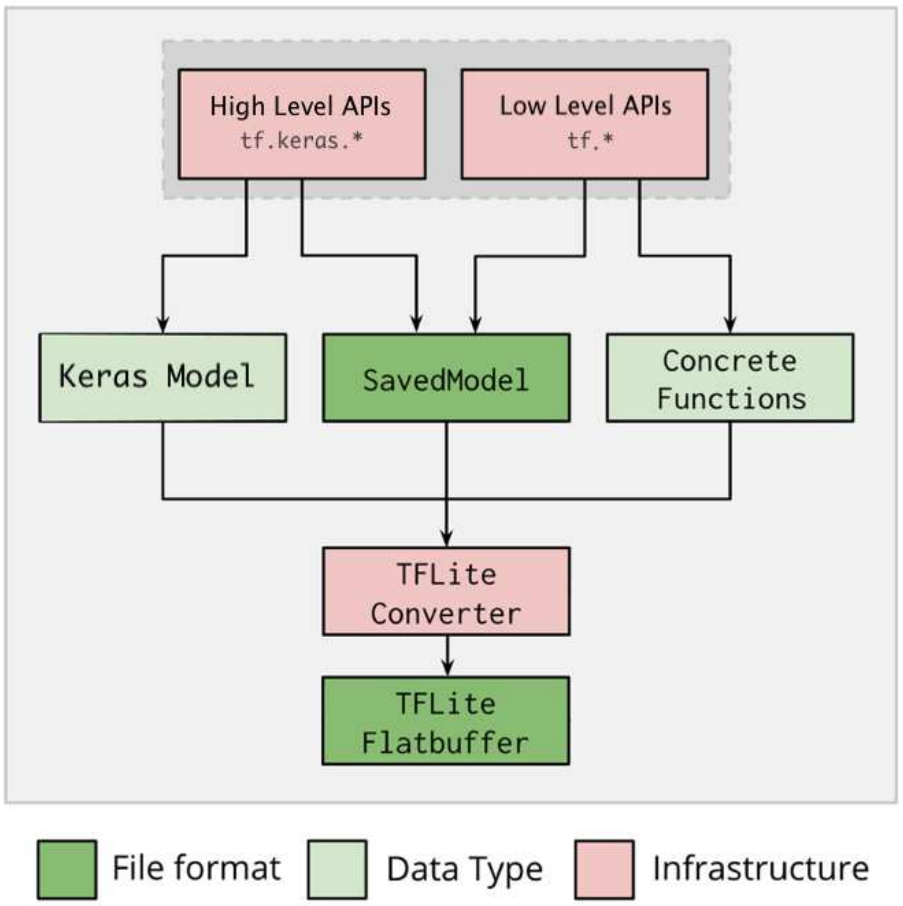
Force Generation of C files

If anything happens that makes the generated `.c` and `.h` get out of sync with the `.tflite` file, the project can be forcefully regenerated by pressing the Force Generation button on the Project Details pane of the Project Configurator.

Developing a Model

Developing a Machine Learning Model

Developing a Model Manually using TensorFlow and Keras



When developing and training neural networks for use in embedded systems, it is important to note the limitations on TFLM that apply to model architecture and training. Embedded platforms also have significant performance constraints that must be considered when designing and evaluating a model. The embedded TLFM documentation links describe these limitations and considerations in detail.

Additionally, the TensorFlow Software Components in Simplicity Studio require a quantized `*.tflite` representation of the trained model. As a result, TensorFlow and Keras are the recommended platforms for model development and training because both platforms are supported by the TensorFlow Lite Converter that generates `.tflite` model representations.

Both TensorFlow and Keras provide guides on model development and training:

TensorFlow Basic Training Loops
- Keras Training and Evaluation

After a model has been created and trained in TensorFlow or Keras, it needs to be converted and serialized into a `*.tflite` file. During model conversion, it is important to optimize the memory usage of the model by quantizing it. It is highly recommended to use integer quantization on Silicon Labs devices.

- TensorFlow Lite Converter
- Quantization Overview

A complete example demonstrating the training, conversion, and quantization of a simple TFLM compatible neural network is available from TensorFlow:

- TensorFlow Hello World Training Example
- Trained Hello World Model (Downloads a .zip file containing the model)

Developing a Model using Silicon Labs AI/ML Partners

See the AI/ML Partners section on the Silicon Labs Machine Learning in IoT page for more information about Silicon Labs AI/ML partners. These partners can help you develop a model that is compatible with Silicon Labs devices.

Developing a Model using the MLTK

> Note: MLTK is experimental and official support is unavailable. It is expected that the MLTK is used by an ML Expert with deep knowledge of TensorFlow and Python, or by a developer willing to learn.

The Silicon Labs Machine Learning Toolkit (MLTK) is a Python package that implements a layer above TensorFlow to help the TensorFlow developer build models that can be successfully deployed on Silicon Labs chips. These scripts are a reference implementation for the audio use case, which includes the use of the Audio Feature Generator on both the training and inference side. This is modified version of the TensorFlow "microfrontend" audio front end.

The MLTK is offered as a self-serve, self-support, fully documented, Python reference package published through GitHub. We are delivering this as an Experimental package, which means it’s available “as-is”, un-tested, and without support.

See the MLTK documentation for more information.

AI/ML Extension Setup

Installing the AI/ML Extension for Silicon Labs Simplicity Studio

This guide details the installation process for the AI/ML extension within Silicon Labs Simplicity Studio. This extension empowers developers to integrate machine learning capabilities into their Silicon Labs-based projects.

Prerequisites

Before proceeding with the installation, ensure you have the following:

- Silicon Labs Simplicity Studio: The latest version of Simplicity Studio is essential. Download it from the official Silicon Labs website.
- Supported Hardware: Verify that your target Silicon Labs hardware is compatible with the AI/ML extension. Consult the extension's release notes or documentation for a list of supported devices.
- Python Environment (Potentially): Some AI/ML functionalities might rely on a Python environment. It is recommended to have Python 3.7 or higher installed. A virtual environment is strongly advised to manage dependencies effectively.
- Git (Potentially): Git might be required for certain installation methods or for accessing specific resources. Ensure Git is installed on your system.

Installation Methods

The AI/ML extension can be installed via the following methods:

1. Simplicity Installer (Recommended)

This is the most straightforward and recommended approach.

1. Download Simplicity Installer using this link for your OS.
2. Open Simplicity Installer.
3. Click on Installation Wizard.
4. Click Technology Install.
5. Select the AI/ML from the list of available extensions under Optional Packages.
6. Click NEXT. On the next page, accept the Terms of Use and License Agreement, and then click INSTALL to begin the installation.

2. Silicon Labs Tool (command line)

1. Download Silicon Labs Tool (SLT) using this link for your OS.
2. Unzip the downloaded zip file to your preferred location. Add this location to your PATH environment variable so that `slt` command is available globally.
3. Install Simplicity SDK by invoking `slt install simplicity-sdk` on the command line.

Post-Installation Steps

After successful installation:

1. Start Simplicity Studio: Restart Simplicity Studio to ensure the changes take effect.

2. Verify Installation: Check the Simplicity Studio preferences or installed software list to confirm that the AI/ML extension is listed and installed.
3. Explore Documentation and Examples: The AI/ML extension should include documentation and example projects. These resources are crucial for getting started and understanding how to use the extension's features and APIs.

Troubleshooting

If you encounter any issues during installation:

1. Check Simplicity Studio Logs: Examine the Simplicity Studio logs for any error messages. These logs can provide valuable clues for troubleshooting.
2. Review Documentation: Refer to the AI/ML extension's documentation and release notes for troubleshooting tips and known issues.
3. Silicon Labs Community Forum: The Silicon Labs community forum is an excellent resource for finding solutions to common problems or asking for help from other users and Silicon Labs experts.

Updating the Extension

To update the AI/ML extension, follow the same installation steps as described above. Simplicity Studio will typically detect the newer version and guide you through the update process.

Uninstallation

To uninstall the AI/ML extension:

1. Open Simplicity Installer.
2. Click Package Manager.
3. Search for Simplicity SDKs.
4. Click the SDK from which you want to uninstall the AI/ML Extension and click the Trash Bin icon beside it.

Flatbuffer Converter Tool

Flatbuffer Converter Tool

The Flatbuffer Converter Tool helps to convert a `.tflite` file into a C array that can be compiled into a binary for an embedded system. This array can be used with the TensorFlow Lite for Microcontrollers API, which takes a void pointer to a buffer containing the model as an argument to its `tflite::GetModel()` init function.

In addition to converting the flatbuffer into a C array, the tool supports emitting model parameters embedded as metadata in the `.tflite` file as C preprocessor macros.

Input

The tool takes a directory containing one or more `.tflite` files as input. If the directory consists of multiple files, only the first file in alphabetical order is converted.

> Tip: If you have multiple files in the directory, but the one you want to convert isn't the first file in alphabetical order, you can rename the other files to add a `.bak` extension or rename the target file accordingly.

Output

The tool writes its output into multiple files in a single output directory.

Model Array

The tool always emits a pair of files `sl_tflite_micro_model.c` / `.h`, which declares the variables as follows.

- `const uint8_t sl_tflite_model_array[]` containing the full contents of the `.tflite` file
- `const uint32_t sl_tflite_model_len` containing the length of the model array

Opcode Resolver

A header file `sl_tflite_micro_opcode_resolver.h` is also emitted. This file declares a C preprocessor macro `SL_TFLITE_MICRO_OPCODE_RESOLVER(opcode_resolver, error_reporter)` that instantiates a `tflite::MicroMutableOpResolver` object and automatically registers the set of operators required to parse the model.

This macro can be used as part of an initialization sequence to automatically initializing the optimal opcode resolver.

Model Parameters

If the `.tflite` file contains model parameters in its metadata section, a third header file `sl_tflite_micro_model_parameters.h` is emitted.

For every model parameter key-value pair, a C preprocessor macro `SL_TFLITE_MODEL_<key>` is created with the relevant value.

See the MLTK documentation to learn more about embedding model parameters in the `.tflite` file.

SLC Project Configuration Integration

The Flatbuffer Converter Tool integrates with the SLC project configuration tools in Simplicity Studio and on the command line using SLC-CLI. When using these tools, the Flatbuffer Converter is automatically run with the `config/tflite/` directory of the project as input, and the `autogen/` directory as output.

In other words, any file with the `.tflite` extension in the `config/tflite/` directory in the project will be automatically converted when the project is generated. If using Simplicity Studio, a directory watcher ensures that the project is automatically generated if a `.tflite` file is added or removed. This means that it is sufficient to drag-and-drop a `.tflite` file into the project, and it is automatically converted into C code.

Manual Usage

If conversion is desired outside of a full SLC project generation cycle, the flatbuffer converter can be invoked manually. This is generally not necessary, but is an option for advanced users. This can only be done using the SLC-CLI, users of Simplicity Studio are recommended to regenerate the full project.

As Standalone Conversion Command

The Flatbuffer Converter Tool runs as a standalone Python script outside of a project generation flow. Execute the following command using absolute paths for both input and output directories.

- NumPy must be installed. If not install it using:

```
pip install numpy
```

- The output directory must already exist before running the tool. Create one in any location you prefer:

```
mkdir -p /path/to/output/folder
```

- Identify your TFLite model folder where `.tflite` model files are stored. You may also use your own custom `.tflite` model files by supplying their absolute path to `-i`.
- Run `slt where aiml` to locate the AIML package installation path, and then go to the `tool/tflite` directory where `tflite.py` is located.
- Navigate to the folder containing flatbuffer_converter.py:

```
cd "$(slt where aiml)/tool/flatbuffer-converter/flatbuffer_converter.py"
```

- Run the Flatbuffer Converter Tool using:

```
python flatbuffer_converter.py generate </path/to/tflite/model/folder> </path/to/output/folder> <part_number>
```

Tool Help

You can run the help command at any time to view the latest usage instructions for this tool:

```
python flatbuffer_converter.py generate --help
```

```
usage: flatbuffer_converter.py generate [-h] input_dir output_dir part_number

Runs when project is generated using SLC-CLI or Studio.

positional arguments:
  input_dir    Input directory containing .tflite files
  output_dir   Output directory to populate with serialized content.
  part_number  Part Number

options:
  -h, --help   show this help message and exit
```

I2S Configuration for SiWx917

I2S Pin Configuration for ML Audio Applications on SiWx917 platform

The default configuration of I2S pins on SiWx917 does not provide the expected output for classification results on hardware for all AI/ML audio applications. You MUST manually configure the I2S pins using the `.slcp` as described below.

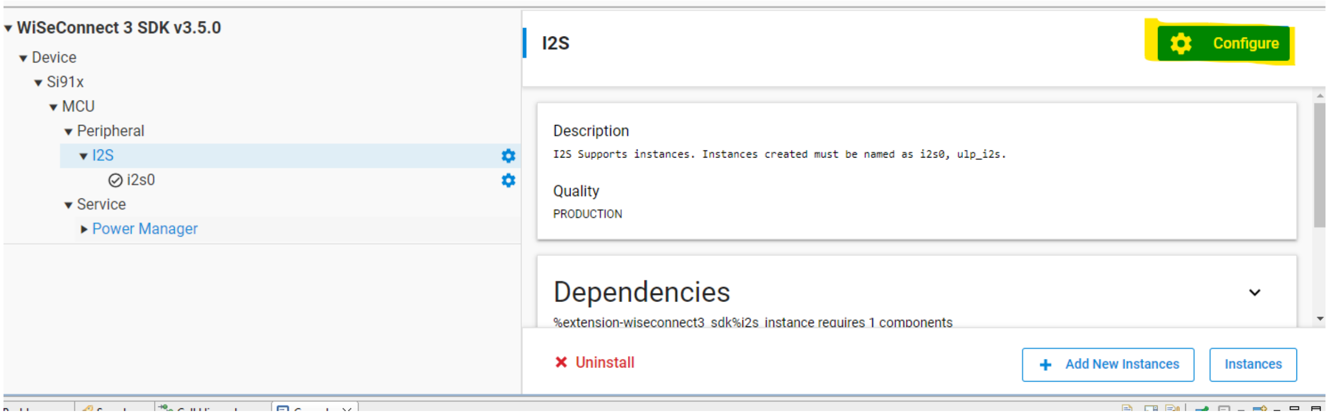
To configure the I2S pins for your board, follow these steps using Simplicity Studio:

1. Enable I2S Peripheral via Software Components

Once the .pintool file has been generated in your project folder, open the .slcp file in Simplicity Studio. After that, navigate to the Software Components tab. Search for and enable the I2S peripheral (e.g., `I2S0`). Navigate to WiSeConnect 3 SDK v3.5.0 > Device > Si91x > MCU > Peripheral > I2S > i2s0 > Configure.

2. Configure I2S Pins

In the Software Components configuration, click the Configure button next to the enabled I2S peripheral.



Description

I2S Supports instances. Instances created must be named as i2s0, ulp_i2s.

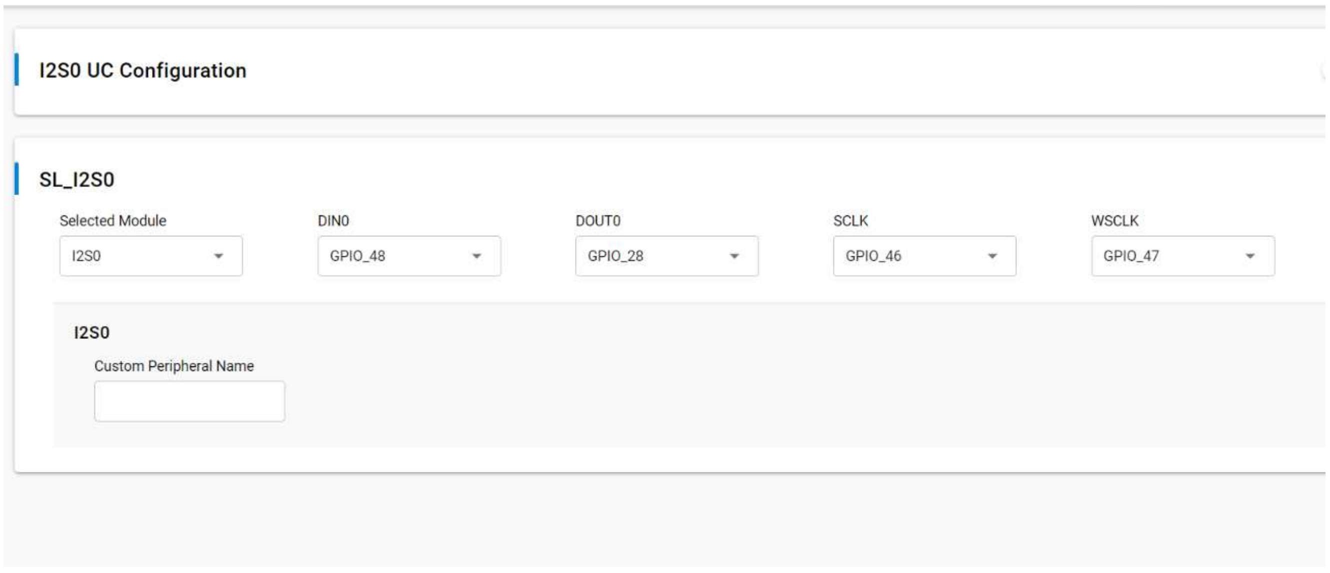
Quality

PRODUCTION

Dependencies

%extension-wiseconnect3_sdk%i2s_instance requires 1 components

Assign the required I2S signals (`DIN0` , `SCLK` , and `WSCLK`) to the appropriate GPIO pins as shown in the image below. The configuration tool displays available pins. Ensure you select only those indicated in the image.


I2S0 UC Configuration

SL_I2S0

Selected Module	DIN0	DOUT0	SCLK	WSCLK
I2S0	GPIO_48	GPIO_28	GPIO_46	GPIO_47

I2S0

Custom Peripheral Name

3. Save and Generate Code

The tool automatically generates the necessary pin configuration code for your project.

4. Rebuild the Project

Build your project to ensure the new pin configuration is included.

For more details, refer to the Simplicity Studio Software Components documentation.

ML Profiler

Silicon Labs Machine Learning Model Profiler

Overview

The Silicon Labs Machine Learning Model Profiler is a performance analysis tool designed to help engineers understand how TensorFlow Lite for Microcontrollers models (`.tflite`) execute on Silicon Labs embedded devices. The purpose of this tool is also to help developers optimize their models before deploying to production.

The profiler enables users to:

- Correlate layer execution with power usage, clock rate, and memory pressure (Power usage profiling is a planned feature)
- Build intuition for optimizing models to reduce stalls and improve inference latency
- Identify performance bottlenecks during inference
- Understand trade-offs between execution on the ARM Cortex-M MCU and the Matrix Vector Processor (MVP).

The tool is available as:

- a standalone GUI
- a command-line interface (CLI) called `sml`
- a Python API (planned feature)

It can also be launched directly from Simplicity Studio v6.

Who This Tool Is For

This documentation is written primarily for embedded and ML engineers.
Product managers, data scientists, and sales engineers are expected to be sufficiently familiar with machine learning concepts to interpret the results.

The profiler focuses exclusively on execution performance, not model accuracy or output quality.

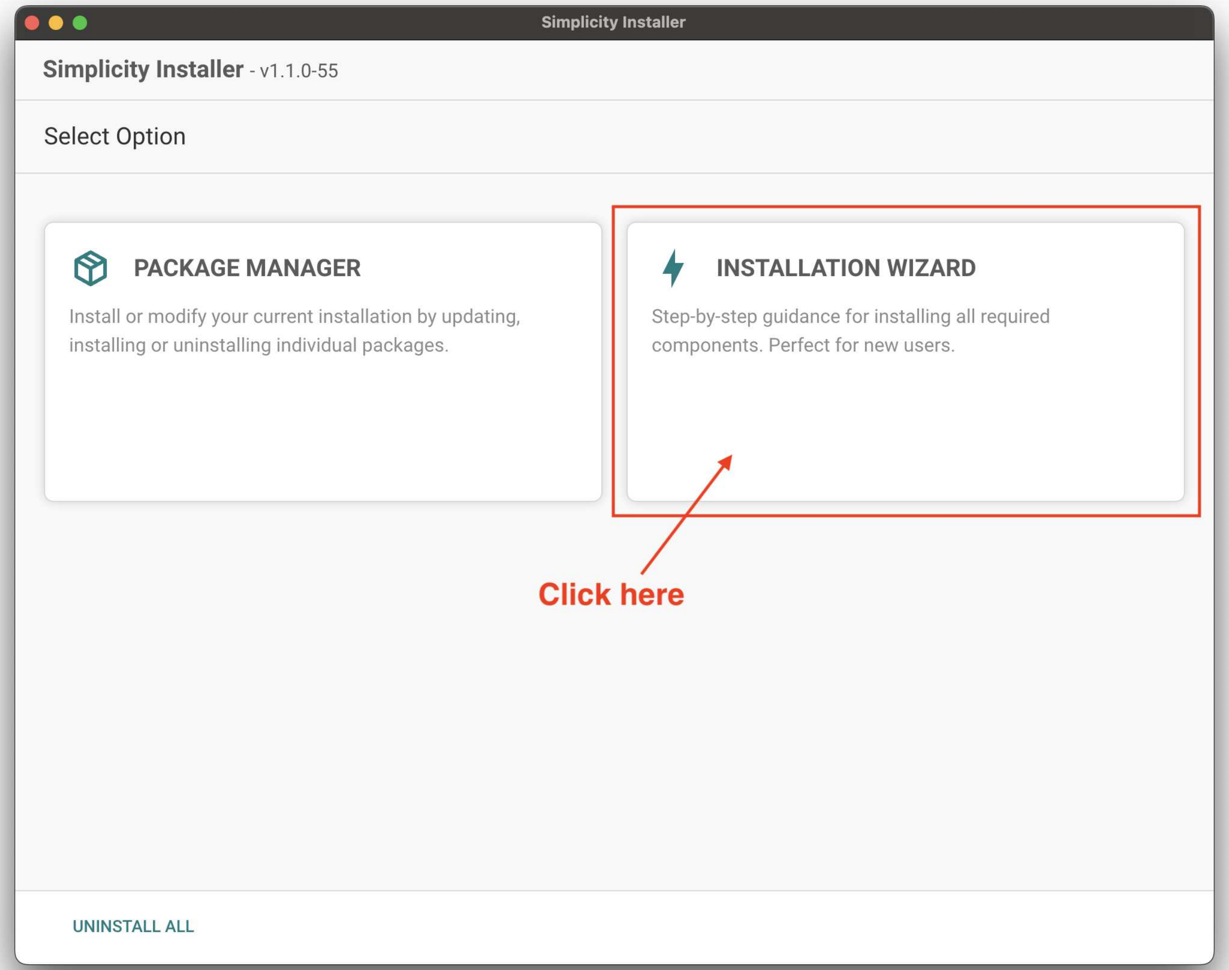
Key Concepts and Terminology

Term	Meaning
Inference	One complete execution of a model
Layer	A neural network layer
Operator	A logical operation within a layer
Kernel	The concrete implementation that executes an operator
MCU	ARM Cortex-M core
MVP	Matrix Vector Processor hardware accelerator
Stall / Wait	Time spent idle due to memory or resource contention
Power	Power consumed during execution
Tensor Arena	Memory allocated for TFLM state
Quantization	Optimization technique to reduce model size
Perfetto	The trace visualization tool used

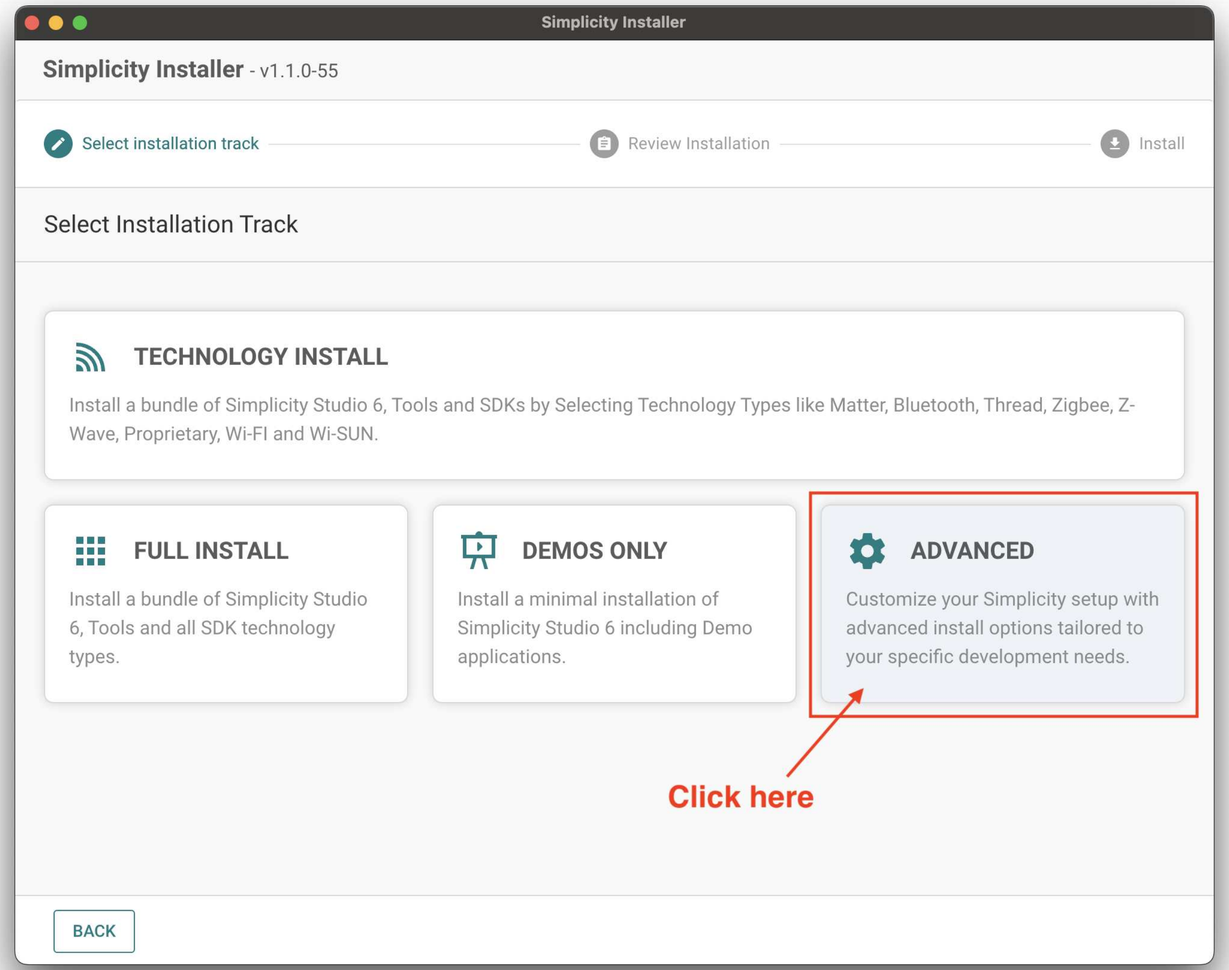
Installation

Using Simplicity Installer

1. Download Simplicity Installer using this link for your OS.
2. Once Installed, open Simplicity Installer and navigate to Installation Wizard.


Simplicity Installer
Simplicity Installer - v1.1.0-55
Select Option
PACKAGE MANAGER
Install or modify your current installation by updating, installing or uninstalling individual packages.
INSTALLATION WIZARD
Step-by-step guidance for installing all required components. Perfect for new users.
Click here
UNINSTALL ALL

3. Click on Advanced tile.



Simplicity Installer - v1.1.0-55

Select Installation Track

TECHNOLOGY INSTALL
Install a bundle of Simplicity Studio 6, Tools and SDKs by Selecting Technology Types like Matter, Bluetooth, Thread, Zigbee, Z-Wave, Proprietary, Wi-FI and Wi-SUN.

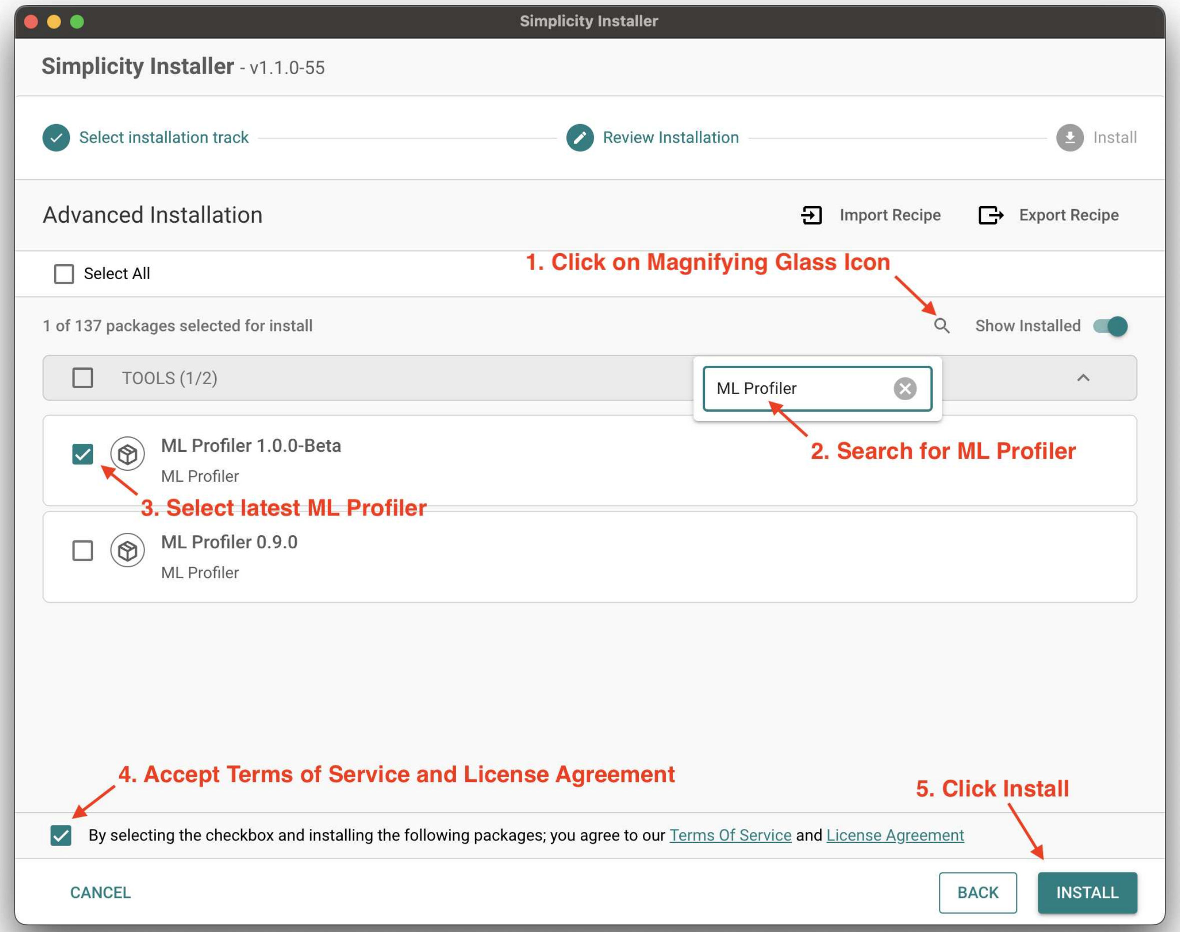
FULL INSTALL
Install a bundle of Simplicity Studio 6, Tools and all SDK technology types.

DEMOS ONLY
Install a minimal installation of Simplicity Studio 6 including Demo applications.

ADVANCED
Customize your Simplicity setup with advanced install options tailored to your specific development needs.

4. Search for ML Profiler:
   1. Click the Magnifying Glass Icon.
   2. Type *ML Profiler*, or a part of it, in the Search Box for GUI version, or *Silabs Machine Learning* for CLI version (`sml`).
   3. Select the latest ML Profiler by selecting the checkbox beside it. Versioning is based on semantic versioning.
   4. Accept Terms of Service and License Agreement by selecting the checkbox at the bottom.

5. Click the Install button.


Simplicity Installer

Simplicity Installer - v1.1.0-55

Select installation track — Review Installation — Install

Advanced Installation   Import Recipe   Export Recipe

Select All

1 of 137 packages selected for install   Show Installed

TOOLS (1/2)

ML Profiler

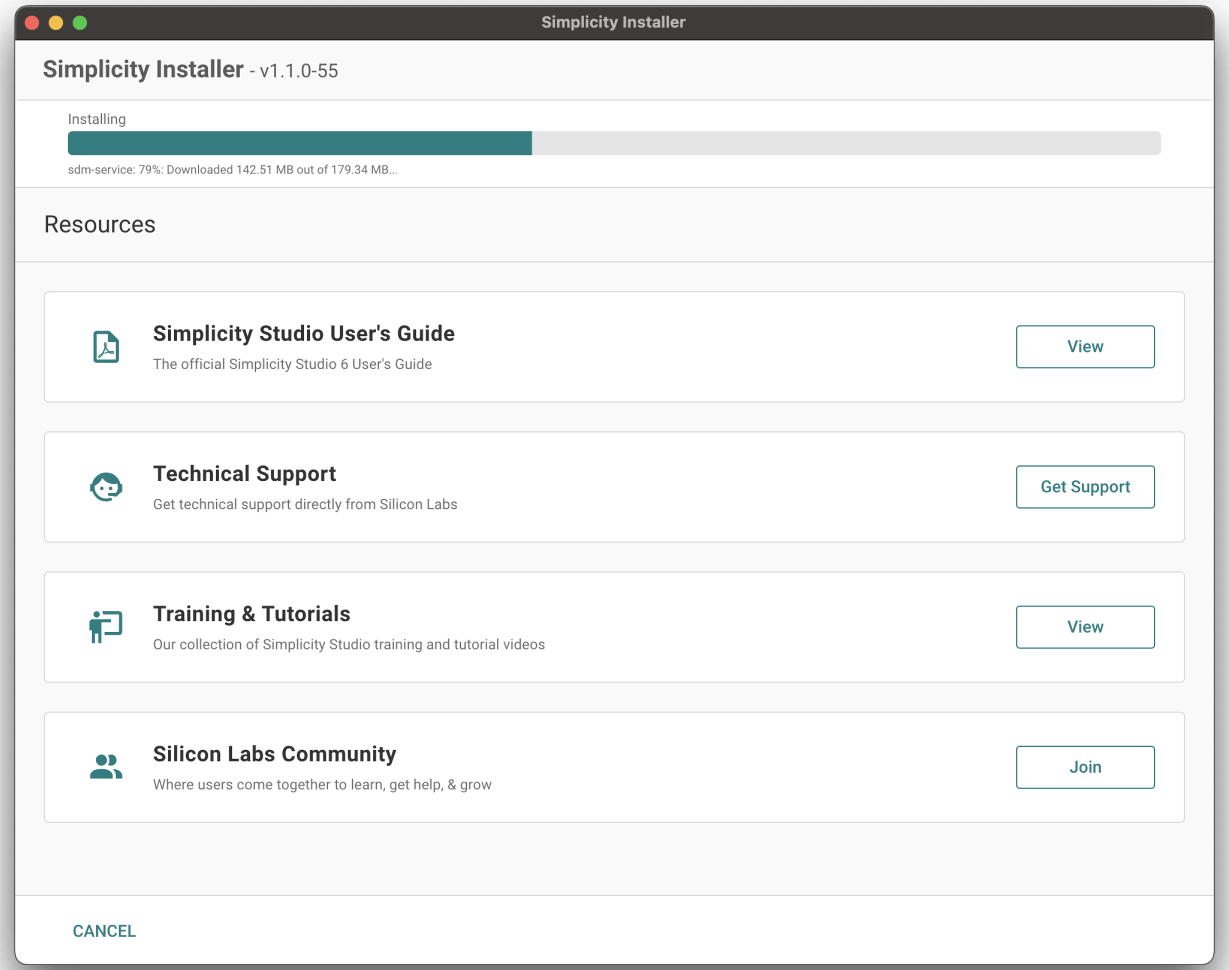
- [x] ML Profiler 1.0.0-Beta — ML Profiler
- [ ] ML Profiler 0.9.0 — ML Profiler

By selecting the checkbox and installing the following packages; you agree to our Terms Of Service and License Agreement

CANCEL   BACK   INSTALL

1. Click on Magnifying Glass Icon
2. Search for ML Profiler
3. Select latest ML Profiler
4. Accept Terms of Service and License Agreement
5. Click Install

5. Installation should progress, as shown below.



Simplicity Installer

Simplicity Installer - v1.1.0-55

Installing

sdm-service: 79%: Downloaded 142.51 MB out of 179.34 MB...

Resources

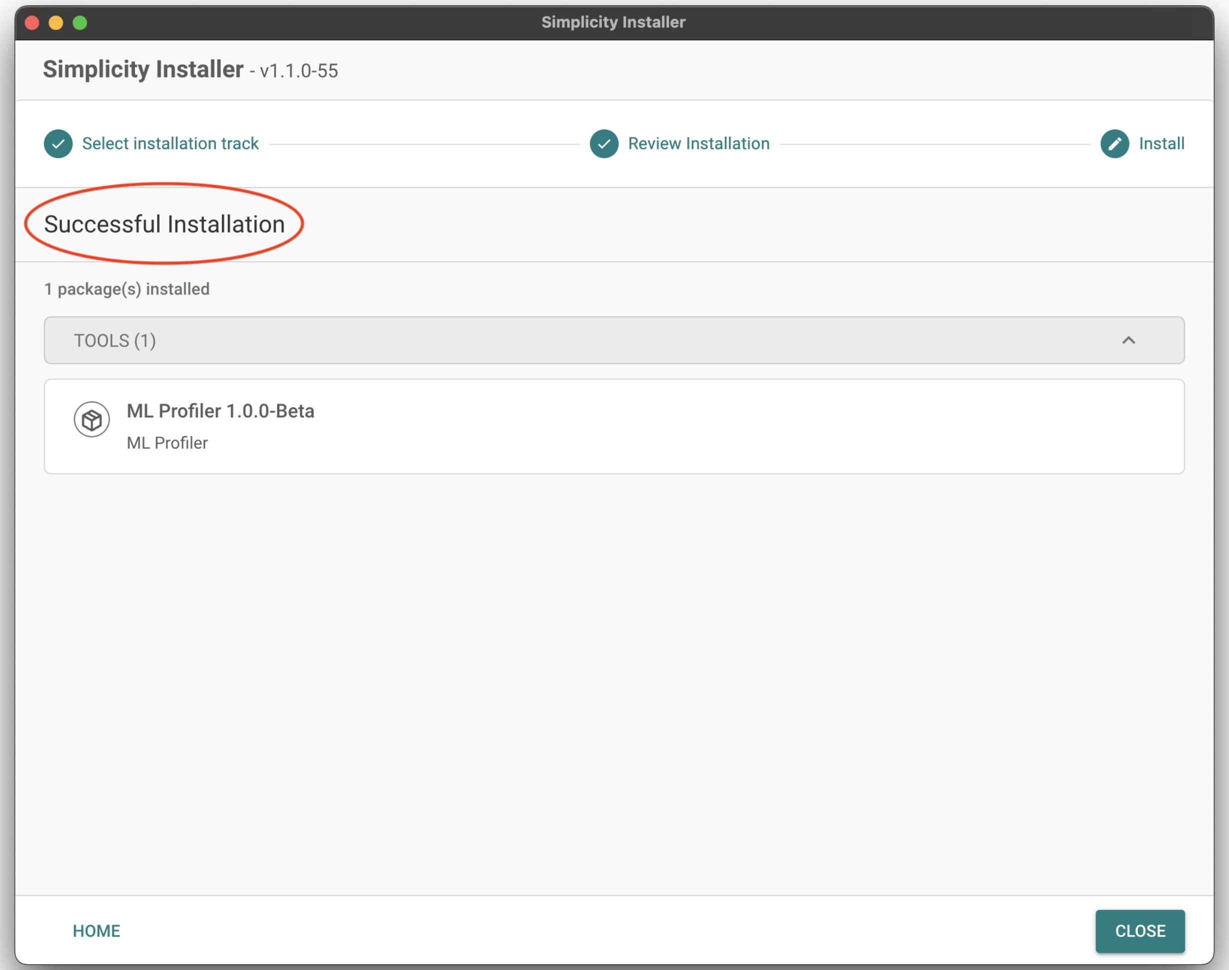
Simplicity Studio User's Guide
The official Simplicity Studio 6 User's Guide

Technical Support
Get technical support directly from Silicon Labs

Training & Tutorials
Our collection of Simplicity Studio training and tutorial videos

Silicon Labs Community
Where users come together to learn, get help, & grow

6. Once the installation is successful, you should see the following.



Successful Installation

1 package(s) installed

TOOLS (1)

ML Profiler 1.0.0-Beta
ML Profiler

7. If you installed the CLI version (`sml`), add its location to your PATH environment variable. You can find the location using `slt where sml` .

Using SLT on the command line

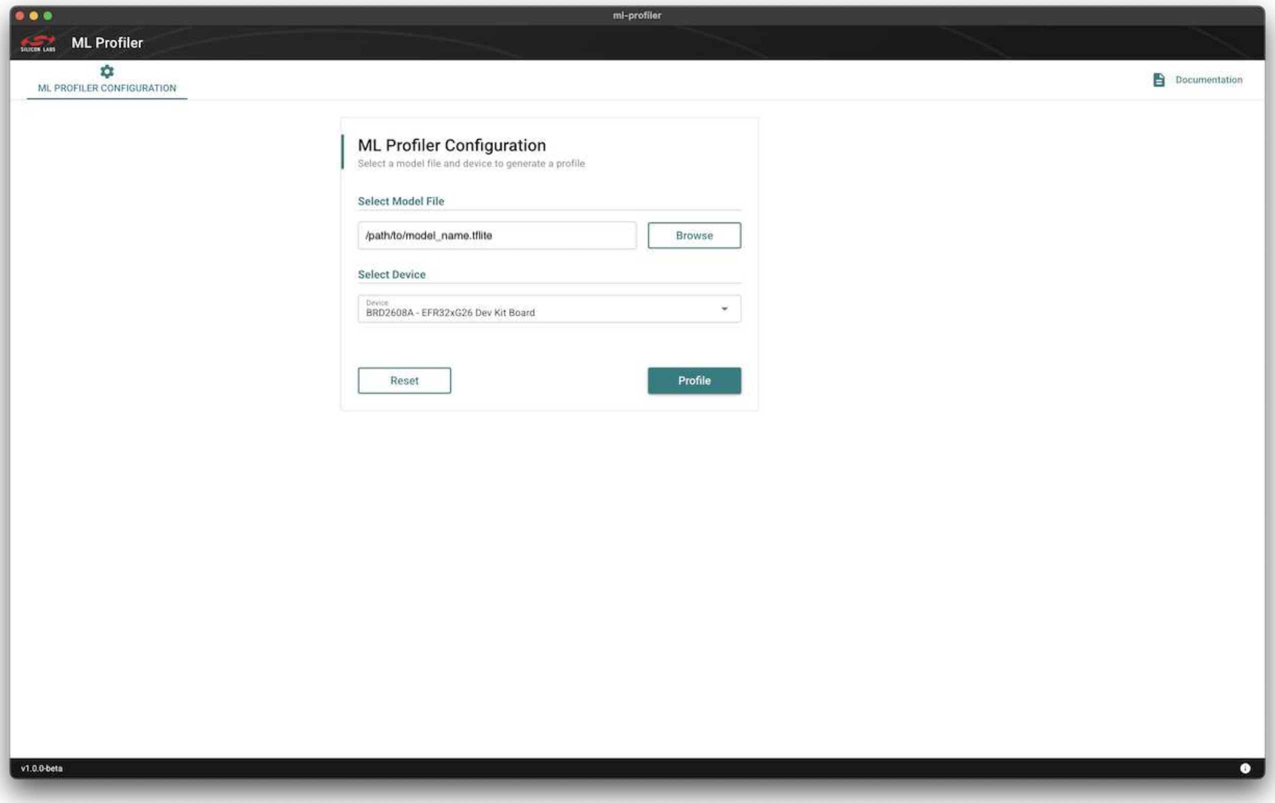
1. Download Silicon Labs Tool (SLT) using this link for your OS.
2. Unzip the downloaded zip file to your preferred location. Add this location to your PATH environment variable so that `slt` command is available globally.
3. Install ML Profiler using `slt install ml-profiler` for the GUI version or `slt install sml` for the CLI version.

Running a Profiling Session from the GUI

1. Launch the ML Profiler from Simplicity Studio v6
   1. Open Simplicity Studio and navigate to the Tools tab on the left panel.
   2. Hover on ML Profiler and click the Play icon that appears on the right side.
2. Or, Launch the ML Profiler from the command line:

```
slt launch ml-profiler
```

3. Step 1 or 2 should launch the ML Profiler. It is recommended to keep the window of ML Profiler maximized or in full-screen for the best user experience.



ML Profiler Configuration

Select a model file and device to generate a profile

Select Model File

/path/to/model_name.tflite

Browse

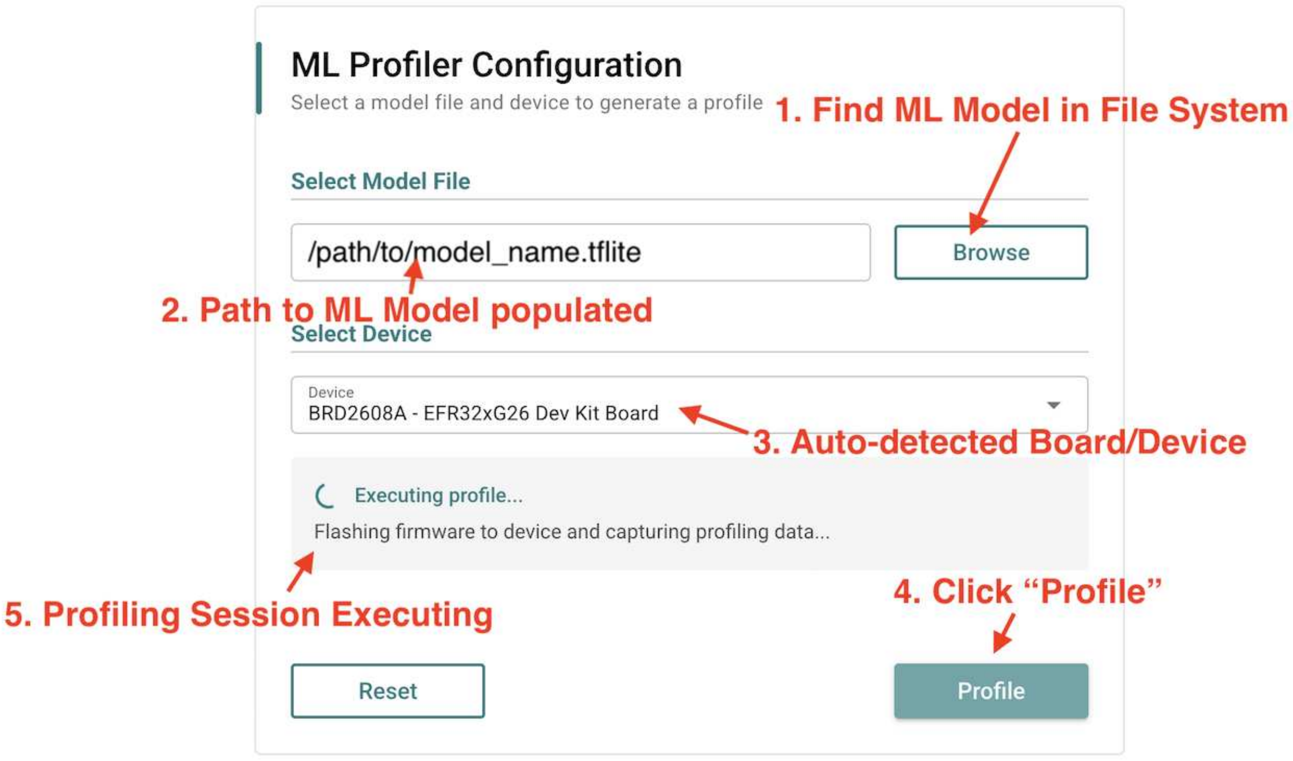
Select Device

Device: BRD2608A - EFR32xG26 Dev Kit Board

Reset

Profile

4. Connect the board you on which you want to profile your model. The board will be detected automatically.
5. Select the `.tflite` model you want to profile by clicking the Browse button and navigating your file system for the model file.
6. Click the Profile Button


ML Profiler Configuration

Select a model file and device to generate a profile

Select Model File

/path/to/model_name.tflite

Browse

Select Device

Device: BRD2608A - EFR32xG26 Dev Kit Board

Executing profile...

Flashing firmware to device and capturing profiling data...

Reset

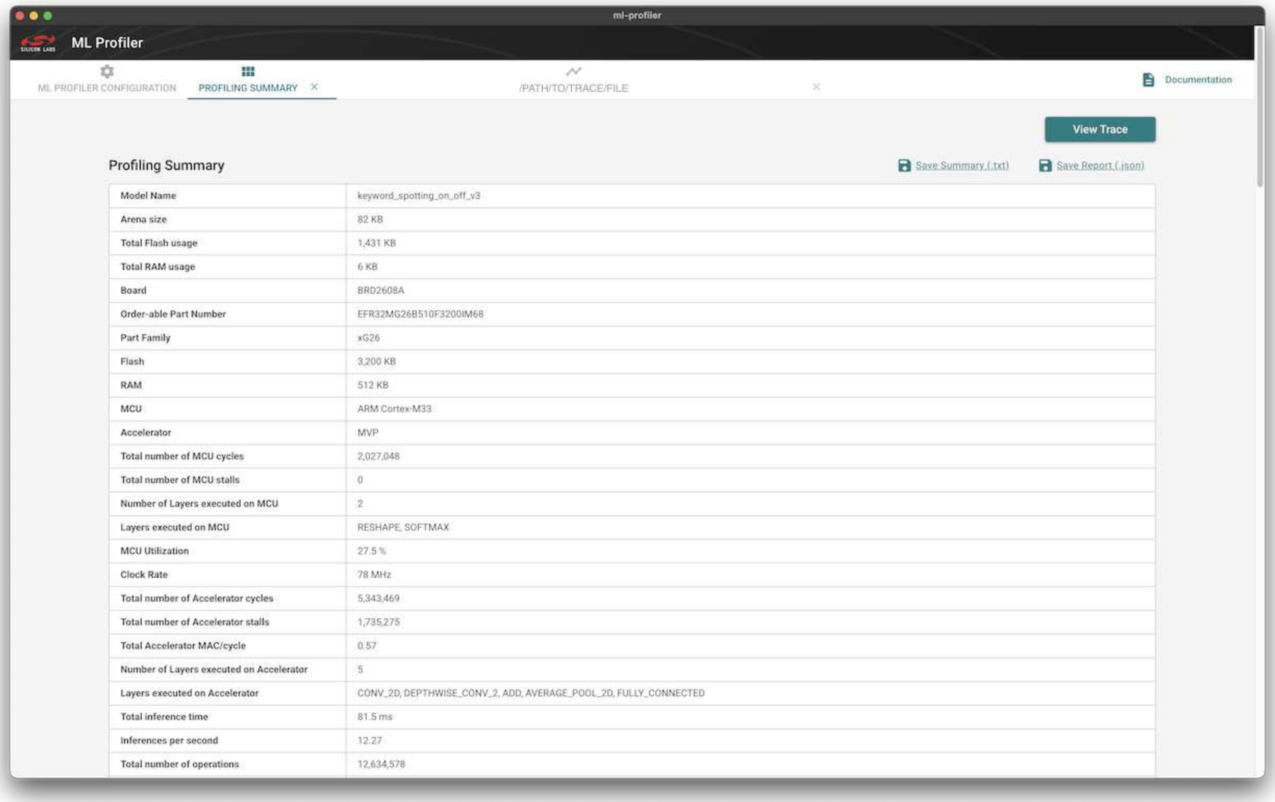
Profile

1. Find ML Model in File System
2. Path to ML Model populated
3. Auto-detected Board/Device
4. Click “Profile”
5. Profiling Session Executing

> *NOTE:* See Troubleshooting section for handling any errors.

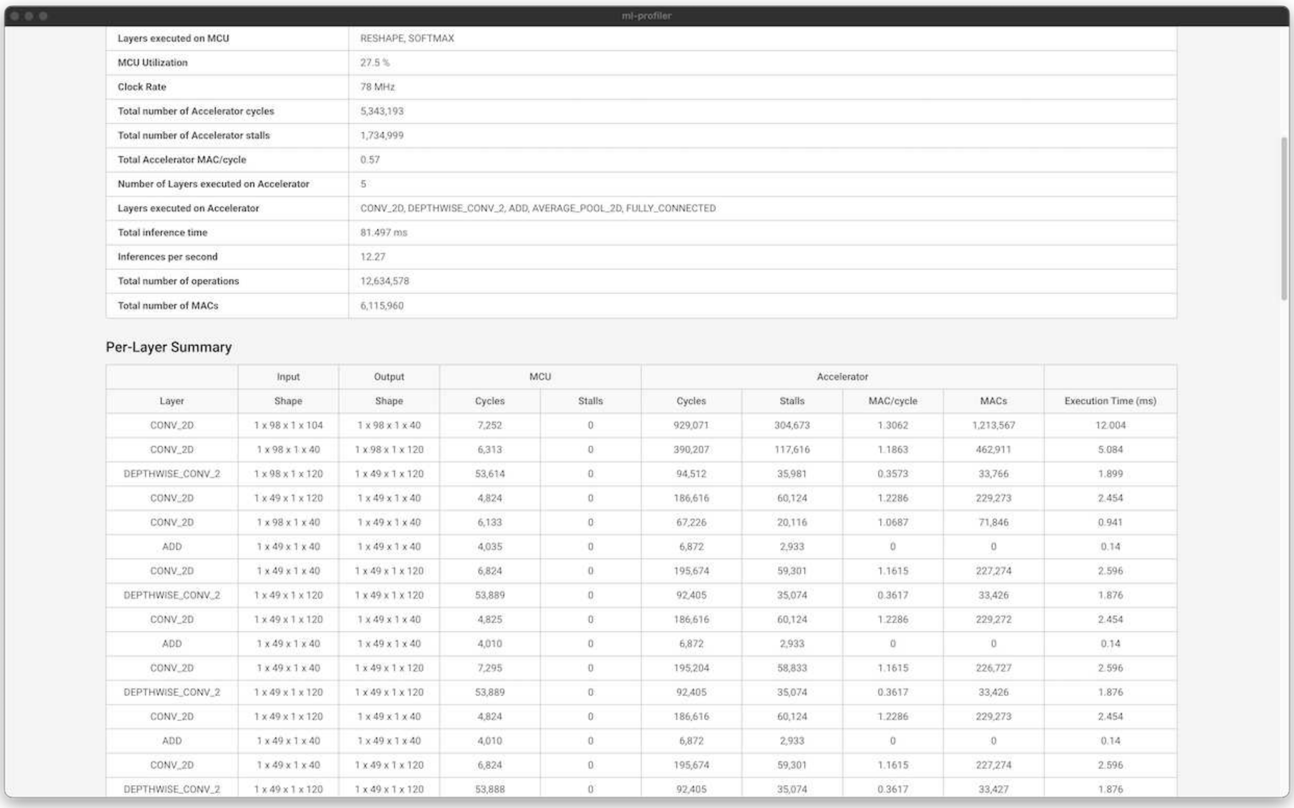
Outputs

- Summary tab: inference time, MCU vs MVP cycles, power



Profiling Summary

Model Name	keyword_spotting_on_off_v3
Arena size	82 KB
Total Flash usage	1,431 KB
Total RAM usage	6 KB
Board	BRD2608A
Order-able Part Number	EFR32MG26B510F3200IM68
Part Family	xG26
Flash	3,200 KB
RAM	512 KB
MCU	ARM Cortex-M33
Accelerator	MVP
Total number of MCU cycles	2,027,048
Total number of MCU stalls	0
Number of Layers executed on MCU	2
Layers executed on MCU	RESHAPE, SOFTMAX
MCU Utilization	27.5 %
Clock Rate	78 MHz
Total number of Accelerator cycles	5,343,469
Total number of Accelerator stalls	1,735,275
Total Accelerator MAC/cycle	0.57
Number of Layers executed on Accelerator	5
Layers executed on Accelerator	CONV_2D, DEPTHWISE_CONV_2, ADD, AVERAGE_POOL_2D, FULLY_CONNECTED
Total inference time	81.5 ms
Inferences per second	12.27
Total number of operations	12,634,578

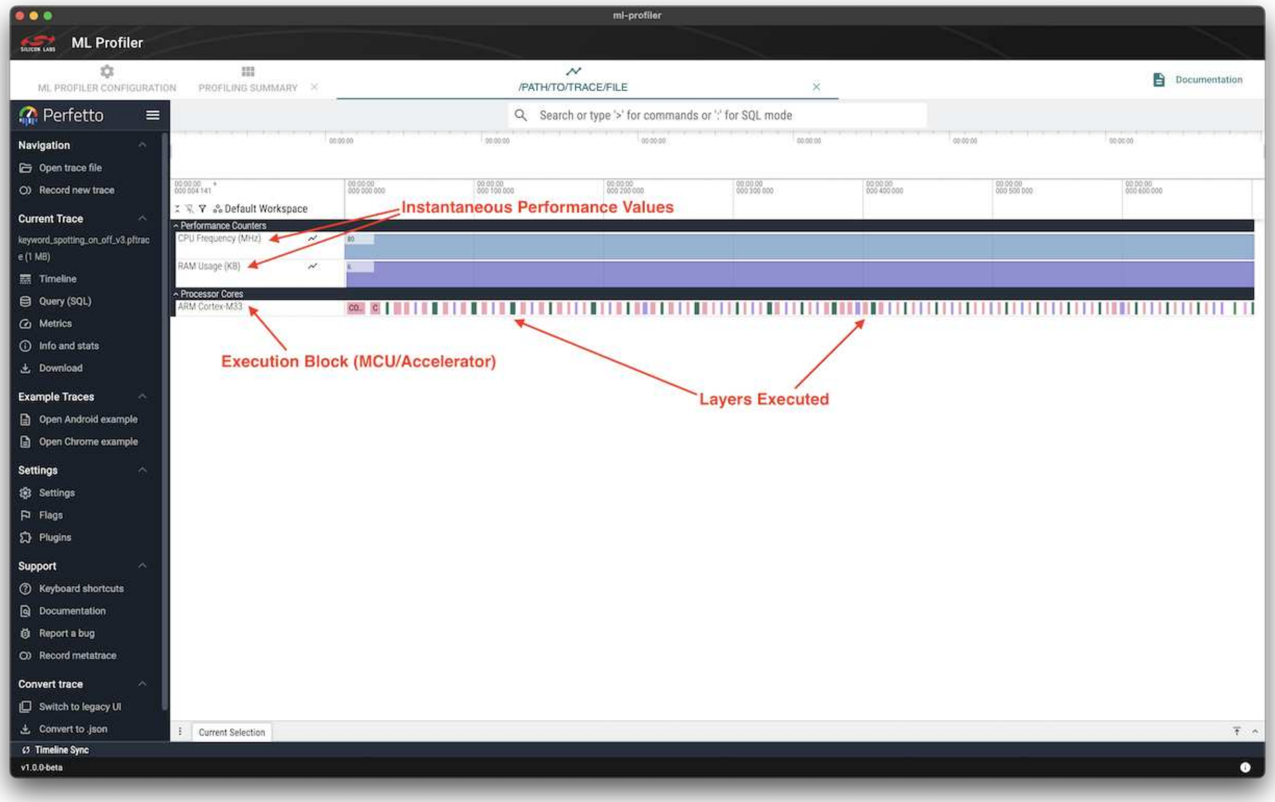

ml-profiler

Layers executed on MCU	RESHAPE, SOFTMAX
MCU Utilization	27.5 %
Clock Rate	78 MHz
Total number of Accelerator cycles	5,343,193
Total number of Accelerator stalls	1,734,999
Total Accelerator MAC/cycle	0.57
Number of Layers executed on Accelerator	5
Layers executed on Accelerator	CONV_2D, DEPTHWISE_CONV_2, ADD, AVERAGE_POOL_2D, FULLY_CONNECTED
Total inference time	81.497 ms
Inferences per second	12.27
Total number of operations	12,634,578
Total number of MACs	6,115,960

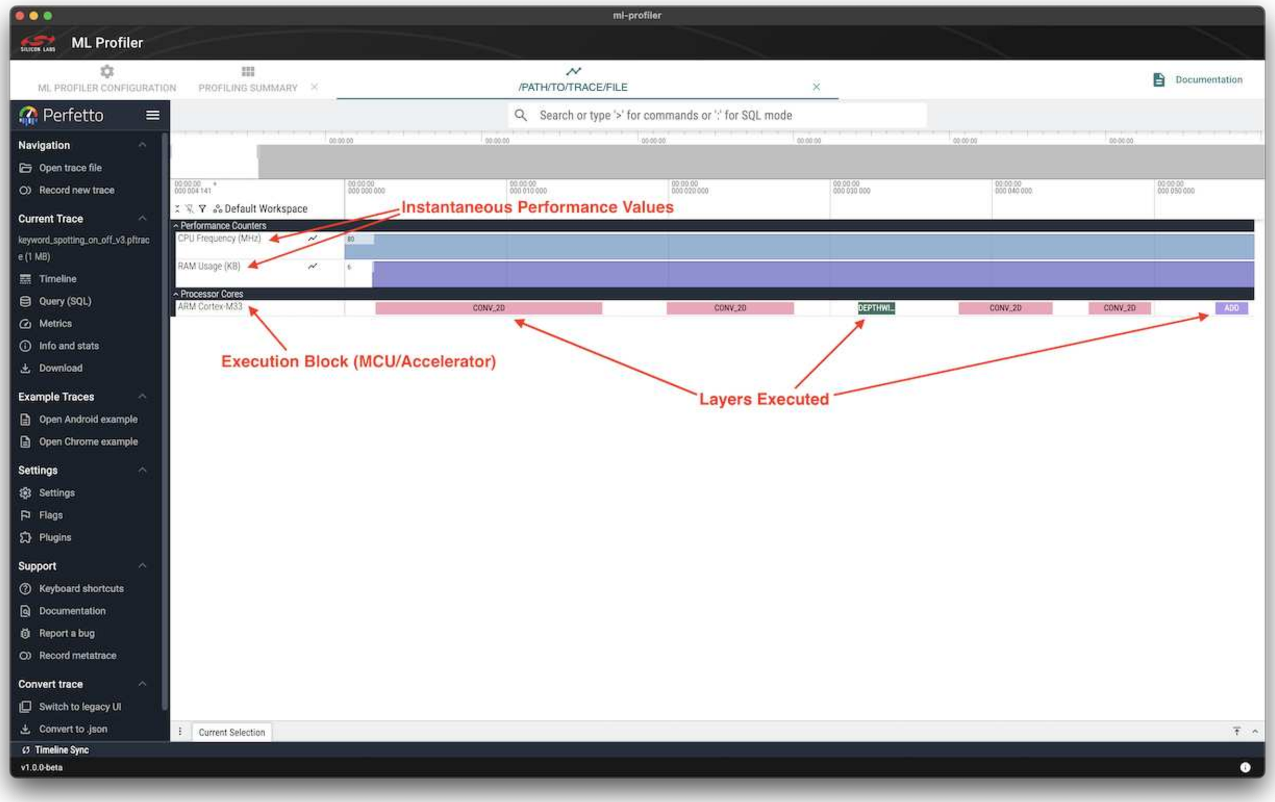
Per-Layer Summary

	Input	Output	MCU		Accelerator				
Layer	Shape	Shape	Cycles	Stalls	Cycles	Stalls	MAC/cycle	MACs	Execution Time (ms)
CONV_2D	1 x 98 x 1 x 104	1 x 98 x 1 x 40	7,252	0	929,071	304,673	1.3062	1,213,567	12.004
CONV_2D	1 x 98 x 1 x 40	1 x 98 x 1 x 120	6,313	0	390,207	117,616	1.1863	462,911	5.084
DEPTHWISE_CONV_2	1 x 98 x 1 x 120	1 x 49 x 1 x 120	53,614	0	94,512	35,981	0.3573	33,766	1.899
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,824	0	186,616	60,124	1.2286	229,273	2.454
CONV_2D	1 x 98 x 1 x 40	1 x 49 x 1 x 40	6,133	0	67,226	20,116	1.0687	71,846	0.941
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,035	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	6,824	0	195,674	59,301	1.1615	227,274	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,889	0	92,405	35,074	0.3617	33,426	1.876
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,825	0	186,616	60,124	1.2286	229,272	2.454
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,010	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	7,295	0	195,204	58,833	1.1615	226,727	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,889	0	92,405	35,074	0.3617	33,426	1.876
CONV_2D	1 x 49 x 1 x 120	1 x 49 x 1 x 40	4,824	0	186,616	60,124	1.2286	229,273	2.454
ADD	1 x 49 x 1 x 40	1 x 49 x 1 x 40	4,010	0	6,872	2,933	0	0	0.14
CONV_2D	1 x 49 x 1 x 40	1 x 49 x 1 x 120	6,824	0	195,674	59,301	1.1615	227,274	2.596
DEPTHWISE_CONV_2	1 x 49 x 1 x 120	1 x 49 x 1 x 120	53,888	0	92,405	35,074	0.3617	33,427	1.876

- Perfetto trace tab: time-based execution and resource traces



(Zoomed in view)



> *NOTE:* The profiler currently tracks only the ARM Cortex‑M MCU processor timeline. Usage and cycle information for the Matrix Vector Processor (MVP) is instead provided in the summary tab.

Running a Profiling Session from the CLI

1. Connect the board on which you want to profile your model. The board will be detected automatically, once connected.
2. Find the "device ID" of the connected board.
   1. Linux/macOS

      ```
      $ ~/.silabs/slt/installs/archive/sdm-darwin-arm64/sdm adapter list
      👉 Total adapter count: 1
      ↳ xxxxx [ usb wstk 440339411 xxxxx 127.0.0.1 ]
      ```

   2. Windows

      ```
      PS> $HOME\.silabs\slt\installs\archive\sdm-windows-amd64\sdm.exe adapter list
      👉 Total adapter count: 1
       ↳ xxxxx [ usb wstk 440339411 xxxxx 127.0.0.1 ]
      ```

   The device ID is "440339411".
3. Run Profiling

   ```
   sml profile /path/to/model_name.tflite 440339411
   ```

> *NOTE:* See Usage for more command line arguments. See Troubleshooting section for handling

> any errors.

Output

The following is an example of the output you can expect to see on the command line terminal. Usernames and other sensitive information has been stubbed.

The log below includes:

- Inference time
- MCU vs MVP cycle breakdown
- Power consumption (planned feature)

You can access:

- text summary
- detailed JSON report

```
🔍 Checking SDM server status...
✅ SDM server is connected and accessible

🔍 Checking if device exists in connected adapters...
✅ Device "440339411" found in connected adapters

🔍 Checking board support...
✅ Board "BRD2608A" is supported

🔍 Checking model file...
✅ Model file found: /path/to/model_name.tflite (534336 bytes)
🚀 Starting ML Profiler...

SDM Service connected

🚀 Starting profiling workflow...
   Device: 440339411 (BRD2608A)
   Model:  model_name

📡 Step 1: Connecting to debug channel...
   Terminal socket opened
   Debug channel connected, ready to capture packets...

⚡ Step 2: Combining model with firmware and flashing...
   Using firmware: aiml_soc_profiler_firmware_efr32_brd2608a_mvp.s37
   Combined: model_name_combined.s37 (2447 KB)
   Flashing to 440339411...
   Firmware flashed successfully

📦 Step 3: Capturing packets and generating trace...
   Captured 313 packets, building trace...
   Decoded 681 packets, generated 648 trace events


╔══════════════════════════════════════════════════════╗
║          ML PROFILER - PROFILING SUMMARY              ║
╚══════════════════════════════════════════════════════╝

SESSION SUMMARY
────────────────────────────────────────────────────────
  Model Name                           model_name
  Arena size                           82 KB
...more session summary...
  Board                                BRD2608A
...truncated for this documentation...
  Total number of MACs                 6,115,960

PER-LAYER SUMMARY
────────────────────────────────────────────────────────────────────────────────────────────────────────────
                |   Input          |   Output         |          MCU            |          Acc            |
----------------+------------------+------------------+------------------------+------------------------+---------
Layer           | Shape            | Shape            | Cycles     | Stalls     | Cycles     | Stalls     | Time(ms)
----------------+------------------+------------------+------------+------------+------------+------------+---------
CONV_2D         | 1 x 98 x 1 x 104 | 1 x 98 x 1 x 40  | 7,252      | 0          | 929,071    | 304,673    | 12.004
...more per-layer summary...
DEPTHWISE_CONV_2 | 1 x 98 x 1 x 120 | 1 x 49 x 1 x 120 | 53,617    | 0          | 94,539     | 36,008     | 1.899
...truncated for this documentation...
SOFTMAX         | 1 x 3            | 1 x 3            | 3,144      | 0          | 0          | 65         | 0.04
────────────────────────────────────────────────────────────────────────────────────────────────────────────


────────────────────────────────────────────────────────
Generated: YYYY-MM-DDTHH:MM:SS.SSSZ

📁 Profile data saved to:
   /path/to/profiling/data/model_name-YYYY-MM-DDTHH-MM-SSZ

   Includes:
```

```
• captured-packets.json (decoded packets)
  • report.json (profiling data)
  • summary.txt (readable summary)

  📄 See summary.txt for the complete profiling summary.

🌐 Step 4: Starting Perfetto UI server...
  Perfetto server started on port 10000

✅ Profiling completed successfully!

📊 Trace URL: http://localhost:10000#!/?
url=http%3A%2F%2Flocalhost%3A10000%2Ftrace%2FL1VzZXJzL3JhYW5zYXJpLy5zaWxhYnMvbWwtcHJvZmlsZXIva2V5d29yZF9zcG90dGlu

Use --gui to open GUI after profiling.
```

Press Ctrl+C to exit the profiling session.

Usage

```
sml --help
```

```
Usage: sml [options] [command]

Silicon Labs ML tooling.

Options:
  -v, --version                     Show version and exit.
  --gui                        Open GUI after command completes (if supported).
  --dry-run                       Validate and print the effective configuration, but do not
                             execute the command.
  -h, --help                      display help for command

Commands:
  profile [options] <model> <device>  Profile a machine learning model on a Silicon Labs device. Emits
                             a Perfetto-compatible trace (.pftrace) or JSON summary (.json).
  version                        Show the version number
  help [command]                     display help for command
```

Understanding Performance

The profiler presents a hierarchical execution view:

Inference → Layer → Operator → Kernel

Time-aligned tracks allow correlation between:

- MCU vs MVP execution
- memory usage
- clock rate
- power consumption

Idle time per kernel helps identify when:

- accelerator overhead dominates
- memory stalls occur
- MCU execution may be more efficient

Limitations

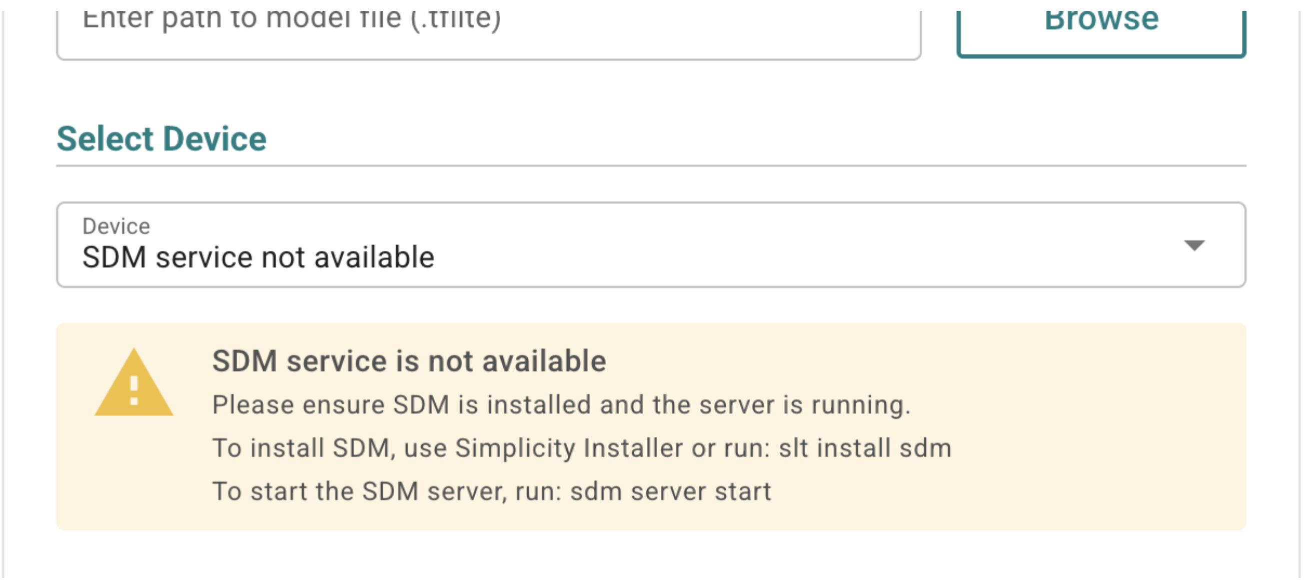
- Requires real Silicon Labs hardware, currently only supports xG24, xG26, and xG28 devices, specifically BRD2601B, BRD2608A, and BRD2705A dev kits.
- Simulator support is in development
- Does not auto-compare MCU vs MVP
- Does not measure model accuracy. This is not a target of this tool. It is geared exclusively towards execution performance analysis.

Summary

The Silicon Labs Machine Learning Model Profiler helps engineers reason about embedded ML performance by making execution behavior visible, comparable, and intuitive.

Troubleshooting

"SDM Service is not available" warning



SDM service not available

SDM service is not available
Please ensure SDM is installed and the server is running.
To install SDM, use Simplicity Installer or run: slt install sdm
To start the SDM server, run: sdm server start

Solution

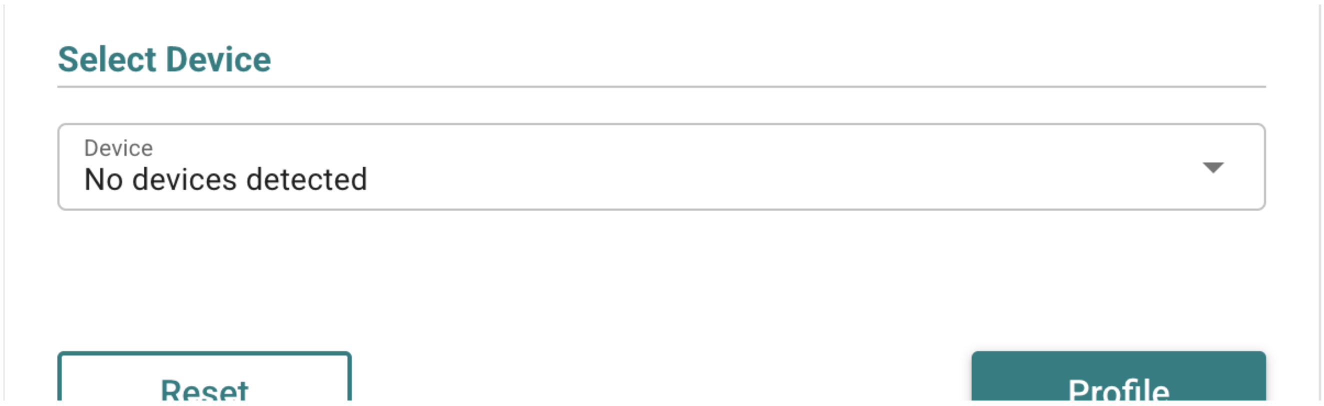
1. Verify if Simplicity Device Manager (SDM) is installed using, `slt locate sdm`.
2. If you see no output on the console, install SDM using:
    1. `slt install sdm` through the CLI, or
    2. Simplicity Installer by following the steps mentioned in the Install using Simplicity Installer section. Search for "Simplicity Device Manager" instead of "ML Profiler".
3. Start SDM server.
    1. Linux/macOS

    ```
    ~/.silabs/slt/installs/archive/sdm-darwin-arm64/sdm server start
    ```

    2. Windows

```
PS> $HOME\.silabs\slt\installs\archive\sdm-windows-amd64\sdm.exe server start
```

"No devices connected" message in the "Select Device" field



Solution

Connect the desired board on which you want to profile your models.

Any type of "Firmware preparation/flashing failed: Failed to combine model with firmware" error

Examples of this type of error:

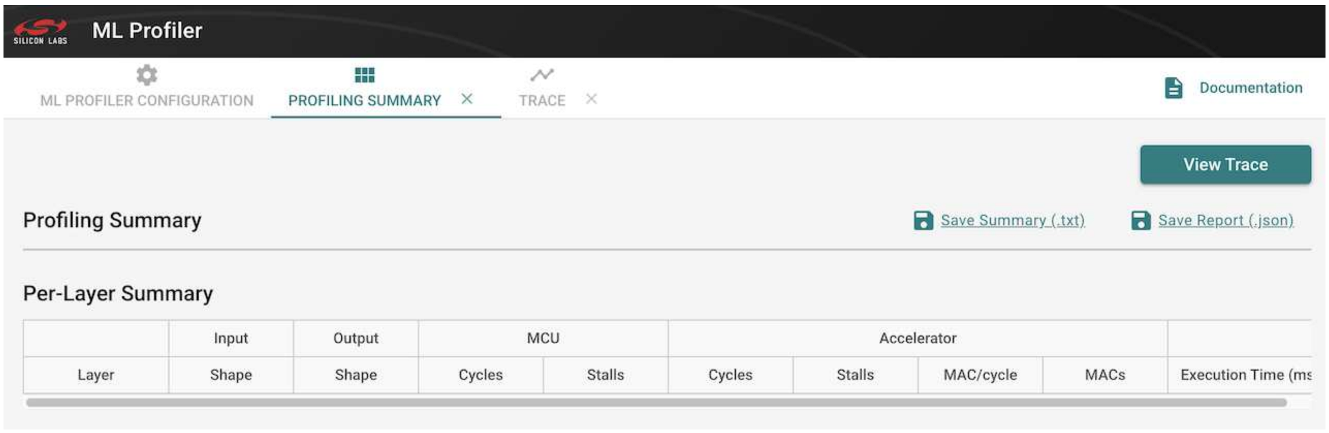
1. Firmware preparation/flashing failed: Failed to combine model with firmware: 404 Not Found: Not Found
2. Firmware preparation/flashing failed: Failed to combine model with firmware: Combine binary job failed: Error: Could not find function simpleCombineConvertBinaries. It is either typed wrong, you miss an adapter pack, or you need to upgrade one.

Solution

1. This issue is most commonly caused due to an older version of either Simplicity Device Manager or Simplicity Commander.
2. Update Simplicity Commander to v1.22+. Install using `slt install commander`. Verify using `slt locate commander`.
3. Update Simplicity Device Manager to v0.101.4+. Install using `slt install sdm`. Verify using `slt locate sdm`.

No Profiling Output

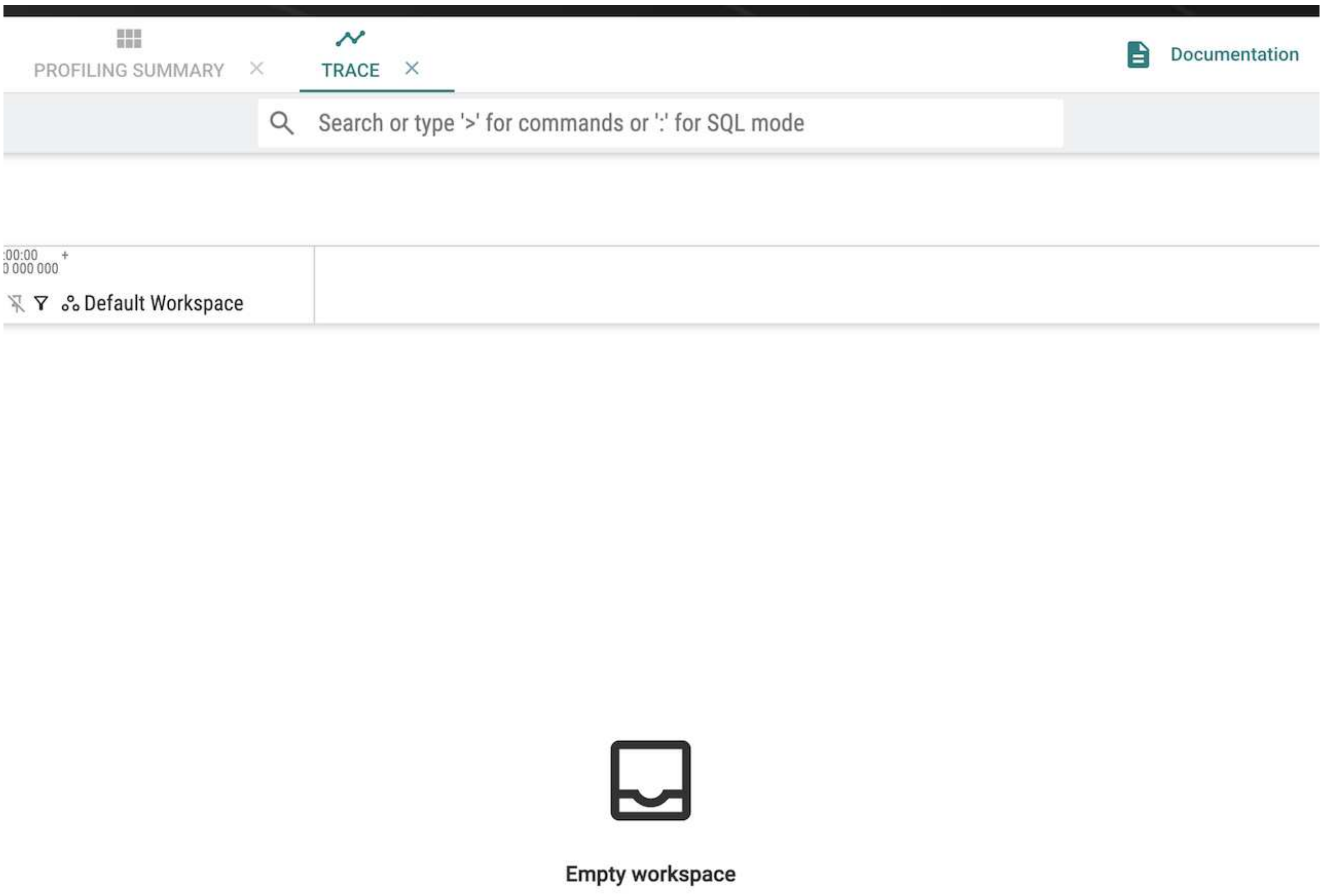
GUI



Profiling Summary

Per-Layer Summary

	Input	Output	MCU		Accelerator				
Layer	Shape	Shape	Cycles	Stalls	Cycles	Stalls	MAC/cycle	MACs	Execution Time (ms



CLI

```
📦 Step 3: Capturing packets and generating trace...
  Captured 43 packets, building trace...
  Decoded 0 packets, generated 0 trace events

╔══════════════════════════════════════════════════════════╗
║       ML PROFILER - PROFILING SUMMARY                ║
╚══════════════════════════════════════════════════════════╝

──────────────────────────────────────────────────────────
```

Solution

1. Verify Simplicity Device Manager Service is up: `sdm server status`. If not, invoke `sdm server start`.
2. Find the device ID, see Running a Profiling Session From the CLI section.
3. Jump into admin console of your device: `sdm terminal -a <device_id> -c admin`.

4. Verify the debug message version: `dch message version` , If the output is `Message protocol version : 3` , use Step 5 below.
5. Invoke `dch message version 2` . The output must show `Current version = 2` .
6. Execute a Profiling session again.