IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
AUSTIN DIVISION

LUCIO DEVELOPMENT LLC,	§	
	§	
Plaintiff,	§	Case No: 1:19-cv-077
	§	
vs.	§	PATENT CASE
	§	
SILICON LABORATORIES INC.	§	
	§	
Defendant.	§	
	§	

COMPLAINT

Plaintiff Lucio Development LLC ("Plaintiff" or "Lucio") files this Complaint against Silicon Laboratories Inc. ("Defendant" or "SLI") for infringement of United States Patent No. 7,069,546 (hereinafter "the '546 Patent").

PARTIES AND JURISDICTION

1. This is an action for patent infringement under Title 35 of the United States Code. Plaintiff is seeking injunctive relief as well as damages.

2. Jurisdiction is proper in this Court pursuant to 28 U.S.C. §§ 1331 (Federal Question) and 1338(a) (Patents) because this is a civil action for patent infringement arising under the United States patent statutes.

3. Plaintiff is a Texas limited liability company with its office address at 555 Republic Dr., Suite 200, Plano, Texas 75074.

4. On information and belief, Defendant is a Delaware corporation with a principal place of business at 400 W. Cesar Chavez, Austin, TX 78701. On information and belief, Defendant may be served through its registered agent, The Corporation Trust Company, at 1209 N. Orange St., Wilmington, DE 19801.

5. This Court has personal jurisdiction over Defendant because Defendant has committed, and continues to commit, acts of infringement in this District, has conducted business in this District, and/or has engaged in continuous and systematic activities in this District.

6. On information and belief, Defendant's instrumentalities that are alleged herein to infringe were and continue to be used, imported, offered for sale, and/or sold in this District.

VENUE

7. Venue is proper in this District pursuant to 28 U.S.C. §1400(b) because acts of infringement are occurring in this District and Defendant has a regular and established place of business in this District. For instance, on information and belief, Defendant has a regular and established place of business at 400 W. Cesar Chavez, Austin, TX 78701.

COUNT I
(INFRINGEMENT OF UNITED STATES PATENT NO. 7,069,546)

8. Plaintiff incorporates paragraphs 1 through 7 herein by reference.

9. This cause of action arises under the patent laws of the United States and, in particular, under 35 U.S.C. §§ 271, *et seq*.

10. Plaintiff is the owner by assignment of the '546 Patent with sole rights to enforce the '546 Patent and sue infringers.

11. A copy of the '546 Patent, titled "Generic Framework for Embedded Software Development," is attached hereto as Exhibit A.

12. The '546 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code.

13. On information and belief, Defendant has infringed and continues to infringe one or more claims, including at least Claim 1, of the '546 Patent by making, using, importing,

selling, and/or offering for sale a software platform for embedded software development, which is covered by at least Claim 1 of the '546 Patent. Defendant has infringed and continues to infringe the '546 Patent directly in violation of 35 U.S.C. § 271.

14. Defendant, sells, offers to sell, and/or uses embedded software development packages including, without limitation, Simplicity Studio 4, and any similar products ("Product"), which infringe at least Claim 1 of the '546 Patent.

15. Simplicity Studio is an Integrated Development Environment (IDE) by Silicon Labs and comprises embedded software to build applications, evaluate, configure and develop with EFM32, EFM8 and 8051 Microcontroller Units (MCUs), Wireless Geckos and MCUs, Zigbee System on chips (SOCs) and wireless modules. Simplicity Studio provides one or more generic application handler programs, each such program comprising computer program code for performing generic application functions common to multiple types of hardware modules used in a communication system. For example, Simplicity Studio provides a Hardware Abstraction Layer (HAL) containing programs, functions and data structures which are common and uniform across all supported devices (EmberZNet PRO, EmberZNet RF4CE and Silicon Labs Thread). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

SILICON LABS

AN0822: Simplicity Studio™ User's Guide

Simplicity Studio greatly reduces development time and complexity with Silicon Labs' EFM32, EFM8, and 8051 MCUs, wireless MCUs, and Zigbee SoCs.

Simplicity Studio can create wireless applications and provides hardware configuration, network analysis, real-time energy debugging, a high-powered IDE, and links to helpful resources, all in one place.

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

KEY POINTS

- Simplicity Studio makes the development process easier, faster, and more efficient.
- The IDE and integrated tools help optimize designs.
- Getting started with development is quick and easy with Demos and Software Examples.
- Quickly find help and design resources.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 1.

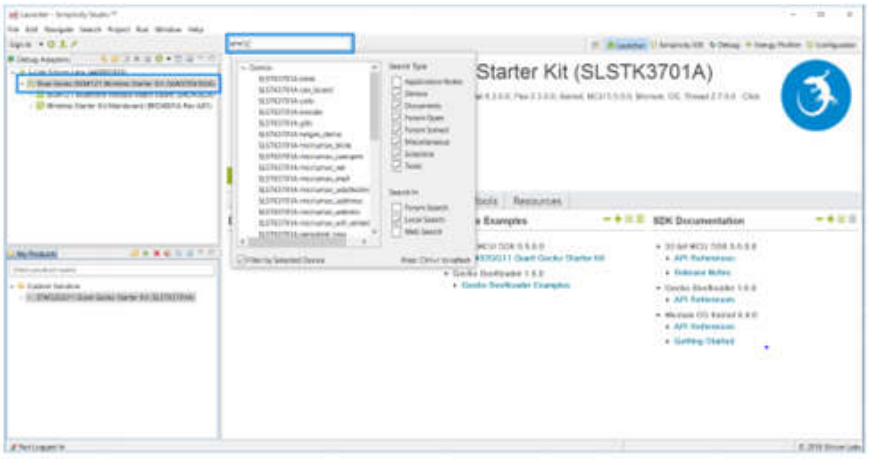
2. Overview

Simplicity Studio is a launching pad for everything needed to evaluate, configure, and develop with EFM32, EFM8, and 8051 MCUs, Wireless Geckos and MCUs, Zigbee SoCs, and Wireless Modules. The software gathers all of these tools into four categories: Getting Started, Documentation, Compatible Tools, and Resources.

The contents of these sections are contextualized for the selected device or kit so that only the relevant information and tools are shown.

2.1 Part Selection

Selecting a device will change the available tiles and the behavior of each tile in the launcher. The device can be selected by connecting a kit or board or by adding a part to the [**My Products**] window. Examples and documentation can also be found by searching for the device or kit.



Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 3.

3. Simplicity IDE

The Simplicity IDE is an Eclipse-based Integrated Development Environment (IDE) enabling code editing, downloading, and debugging for Wireless (including EM35xx), Wireless Geckos, Wireless Modules, EFM32, EFM8, and 8051 devices.

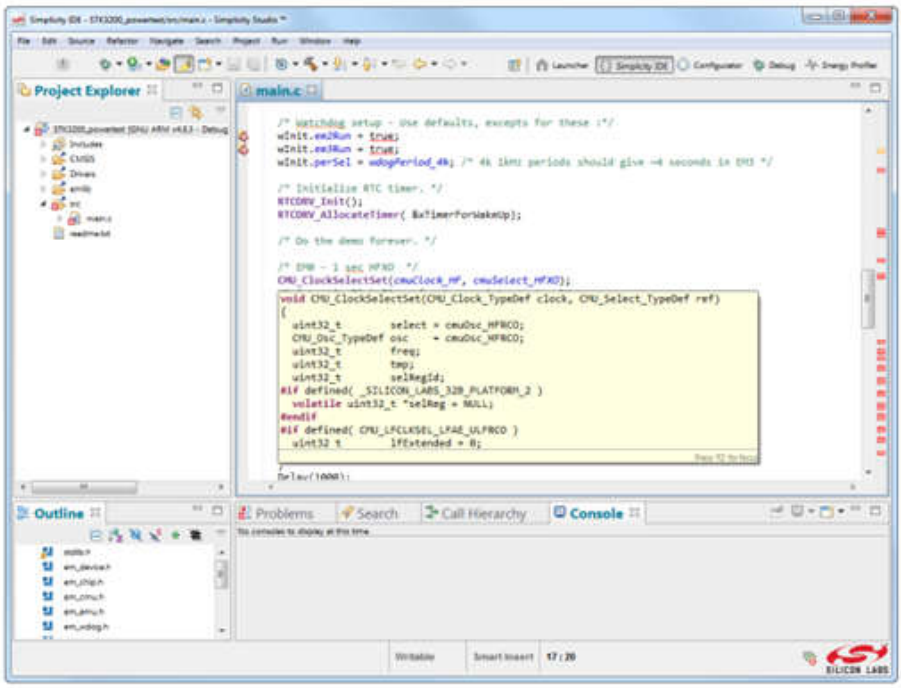


Figure 3.1. Simplicity IDE

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 8.

3.1 Projects

A Simplicity IDE project contains files, build options, and project settings. Projects generally exist as a directory containing sub-directories and files. The project structure seen in the IDE in the [**Project Explorer**] view is replicated physically on the disk. However, a project may also contain linked files or directories which are just pointers to files or folders outside of the project directory.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf ,page8

SILICON LABS

UG103.4: HAL Fundamentals

Silicon Labs HAL (Hardware Abstraction Layer) is program code between a system's hardware and its software that provides a consistent interface for applications that can run on several different hardware platforms. The HAL is designed for developers using EmberZNet PRO, EmberZNet RF4CE, and Silicon Labs Thread. This document contains new information for the Mighty Gecko (EFR32MG) family and updates for the EM3x-based MCU family of products.

KEY POINTS

- Provides information on how the HAL is organized, including naming conventions, API files, and directory structure.
- Includes an overview of each of the main subsections of the HAL functionality.
- Describes how to adapt the HAL to your specific hardware and application requirements.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 1.

3. Naming Conventions

HAL function names have the following prefix conventions:

- `hal`: The API Sample applications use. You can remove or change the implementations of these functions as needed.
- `halCommon`: The API used by the stack and that can also be called from an application. Custom HAL modifications must maintain the functionality of these functions.
- `halStack`: Only the stack uses this API. These functions should not be directly called from any application, as this may violate timing constraints or cause re-entrancy problems. Custom HAL modifications must maintain the functionality of these functions.
- `halInternal`: The API that is internal to the HAL. These functions are not called directly from the stack and should not be called directly from any application. They are called only from `halStack` or `halCommon` functions. You can modify these functions, but be careful to maintain the proper functionality of any dependent `halStack` or `halCommon` functions.

Most applications will call `halXXX` and `halCommonXXX` functions and will not need to modify the HAL. If you need a special implementation or modification of the HAL, be sure to read the rest of this document as well as the data sheet for your Silicon Labs platform first.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 3.

4. API Files and Directory Structure

The HAL directory structure and files are organized to facilitate independent modification of the compiler, the MCU, and the PCB configuration.

- `<hal>/hal.h`: This master include file comprises all other relevant HAL include files, and you should include it in any source file that uses the HAL functionality. Most programs should not include lower-level includes, and instead should include this top-level `hal.h`.
- `<hal>/ember-configuration.c`: This file defines the storage for compile-time configurable stack variables and implements default implementations of functions. You can customize many of these functions by defining a preprocessor variable at compile-time and implementing a custom version of the function in the application. (For more information, see `ember-configuration-defaults.h` in the API Reference for your software).
- `<hal>/micro/generic`: This directory contains files used for general MCUs on POSIX-compliant systems. The default compiler is GCC.

4.1 HAL Implementation for ARM Cortex-M3 SOC Platforms

There is a HAL implementation for the ARM Cortext-M3 System on Chip (SOC) platforms as follows:

- `<hal>/micro/cortexm3`: This directory contains the implementation of the HAL for the cortexm3, which is the processor core used by the EM3x and EFR32 platforms. Functions in this directory are specific to the cortexm3 but are not specific to a particular microcontroller family or variant (see the next entry).
- `<hal>/micro/cortexm3/{mcu_family}`: This directory implements functions that are specific to the a particular MCU family, such as `<hal>/micro/cortexm3/em35x` for the EM3x-based MCU family, including variants such as the EM357, EM3588, and EM346; or `<hal>/micro/cortexm3/efm32` for the EFR32MG.
- `<hal>/micro/cortexm3/bootloader`: This directory implements functions that pertain to the on-chip bootloaders used on Cortex M3-based platforms to facilitate runtime loading/updating of applications. (More MCU-specific files can be found in `<hal>/micro/cortexm3/{mcu_family}/bootloader`.)
- `<hal>/micro/cortexm3/{mcu_family}/board`: This directory contains header files that define the peripheral configuration and other PCB-level settings, such as initialization functions. These are used in the HAL implementations to provide the correct configurations for different PCBs.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 4.

Table 6.1. HAL Source Modules

Source Module	Description
Adc	Sample functionality for accessing analog-to-digital converters built into the SoC (refer to document *AN715: Using the EM35x ADC* for additional information).
bootloader-interface-app	APIs for using the application bootloader.
bootloader-interface-standalone	APIs for using the standalone bootloader.
Button	Sample functionality that can be used to access the buttons built into the development kit carrier boards.
buzzer	Sample functionality that can play notes and short tunes on the buzzer built into the development kit carrier boards.
crc	APIs that can be used to calculate a standard 16-bit CRC or a 16-bit CCITT CRC as used by 802.15.4
diagnostic	Sample functionality that can be used to help diagnose unknown watchdog resets and other unexpected behavior.
Flash	Internal HAL utilities used to read, erase, and write Flash in the SoC.
Led	Sample functionality that can be used to manipulate LEDs.
mem-util	Common memory manipulation APIs such as memcopy.
Micro	Core HAL functionality to initialize, put to sleep, shutdown, and reboot the microcontroller and any associated peripherals.
Random	APIs that implement a simple pseudo-random number generator that is seeded with a true-random number when the stack is initialized.
sim-eeprom	Simulated EEPROM system for storage of tokens in the SoC.
Spi	APIs that are used to access the SPI peripherals.
symbol-timer	APIs that implement the highly accurate symbol timer required by the stack.
system-timer	APIs that implement the basic millisecond time base used by the stack.
Token	APIs to access and manipulate persistent data used by the stack and many applications.
Uart	Low-level sample APIs used by the serial utility APIs to provide serial input and output.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 10.

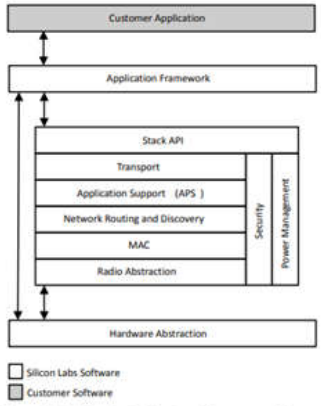


Figure 2.1. Silicon Labs Stack and Customer Software Interaction

The network stack software is a collection of libraries, source code, tools, sample applications, and product documentation. The network stack API is documented in an online API reference as well as other documents installed with the stack installer, or available through the development environment. The network stack is delivered as a collection of libraries that you can link to your applications. A description of each library is provided in the development environment.

Source: https://www.silabs.com/documents/public/user-guides/UG103-08-AppDevFundamentals-Tools.pdf ,page2

5.3 Peripheral Access

The networking stack requires access to certain on-chip peripherals; additionally, applications may use other on-chip or on-board peripherals. The default HAL provides implementations for all required peripherals and also for some commonly used peripherals. Silicon Labs recommends that developers implement additional peripheral control within the HAL framework to facilitate easy porting and upgrade of the stack in the future.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf ,page5

Source: https://www.silabs.com/documents/public/user-guides/UG103-08-AppDevFundamentals-Tools.pdf, page 2.

EFM32™ Zero Gecko Starter Kit

EFM32ZG-STK3200

Silicon Labs' EFM32ZG-STK3200 provides an excellent tool to get familiar with the EFM32™ Zero Gecko 32-bit microcontrollers (MCUs), the industry's most energy-friendly MCU based on the ARM® Cortex®-M0+ processor.

Supported by Simplicity Studio, the starter kit contains sensors and peripherals demonstrating some of the MCU's many capabilities and can serve as a starting point for application development. The EFM32ZG-STK3200 features an on-board SEGGER J-Link debugger and an Advanced Energy Monitoring system, allowing you to program, debug and perform real-time current profiling of your application without using external tools.

Source: https://www.silabs.com/products/development-tools/mcu/32-bit/efm32-zero-gecko-starter-kit

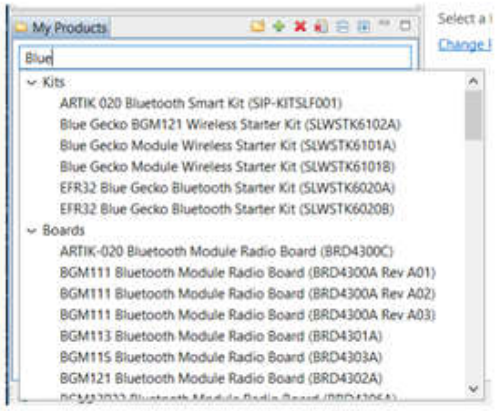


Figure 2.2. Adding to My Products

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 4.

16. Simplicity Studio generates specific application handler code to associate the generic application functions with specific functions of a device driver for at least one of the types of the hardware modules. For example, in addition to the generic drivers and HAL, Simplicity Studio also includes specific application handler code that is specific to the application (such as communication, general purpose, etc.) and specific to particular hardware (such as Bluetooth module radio board EmberZNet PRO, EmberZNet RF4CE and Silicon Labs Thread and other various components). Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

3.5 Code Editing Features

Simplicity IDE is a code editing and development environment. The editor includes context highlighting, reference searching, and standard features found in any modern editor.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf , page 10

Open Declaration

In addition to the basic features, Simplicity IDE supports many advanced code-editing features. For example, the IDE automatically indexes all code within the project to support symbol lookup. The code does not have to build completely for the indexer to work, though certain features may not be available if, for example, the main() routine is not declared.

In this example, the **[Open Declaration]** (F3 shortcut) feature quickly finds symbol declarations.

1. Open the file of interest by double-clicking it in the **[Project Explorer]**.
2. Right-click on the desired symbol to display the context menu.
3. Click **[Open Declaration]** to quickly navigate to the definition of the symbol (e.g., **[TMR2CN_TF2H]**).
4. Studio will automatically open the file and highlight the line containing the declaration of the symbol.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 11.

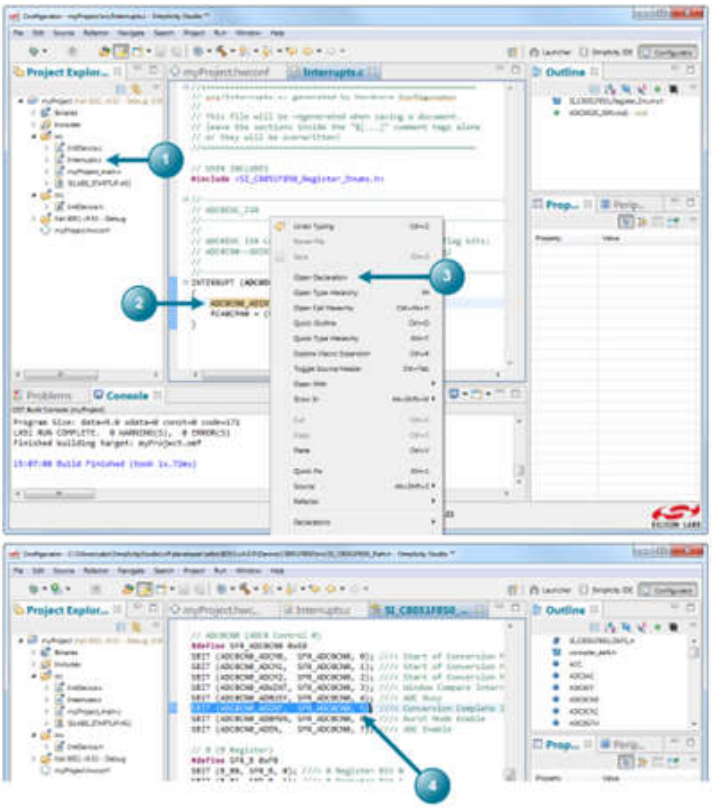


Figure 3.2. Finding Symbol Declarations

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 11.

Link with Editor

The **[Link with Editor]** button synchronizes the editor with the **[Project Explorer]** view, highlighting the file currently selected in the editor.

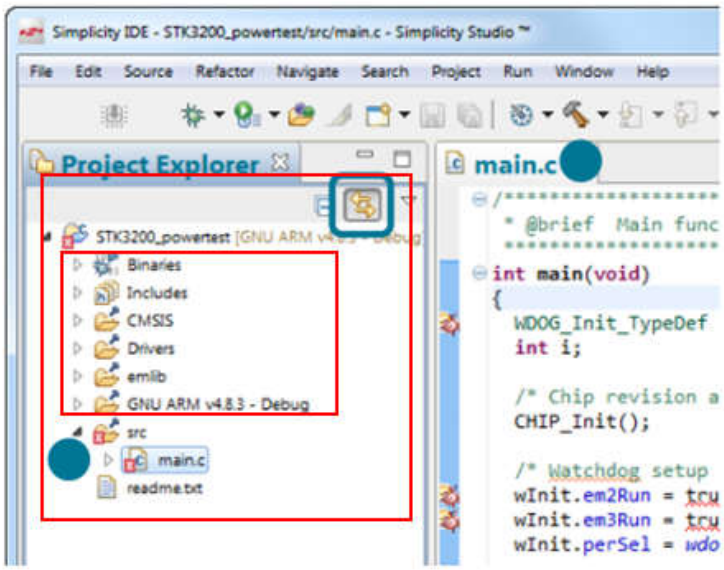


Figure 3.4. Link with Editor

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 13.

17. Simplicity Studio generates specific application handler code and defines a specific element in the specific code to be handled by one of the generic application functions for that hardware module. For example, Simplicity Studio generates system-specific application handler code by defining specific elements such as functions and data structures corresponding to specific hardware components (such as Bluetooth module radio board EmberZNet PRO, EmberZNet RF4CE and Silicon Labs Thread and other various components) that extend or otherwise connect the system-specific application handler code to the functions and data structures defined and made available by the HAL. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

Content Assist

Simplicity Studio also supports code completion, a feature called [**Content Assist**]. Content Assist requires that the appropriate header files be included in the file so that the symbols are available. To use [**Content Assist**], type the first few letters of a symbol or include file and press [**Ctrl+Space**] to display a list of symbols that match. For example, to use [**Content Assist**] to display a list of symbols starting with the characters [**P1**]:

1. Open a file and type the desired characters in the file (e.g., [**P1**]).
2. Press [**Ctrl+Space**] to display the [**Content Assist**] list.
3. Use the arrow keys or page up and down keys to look through the list of matching symbols.
4. Pressing [**Enter**] will replace the typed characters with the selected symbol.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page 12.

5. Peripheral Drivers

Embedded source C code is provided for drivers of peripherals such as the serial controller and analog-to-digital converter (ADC). These drivers let you incorporate standard functionality into custom applications. For more information on these drivers, see the stack API reference for your platform.

Source: https://www.silabs.com/documents/public/user-guides/UG103-08-AppDevFundamentals-Tools.pdf, page 5.

6.2 Custom PCBs

Creating a custom GPIO configuration for your target hardware is most easily done by using the Simplicity Studio IDE to generate a board header file based on a copy of an existing board header file and then editing the generated file to match the configuration of the custom board. The board header file includes definitions for all the pinouts of external peripherals used by the HAL as well as macros to initialize and power up and down these peripherals. The board header is identified through the BOARD_HEADER preprocessor definition specified at compile time. Board header files generated by the IDE will typically have a `_board.h` suffix, but several pre-made board header files for specific reference boards are available in the `hal/micro/cortexm3/{mcu_family}/board` folder of your stack release.

Modify the port names and pin numbers (and potentially location numbers in the case of EFR32) used for peripheral connections as appropriate for the custom board hardware. These definitions can usually be easily determined by referring to the board's schematic.

Once the new file is complete, change the preprocessor definition BOARD_HEADER for this project to refer to the new filename.

In addition to the pinout modification, functional macros are defined within the board header file and are used to initialize, power up, and power down any board-specific peripherals. The macros are:

- `halInternalInitBoard`
- `halInternalPowerDownBoard`
- `halInternalPowerUpBoard`

Within each macro, you can call the appropriate helper halInternal APIs or, if the functionality is simple enough, insert the code directly.

Certain modifications might require you to change additional source files in addition to the board header. Situations that might require this include:

- Using different external interrupts or interrupt vectors
- Functionality that spans multiple physical IO ports
- Changing the core peripheral used for the functionality (for example, using a different timer or SPI peripheral)

In these cases, refer to section 6.3 Modifying the Default Implementation.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 9.

9.1.4 Generating a Project

After making selections for any portions of the application the developer wishes to customize, AppBuilder will generate a software project with customized header files and array definitions in C code to represent the desired application behavior. The resulting project can be built from within Simplicity IDE to produce a binary that can be loaded onto the target wireless SoC.

This project generation is specific to the stack version and target chip where the configuration is done, but the Silicon Labs Application Framework abstracts many of these differences within its APIs and state machines, so AppBuilder configurations can be reused across different stack versions or chip platforms. This makes it much easier to transition from one version of a stack release to the next.

Source: https://www.silabs.com/documents/public/application-notes/AN0822-simplicity-studio-user-guide.pdf, page29.

2. Stack Software

The network stack software is a collection of libraries, source code, tools, sample applications, and product documentation. The network stack API is documented in an online API reference as well as other documents installed with the stack installer, or available through the development environment. The network stack is delivered as a collection of libraries that you can link to your applications. A description of each library is provided in the development environment.

Source: https://www.silabs.com/documents/public/user-guides/UG103-08-AppDevFundamentals-Tools.pdf, page 2.

18. Simplicity Studio compiles the generic application handler programs together with the specific application handler code to produce machine-readable code to be executed by an embedded processor in the at least one of the types of the hardware modules. For example, when a specific application is needed for a particular hardware, the generic functions and the specific functions are compiled together to yield a machine readable code. Silicon Labs and/or its customers compile the generic functions and the specific functions using Simplicity Studio and/or any other compiling IDE supported by Silicon Labs. Certain elements of this limitation are illustrated in the screenshots below and in the screenshots referenced in connection with other elements herein.

6. Customizing the HAL

This section describes how to adapt the Silicon Labs-supplied standard HAL to your specific hardware and application requirements.

6.1 Compile-Time Configuration

The following preprocessor definitions are used to configure the networking stack HAL. They are usually defined in the Project file, but depending on the compiler configuration they may be defined in any global preprocessor location.

6.1.1 Required Definitions

The following preprocessor definitions must be defined:

- `PLATFORM_HEADER`: The location of the platform header file. For example, the EM3588 uses `hal/micro/cortexm3/compiler/iar.h`.
- `BOARD_HEADER`: The location of the board header file. For example, the EM3588 developer board uses `hal/micro/cortexm3/em35x/board/dev0680etm.h`. Custom boards should change this value to the new file name.
- `PLATFORMNAME` (for example, CORTEXM3).
- `PLATFORMNAME_MICRONAME` (for example, CORTEXM3_EM3588).
- `PHY_PHYNAME` (for example PHY_EFR32).
- `BOARD_BOARDNAME` (for example, BOARD_BRD4151A or BOARD_DEV0680).
- `CONFIGURATION_HEADER`: Provides additional custom configuration options for `ember-configuration.c`.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page8.

- Compiler toolchain
  - GCC (The GNU Compiler Collection) is provided with Simplicity Studio. GCC is used in this document.

    Note: Application images created with GCC are larger than those created with IAR. If you use GCC to compile the example applications in this SDK, you must use a part with at least 512 kB of flash.
  - IAR Embedded Workbench for ARM (IAR-EWARM) (optional)

    Note: See the Release Notes for the IAR version supported by this version of the Silicon Labs Thread SDK. Download the supported version from the Silicon Labs Support Portal, as described at the end of section 2.3 Install Simplicity Studio and the Silicon Labs Thread Stack. Refer to the "QuickStart Installation Information" section of the IAR installer for additional information about the installation process and how to configure your license. Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM.

Source: https://www.silabs.com/documents/public/quick-start-guides/qsg113-efr32-thread.pdf ,page1

4. API Files and Directory Structure

The HAL directory structure and files are organized to facilitate independent modification of the compiler, the MCU, and the PCB configuration.

- `<hal>/hal.h`: This master include file comprises all other relevant HAL include files, and you should include it in any source file that uses the HAL functionality. Most programs should not include lower-level includes, and instead should include this top-level `hal.h`.
- `<hal>/ember-configuration.c`: This file defines the storage for compile-time configurable stack variables and implements default implementations of functions. You can customize many of these functions by defining a preprocessor variable at compile-time and implementing a custom version of the function in the application. (For more information, see `ember-configuration-defaults.h` in the API Reference for your software).
- `<hal>/micro/generic`: This directory contains files used for general MCUs on POSIX-compliant systems. The default compiler is GCC.

Source: https://www.silabs.com/documents/public/user-guides/UG103-04-AppDevFundamentals-HAL.pdf, page 4.

19. Defendant's actions complained of herein will continue unless Defendant is enjoined by this court.

20. Defendant's actions complained of herein are causing irreparable harm and monetary damage to Plaintiff and will continue to do so unless and until Defendant is enjoined

and restrained by this Court.

21. Plaintiff is in compliance with 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, Plaintiff asks the Court to:

(a) Enter judgment for Plaintiff on this Complaint on all causes of action asserted herein;

(b) Enter an Order enjoining Defendant, its agents, officers, servants, employees, attorneys, and all persons in active concert or participation with Defendant who receive notice of the order from further infringement of United States Patent No. 7,069,546 (or, in the alternative, awarding Plaintiff a running royalty from the time of judgment going forward);

(c) Award Plaintiff damages resulting from Defendant's infringement in accordance with 35 U.S.C. § 284;

(d) Award Plaintiff pre-judgment and post-judgment interest and costs; and

(e) Award Plaintiff such further relief to which the Court finds Plaintiff entitled under law or equity.

Dated: January 31, 2019

Respectfully submitted,

/s/ Jay Johnson

JAY JOHNSON
State Bar No. 24067322
D. BRADLEY KIZZIA
State Bar No. 11547550
KIZZIA JOHNSON, PLLC
1910 Pacific Ave., Suite 13000
Dallas, Texas 75201
(214) 451-0164
Fax: (214) 451-0165
jay@kjpllc.com
bkizzia@kjpllc.com

ATTORNEYS FOR PLAINTIFF

EXHIBIT A