AN0009.1：EFM32 and EFR32 Series 1 入门指南

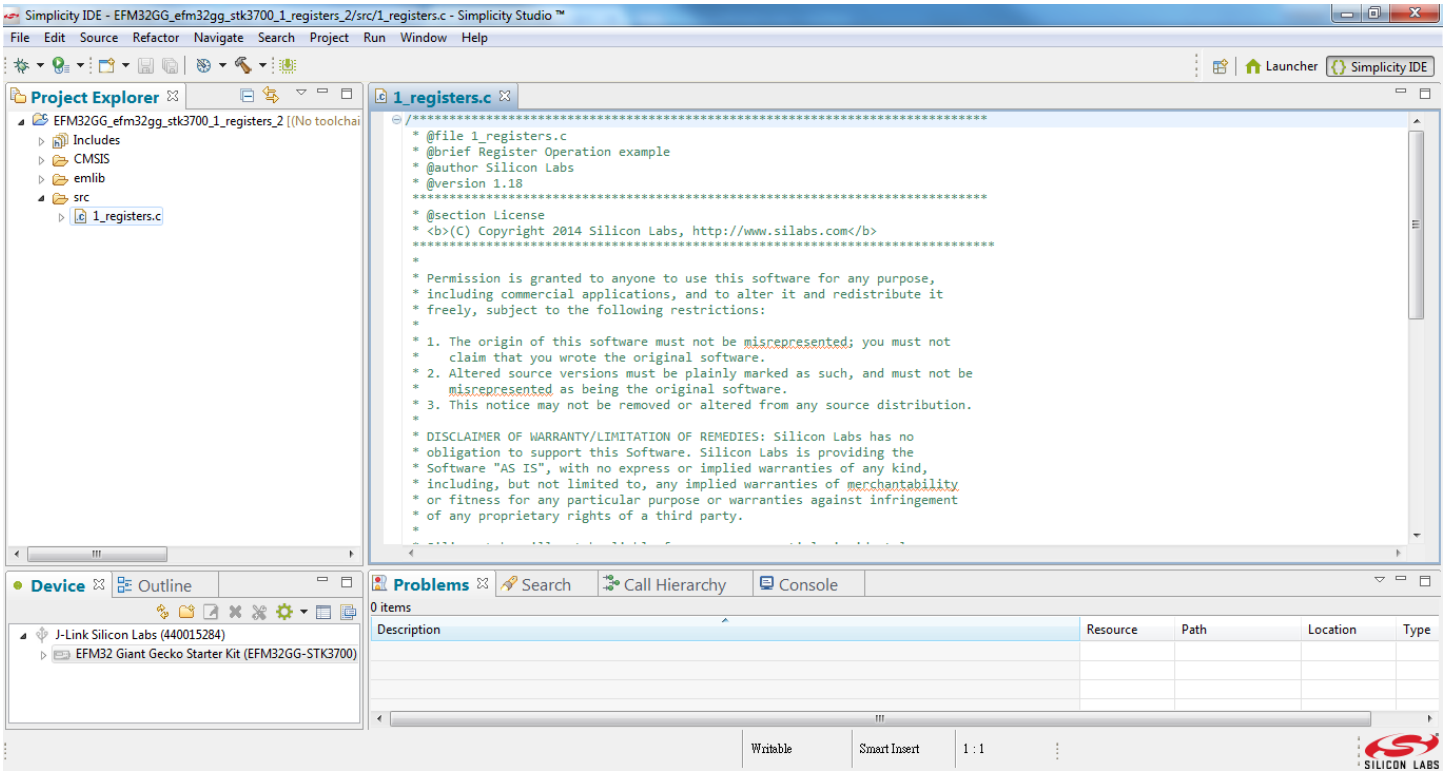
本应用说明介绍了适用于 EFM32 and EFR32 Series 1 的软件示例、库、文档和软件工具。

除了对这些设备的工具提供基本介绍，本文档还包括若干基本固件练习，帮助读者熟悉入门套件硬件、emlib 固件库和 Simplicity Studio 软件工具。

请注意，本文档重点介绍设备的 MCU 部分。有关无线产品（EFR32），请参见专用于该产品的用户指南中提供的其他无线入门信息。有关产品硬件的更多信息，请参见套件用户指南。有关 Simplicity Studio 总体情况的更多信息，请参见 *AN0822：Simplicity Studio™ 用户指南*。有关应用说明的信息，请访问 Silicon Labs 网站（www.silabs.com/32bit-appnotes）或进入 Simplicity Studio。

内容要点

- Simplicity Studio 包含使用 EFM32 and EFR32 Series 1 进行开发需要的所有信息。
- 您将学到：
  - 基本寄存器操作
  - 使用 emlib 函数
  - 闪烁 LED 和读取按钮
  - LCD 控制器
  - 能源模式
  - 实时计数器操作



1. 设备兼容性

本应用说明支持多个设备系列，某些功能因设备不同而有所差异。

MCU Series 1 consists of:

- EFM32 Jade Gecko (EFM32JG1/EFM32JG12)
- EFM32 Pearl Gecko (EFM32PG1/EFM32PG12)
- EFM32 Giant Gecko (EFM32GG11)
- EFM32 Tiny Gecko (EFM32TG11)

Wireless SoC Series 1 consists of:

- EFR32 Blue Gecko (EFR32BG1/EFR32BG12/EFR32BG13)
- EFR32 Flex Gecko (EFR32FG1/EFR32FG12/EFR32FG13)
- EFR32 Mighty Gecko (EFR32MG1/EFR32MG12/EFR32MG13)

2. 介绍

2.1 必备条件

本应用说明中的示例需要访问支持的 Silicon Labs 设备。支持的设备包含 EFM32 Series 1 入门套件。在使用本教程之前，确保通过以下方式提供兼容的 IDE：

- 从 Silabs.com（http://www.silabs.com/simplicity）安装 Simplicity Studio，或
- 如果首选 IAR，则安装最新版的 Segger J-Link 驱动器以支持套件（https://www.segger.com/jlink-software.html）

在 Simplicity Studio 中，点击主窗口左上角的[**更新软件**]按钮，确保安装所有可用的最新程序包：

Note: Simplicity Studio 包含功能齐全的集成 IDE，但也支持某些第三方 IDE，包括 IAR。因此无论首选 IDE 如何，始终建议安装 Simplicity Studio，以便轻松访问这些示例以及帮助开发 EFM32 and EFR32 Series 1 解决方案的全套功能。

2.2 如何使用本应用说明

本应用说明的源代码位于根文件夹 an0009_efm32_getting_started 中。

访问示例源代码和项目的最简单方式是使用 Simplicity Studio [**入门指南**]选项卡项下的[**应用说明**]对话框。点击任意应用说明即可打开[**应用说明**]对话框。

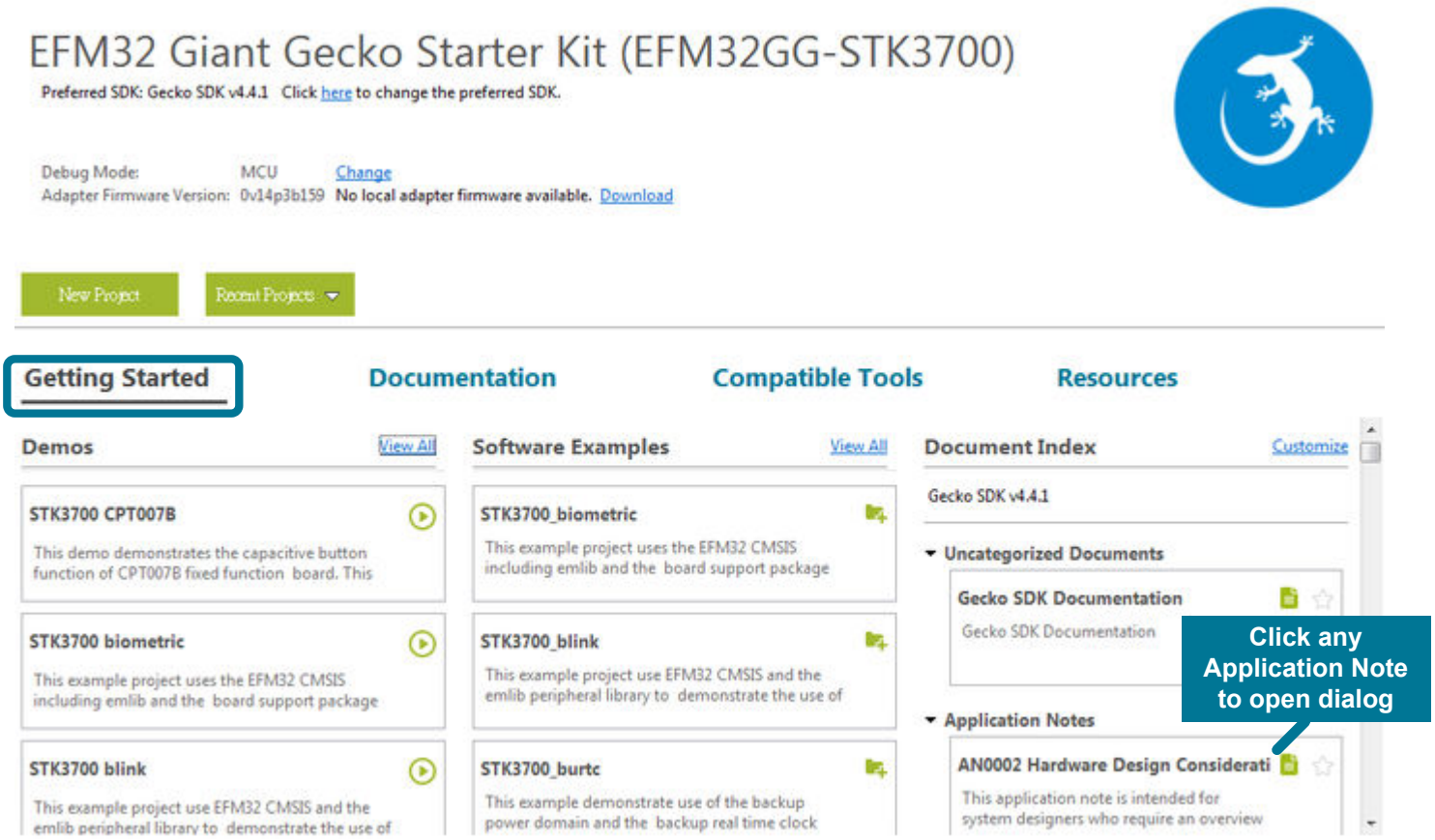


Figure 2.1. Simplicity Studio 中的应用说明

在[**应用说明**]对话框中，导航并选择 [**AN0009 EFM32 入门指南**]条目，然后点击[**导入项目...**]按钮，查看可用的示例项目列表。使用项目名称查找与套件兼容的示例，选择某个示例以将该项目导入 IDE（默认情况下为 Simplicity IDE，但必要时可将该配置更改为备用 IDE）。

另外，适用于多 IDE 的项目保存在用于存放这些示例的文件系统的单独文件夹（iar、arm 等）中，可通过 Simplicity Studio [**应用说明**]对话框中的[**打开文件夹**]按钮进行访问。可在相应的 IDE 中手动装载这些项目。所有 IAR 项目都集中一个在被称为 efm32.eww 的公共工作空间内。由于各种套件的项目略有不同，请务必打开前缀为正在使用的套件名称的项目。

Note: 本应用说明中的代码示例不完整，读者需要在练习过程中填写小段代码。对于每个示例，还提供完整的代码文件（后缀为 *_solution.c）。

3. 寄存器操作

本章介绍如何使用 [**CMSIS**] 和 [**emlib**] 软件库提供的定义和库函数为 EFM32 and EFR32 Series 1 设备写入 C 代码的基础知识。

3.1 地址

EFM32 and EFR32 Series 1 设备包含若干不同类型的外围设备（CMU、RTC、ADC……）。设备中的某些外围设备仅作为单个实例存在，如时钟管理单元 (CMU)。计时器 (TIMERn) 等其他外围设备作为多个实例存在，其名称后缀为表示实例编号的数字。通常，外围设备的两个实例完全相同，但位于内存映射的不同区域。然而，某些外围设备的每个实例具有不同功能集。例如，USART0 可具有 IrDA 接口，而 USART1 不可以。设备数据表和参考手册将介绍这些差异。

EFM32 and EFR32 Series 1 的外围设备使用存储器映射，即每个外围设备实例都具有专用地址区，其中包含可通过读/写操作访问的寄存器。有关外围设备实例和内存区的信息，请参见设备数据表。外围设备实例的起始地址被称为基址。设备系列的参考手册包含每个外围设备内寄存器的完整介绍。每个寄存器的地址作为外围设备实例基址的偏移量。

3.2 寄存器说明

EFM32 and EFR32 Series 1 设备采用 32 位总线，用于读/写访问外围设备，外围设备的每个寄存器包含 32 位（0-31）。未使用的位标记为保留，无法修改。外围设备使用的位可为单个位（如下图中的 OUTEN 位），也可集中为位域（如下图中的 PRSSEL 位域）。每个位域都具有下列属性：

- 位位置
- 名称
- 复位值
- 访问类型
- 描述

24.5.1 IDAC_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset										0x0						0		0	0	0					0x00				0	0	0	0
Access										RW						RW		RW	RW	RW					RW				RW	RW	RW	RW
Name										PRSSEL						OUTENPRS		APORTMASTERDIS	EM2DELAY	PWRSEL					APORTOUTSEL				OUTEN	MINOUTTRANS	CURSINK	EN

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
23:20	PRSSEL	0x0	RW	**IDAC Output PRS channnel Select**

Selects which PRS channel to use, when OUTENPRS is set.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected.
1	PRSCH1	PRS Channel 1 selected.
2	PRSCH2	PRS Channel 2 selected.
3	PRSCH3	PRS Channel 3 selected.
4	PRSCH4	PRS Channel 4 selected.
5	PRSCH5	PRS Channel 5 selected.
6	PRSCH6	PRS Channel 6 selected.
7	PRSCH7	PRS Channel 7 selected.
8	PRSCH8	PRS Channel 8 selected.
9	PRSCH9	PRS Channel 9 selected.
10	PRSCH10	PRS Channel 10 selected.
11	PRSCH11	PRS Channel 11 selected.

Figure 3.1. 示例寄存器说明

3.3 访问类型

每个寄存器具有适用于该寄存器内所有位域的一组访问类型。访问类型用于描述读/写操作对位域的反应。设备中寄存器的不同访问类型如下表所述。

Table 3.1. Register Access Types

Access Type	Description
R	Read only. Writes are ignored
RW	Readable and writable
RW1	Readable and writable. Only writes to 1 have effect
(R)W1	Sometimes readable. Only writes to 1 have effect. Currently only used for IFC registers
W1	Read value undefined. Only writes to 1 have effect
W	Write only. Read value undefined.
RWH	Readable, writable, and updated by hardware
RW(nB), RWH(nB), etc.	"(nB)" suffix indicates that register explicitly does not support peripheral bit set or clear
RW(a), R(a), etc.	"(a)" suffix indicates that register has actionable reads

3.4 CMSIS 和 emlib

内核微控制器软件接口标准（CMSIS）为适用于所有 ARM 内核设备的通用编码标准。由 Silicon Labs 提供的 CMSIS 库包括有头文件、定义（外围设备、寄存器和位域）和用于所有设备的启动文件。此外，CMSIS 还包含所有内核设备通用的函数，如中断处理、内在函数等。虽然可以使用硬编码地址和数据值来写入寄存器，但建议使用定义以确保代码的可移植性和可读性。

为了使用这些定义，项目必须在 c 文件中包含 em_device.h。该文件为所有 EFM32 and EFR32 Series 1 设备的通用头文件。在本文件中，根据为项目定义的预处理器符号，相应设备的头文件内容包含在项目构建中。

为了简化 EFM32 and EFR32 Series 1 设备的编程，Silicon Labs 开发并维护完整的 C 语言函数库 [**emlib**]，为设备的所有外围设备和内核函数提供高效、清晰、稳健的访问和控制。该函数库驻留在以下路径下 emlib 文件夹中的 em_xxx.c（如 em_dac.c）和 em_xxx.h（如 em_dac.h）文件中（其中 v1.1 为 Gecko SDK 套件版本号）。

```
C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.1\platform\emlib
```

在本应用说明包含的源文件中，每个源文件均包含 em_chip.h 并且每个 main() 函数的开头附近均存在 CHIP_Init() 调用。类似于 em_device.h 内容，在 CHIP_Init() 函数中采取的行动取决于所使用的具体部件，但包含对于已知勘误表的修正，或通过其他方式确保不同设备的行为一致。为此，在运行 CHIP_Init() 函数之前，不要在主函数中运行任何代码。

3.4.1 CMSIS 文档

选择相应的设备或套件后，可通过 Simplicity Studio 主窗口中 [**文档**] 项下的 [**Gecko SDK 文档**] 框访问 EFM32 and EFR32 Series 1 的 [**CMSIS**] 库和 [**emlib**] 的完整 Doxygen 文档。该文档也可见于 Silicon Labs 网站 http://devtools.silabs.com/dl/documentation/doxygen/ 或 GitHub 网站 https://siliconlabs.github.io/Gecko_SDK_Doc/。

3.4.2 外围设备结构

在 emlib 头文件中，每种外围设备类型的寄存器定义按结构分组，如下例所示：

```
typedef struct
{
__IO uint32_t CTRL;
__I uint32_t STATUS;
__IO uint32_t CH0CTRL;
__IO uint32_t CH1CTRL;
__IO uint32_t IEN;
__I uint32_t IF;
__O uint32_t IFS;
__O uint32_t IFC;
__IO uint32_t CH0DATA;
__IO uint32_t CH1DATA;
__O uint32_t COMBDATA;
__IO uint32_t CAL;
__IO uint32_t BIASPROG;
} DAC_TypeDef;
```

回顾下包含外围设备实例基址以及额外偏移量的寄存器地址。[**emlib**] 中的外围设备结构简化了寄存器写入并提取出基础地址和偏移量。因此，也可通过以下方式写入 DAC0 外围设备实例中的 CH0DATA：

```
DAC0->CH0DATA = 100;
```

类似地，也可通过以下方式读取寄存器：

```
myVariable = DAC0->STATUS;
```

3.4.3 位域定义

每个设备都具有为每个外围设备定义的相关位域。这些定义见于 efm32xx_xxx.h（如 efm32tg_dac.h）文件，并且自动包含在相应的 [**emlib**] 外围设备头文件中。

```
#define _DAC_CTRL_REFRSEL_SHIFT              20
#define _DAC_CTRL_REFRSEL_MASK               0x300000UL
#define _DAC_CTRL_REFRSEL_DEFAULT            0x00000000UL
#define _DAC_CTRL_REFRSEL_8CYCLES            0x00000000UL
#define _DAC_CTRL_REFRSEL_16CYCLES           0x00000001UL
#define _DAC_CTRL_REFRSEL_32CYCLES           0x00000002UL
#define _DAC_CTRL_REFRSEL_64CYCLES           0x00000003UL
#define DAC_CTRL_REFRSEL_DEFAULT             (_DAC_CTRL_REFRSEL_DEFAULT << 20)
#define DAC_CTRL_REFRSEL_8CYCLES             (_DAC_CTRL_REFRSEL_8CYCLES << 20)
#define DAC_CTRL_REFRSEL_16CYCLES            (_DAC_CTRL_REFRSEL_16CYCLES << 20)
#define DAC_CTRL_REFRSEL_32CYCLES            (_DAC_CTRL_REFRSEL_32CYCLES << 20)
#define DAC_CTRL_REFRSEL_64CYCLES            (_DAC_CTRL_REFRSEL_64CYCLES << 20)
```

对于每个寄存器位域，还定义了相关移位、掩码和默认值位域。

```
#define DAC_CTRL_DIFF                        (0x1UL << 0)
#define _DAC_CTRL_DIFF_SHIFT                 0
#define _DAC_CTRL_DIFF_MASK                  0x1UL
#define _DAC_CTRL_DIFF_DEFAULT               0x00000000UL
#define DAC_CTRL_DIFF_DEFAULT                (_DAC_CTRL_DIFF_DEFAULT << 0)
```

3.4.4 寄存器访问示例

设置控制寄存器中的位时，务必确保固件不会意外清除该寄存器中的其他位。为了确保这一点，可使用初始内容对包含固件需要设置的位的掩码进行“OR”运算，如下例所示：

```
DAC0->CTRL = DAC0->CTRL | DAC_CTRL_LPFEN;
```

更紧凑的版本为：

```
DAC0->CTRL |= DAC_CTRL_LPFEN;
```

通过对寄存器与所有位均经过设置（待清除的位除外）的值进行“AND”运算来清除位：

```
DAC0->CTRL = DAC0->CTRL & ~DAC_CTRL_LPFEN; // or
DAC0->CTRL &= ~DAC_CTRL_LPFEN;
```

在对包含多位的位域设置新值时，简单的“OR”函数无法完成，因为将初始位域内容与掩码进行“OR”运算可能产生错误的结果。务必在使用新值进行“OR”运算之前清除整个位域（且仅清除位域）：

```
DAC0->CTRL = (DAC0->CTRL & ~_DAC_CTRL_REFRSEL_MASK) | DAC_CTRL_REFRSEL_16CYCLES;
```

3.4.5 分组寄存器

一些寄存器在每个外围设备中被分组在一起。这种组的示例为与每个 GPIO 端口相关联的寄存器，如下图所示的数据输出寄存器 (DOUT)。每个 GPIO 端口（A、B、C、...） 包含 1 个 DOUT 寄存器，以下描述对所有寄存器通用。GPIO_Px_DOUT 中的 x 表示端口通配符。

27.5.4 GPIO_Px_DOUT - Port Data Out Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	DOUT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in 1.2 Conventions*		
15:0	DOUT	0x0000	RW	**Data Out**
	Data output on pin.			

Figure 3.2. GPIO 中的分组寄存器

在 CMSIS 定义中，端口寄存器在阵列 P[x] 被分组在一起。使用该阵列时，我们必须使用数字而非端口字母（A=0、B=1、C=2、...）编制索引。可通过以下方式访问端口 C 的 DOUT 寄存器：

```
GPIO->P[2].DOUT = 0x000F;
```

4. 示例 1 - 寄存器操作

本例展示如何使用 CMSIS 定义写入和读取寄存器。本教程还展示如何在 Simplicity Studio 或 IAR 嵌入式工作台中通过调试器观察和操纵寄存器内容。虽然所示的示例仅针对 Simplicity Studio 和 IAR，但也可在其他受支持的 IDE 中完成这些任务。

对于 Simplicity Studio：

1. 将套件连接到 PC，打开 Simplicity Studio。
2. 出现套件后，在 [**设备**] 窗口中点击套件（如 EFM32 Giant Gecko 入门套件 (EFM32GG-STK3700)）。
3. 点击[**入门指南**]选项卡下的任一应用说明，打开[**应用说明**]对话框。
4. 搜索 [**AN0009 EFM32 入门指南**]，点击列表中的文档，然后点击[**导入项目...**]按钮。
5. 在对话框中选择 [**<kit_name>_1_registers.slsproj**] 选项，然后点击[**确定**]。
6. 双击 1_registers.c 文件，在编辑器视图中打开该文件。存在应添加自定义代码的标记。

对于 IAR：

1. 打开 efm32 工作空间 (an\an0009_efm32_getting_started\iar\efm32.eww)。
2. 在 IAR 嵌入式工作台中选择 [**<kit_name>_1_registers**] 项目。
3. 在 1_registers.c（源文件内）主函数中，存在应添加自定义代码的标记。

4.1 第 1 步 - 启用定时器时钟

在本例中，我们将使用 TIMER0。高频 RC 振荡器 (HFRCO) 以默认频带运行，但所有外围设备时钟均被禁用，所以在使用之前必须打开 TIMER0 的时钟。如果翻看参考手册中的 CMU 一章，我们看到可通过设置 CMU 外围设备中 HFPERCLKEN0 寄存器的 TIMER0 位来打开 TIMER0 的时钟。

4.2 第 2 步 - 启动定时器

通过对 TIMER0 中 CMD 寄存器的 START 位写入 1 来启动定时器。

4.3 第 3 步 - 等待阈值

创建等待计时器达到 1000 以后再继续的 while 循环。

4.4 观察

对于 Simplicity IDE，务必通过点击左侧[**项目**]视图中的项目来激活 [**<kit_name>_1_registers**] 项目。然后按下[**调试**]按钮，以自动构建代码并将代码下载到设备。点击[**寄存器**]视图，在 TIMER0 中查找 STATUS 寄存器。展开寄存器之后，请注意 RUNNING 位已设为 0。

在代码视图中，双击左窗格放置断点，然后通过固件启动定时器并点击[**继续**]按钮。然后在使用[**越过**]按钮单步越过表达式时，观察[**寄存器**]视图中的 RUNNING 位是否设为 1。继续单个步骤，注意到 CNT 寄存器的内容正在增加。直接在[**寄存器**]视图中输入，对 CNT 寄存器写入不同的值。

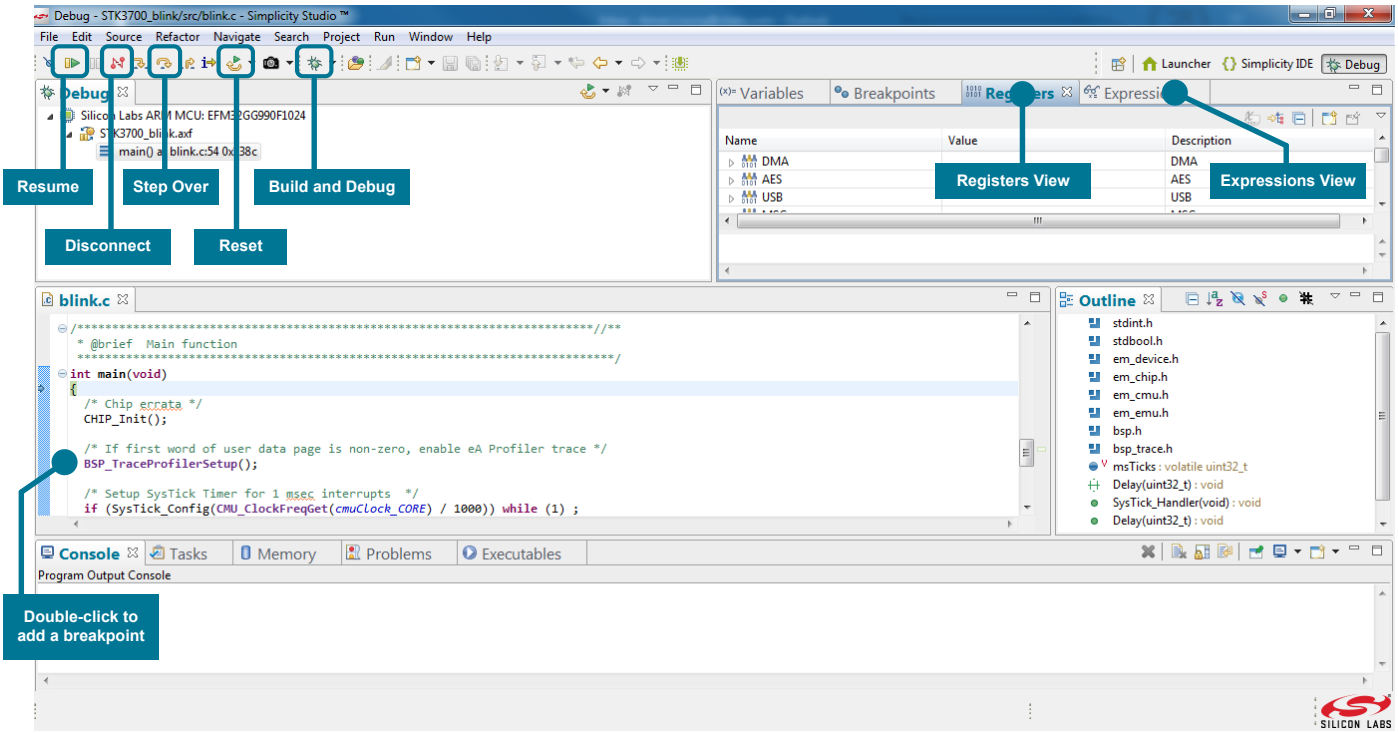


Figure 4.1. Simplicity IDE 中的调试视图

对于 IAR，务必通过按下工作空间窗口底部对应的选项卡来激活 [**<kit_name>_1_registers**] 项目。然后按下[**下载并调试**]按钮，进入[**视图**]->[**寄存器**]，在 TIMER0 中查找 STATUS 寄存器。展开寄存器之后，请注意 RUNNING 位已设为 0。

将光标置于固件启动定时器的行之前，按下[**运行至光标处**]。然后，在点击[**逐步执行**]按钮以越过表达式时，观察[**寄存器**]视图中的 RUNNING 位是否设为 1。继续点击[**逐步执行**]按钮以增加 CNT 寄存器的内容。直接在[**寄存器**]视图中输入，对 CNT 寄存器写入不同的值。

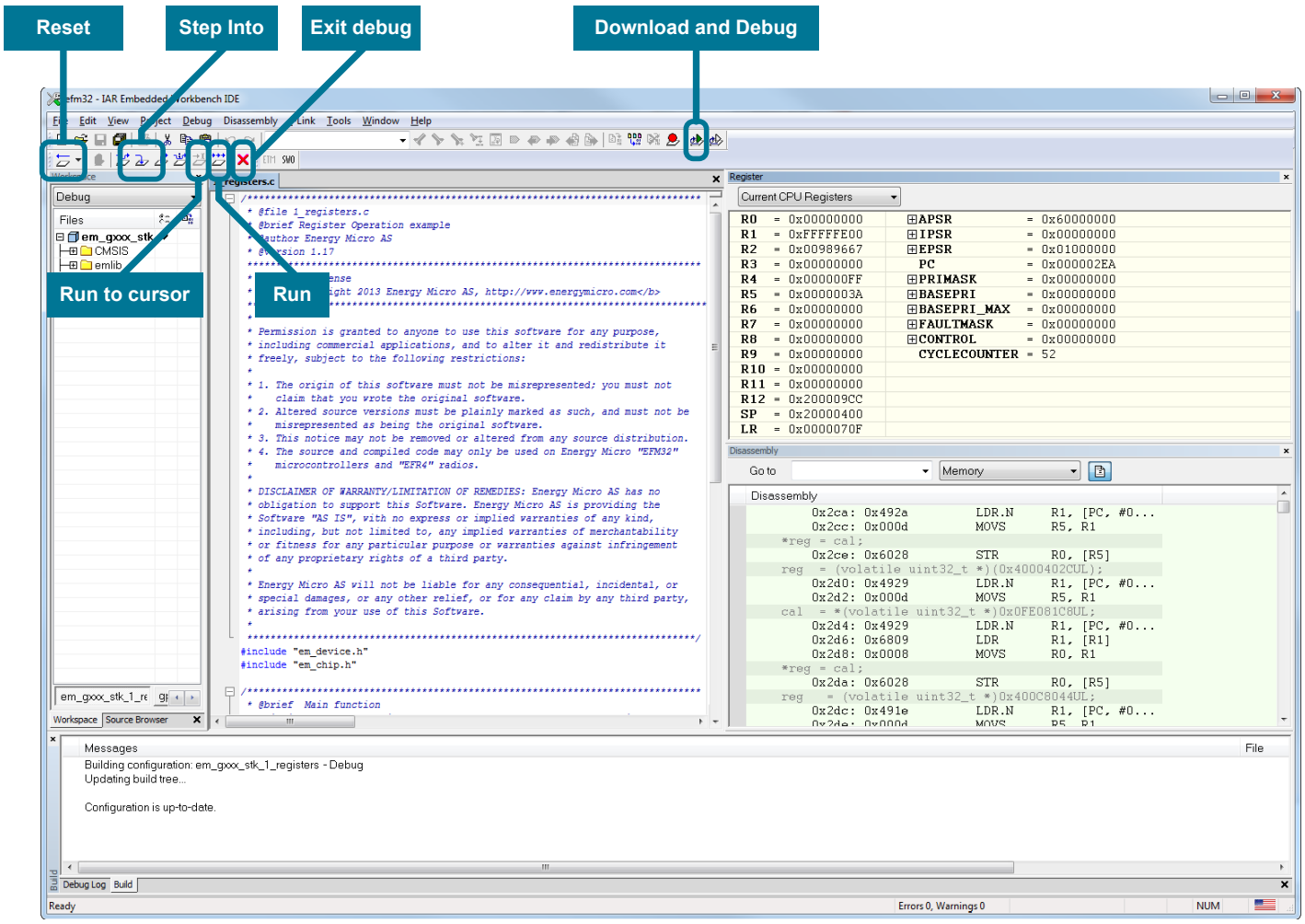


Figure 4.2. IAR 中的调试视图

5. 示例 2 – 通过 STK 使 LED 闪烁

在本例中，目的在于使用 GPIO 引脚点亮 STK 上的 LED 并且在每次按下按钮更改 LED 配置。不直接访问寄存器，而是使用 emlib 函数配置外围设备。

AN0009 应用说明包含在本例中将使用的 Simplicity Studio 和 IAR efm32 工作空间中的 [**<kit_name>_2_leds**] 项目。该项目中包含 emlib C 文件。C 文件的开头包含对应的头文件。

有关存在哪些 emlib 函数及如何使用这些函数的详情，在 Simplicity Studio 中使用[**文档**]下的 [**Gecko SDK 文档**]（或通过访问 http://devtools.silabs.com/dl/documentation/doxygen/）打开 API 文档。点击正确设备（如 Giant Gecko）的软件文档链接之后，打开[**模块->EMLIB**]，选择 [**CMU**] 外围设备。点击窗口右上角的[**函数**]链接，查找该外围设备的函数列表。可使用这些函数轻松运行时钟管理单元。

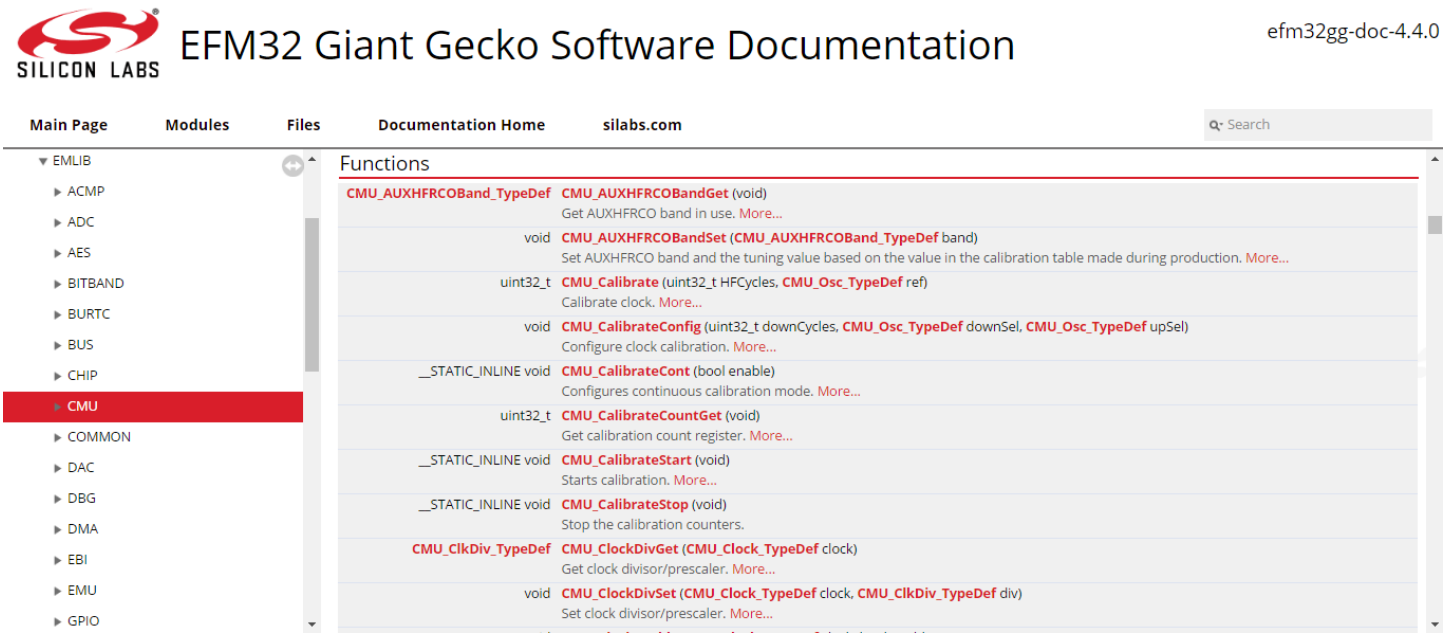


Figure 5.1. 特定于 CMU 的 emlib 函数的文档

5.1 第 1 步 - 打开 GPIO 时钟

在 CMU 函数列表中，我们发现以下函数可以打开 GPIO 的时钟：

```
void CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)
```

如果点击函数，我们将看到如何使用该函数的描述。点击 [**CMU_Clock_TypeDef**] 链接，查看时钟参数允许使用的枚举器的列表。如果要打开 GPIO，请添加如下函数：

```
CMU_ClockEnable(cmuClock_GPIO, true);
```

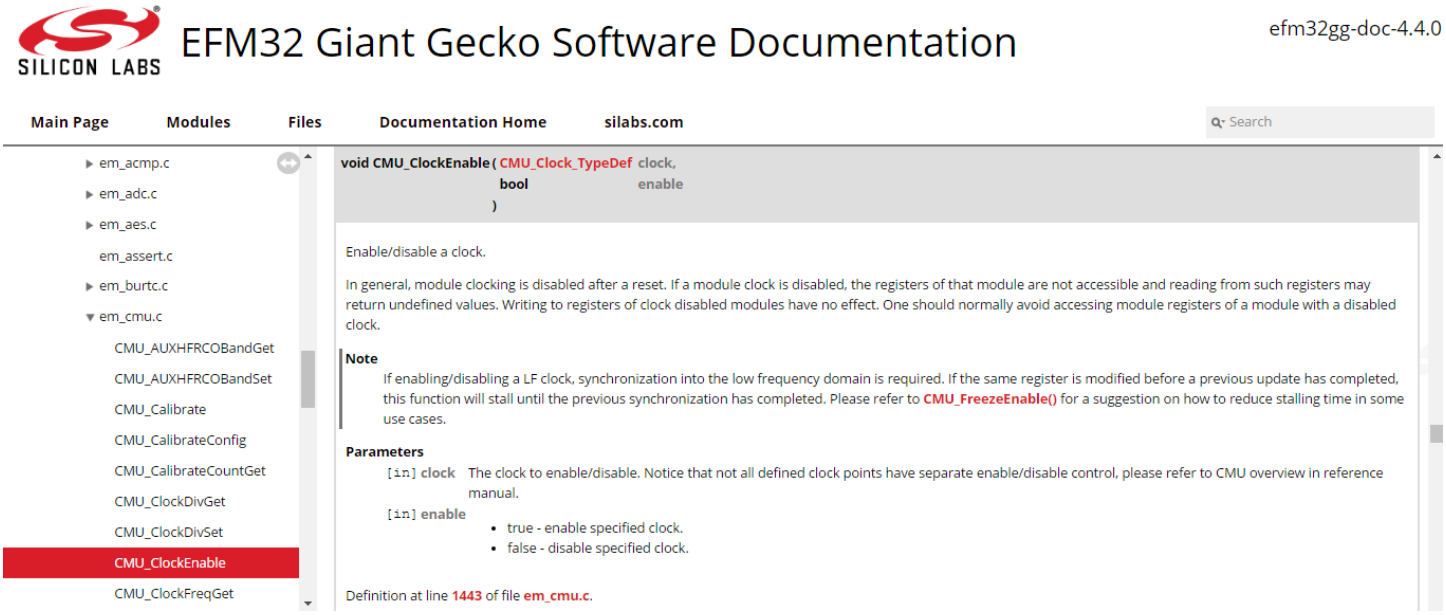


Figure 5.2. CMU_ClockEnable 函数描述

5.2 第 2 步 - 为 LED 配置 GPIO 引脚

在设备或解决方案窗口中选择对应的套件后，可在[**文档**]下的 Simplicity Studio 中查找该套件的用户手册。本文档介绍所有引脚的使用，包括用户 LED。这些 LED 在这些 EFM32 Series 1 套件上按如下方式连接：

- SLSTK3401A_EFM32PG: 2 LEDs on port F, pins 4-5
- SLSTK3402A_EFM32PG12: 2 LEDs on port F, pins 4-5
- SLSTK3701A_EFM32GG11: 2 RGB LEDs on port H, pins 10-12, and port H, pins 13-15, respectively
- SLSTK3301A_EFM32TG11: 2 LEDs on port D pin 2, and port C pin 2, respectively
- BRD4151A_EFR32MG1: 2 LEDs on port F, pins 4-5
- BRD4161A_EFR32MG12: 2 LEDs on port F, pins 4-5
- BRD4158A_EFR32MG13: 2 LEDs on port F, pins 4-5

有关特定于所使用套件的信息，请参考用户手册。

查看适用于 GPIO 的函数，可使用如下函数配置 GPIO 引脚模式：

```
void GPIO_PinModeSet(GPIO_Port_TypeDef port, unsigned int pin,
   GPIO_Mode_TypeDef mode, unsigned int out)
```

使用该函数将 LED 引脚配置为推拉式输出，将初始 DOUT 值设为 0。

5.3 第 3 步 - 为按钮配置 GPIO 引脚

查看关于 STK 的用户手册，了解在何处将按钮 0（PB0）连接在所使用的套件上。此按钮在几个示例套件上按如下方式连接：

- SLSTK3401A_EFM32PG: Port F, pin 6
- SLSTK3402A_EFM32PG12: Port F, pin 6
- SLSTK3701A_EFM32GG11: Port C, pin 8
- SLSTK3301A_EFM32TG11: Port D, pin 5
- BRD4151A_EFR32MG1: Port F, pin 6
- BRD4161A_EFR32MG12: Port F, pin 6
- BRD4158A_EFR32MG13: Port F, pin 6

将该引脚配置为能够检测按钮状态的输入。

5.4 第 4 步 - 在按下按钮时变更 LED 状态

写一个在每次按下 PB0 时切换 LED 的循环。确保不仅要检查按钮是否按下，还要检查是否释放，以便在每次按下按钮时切换一次 LED。PB0 通过外部电阻被拉到高电平。

5.5 额外任务 - LED 动画

在 LED 上试验创建不同的闪烁模式，如使 LED 颜色变浅和运行（如果存在多个 LED）。由于默认情况下设备通常以 HFRCO 频段运行，因此添加延迟函数以实时查看 LED 变化。在示例 1 中，对 TIMER0 使用该代码，创建如下函数：

```
void Delay(uint16_t milliseconds)
```

使用 TIMER0_CTRL 中的 PRESC 位域减小所需值的时钟频率。

6. 示例 3b — 记忆 LCD

本例要求 STK 配备记忆 LCD（如 Happy Gecko、Zero Gecko、Pearl Gecko 和 EFR32xG 无线 STK）。本例展示如何配置记忆 LCD 驱动程序以及如何在记忆 LCD 上写入文本。本例中将使用名为 [<kit_name>_3_lcd_mem] 的软件项目。

Gecko SDK 包括适记忆 LCD 的驱动程序。该显示器驱动程序的文档位于对应套件[**软件文档**]中[**套件驱动程序**]章节下的[**显示器**]中。

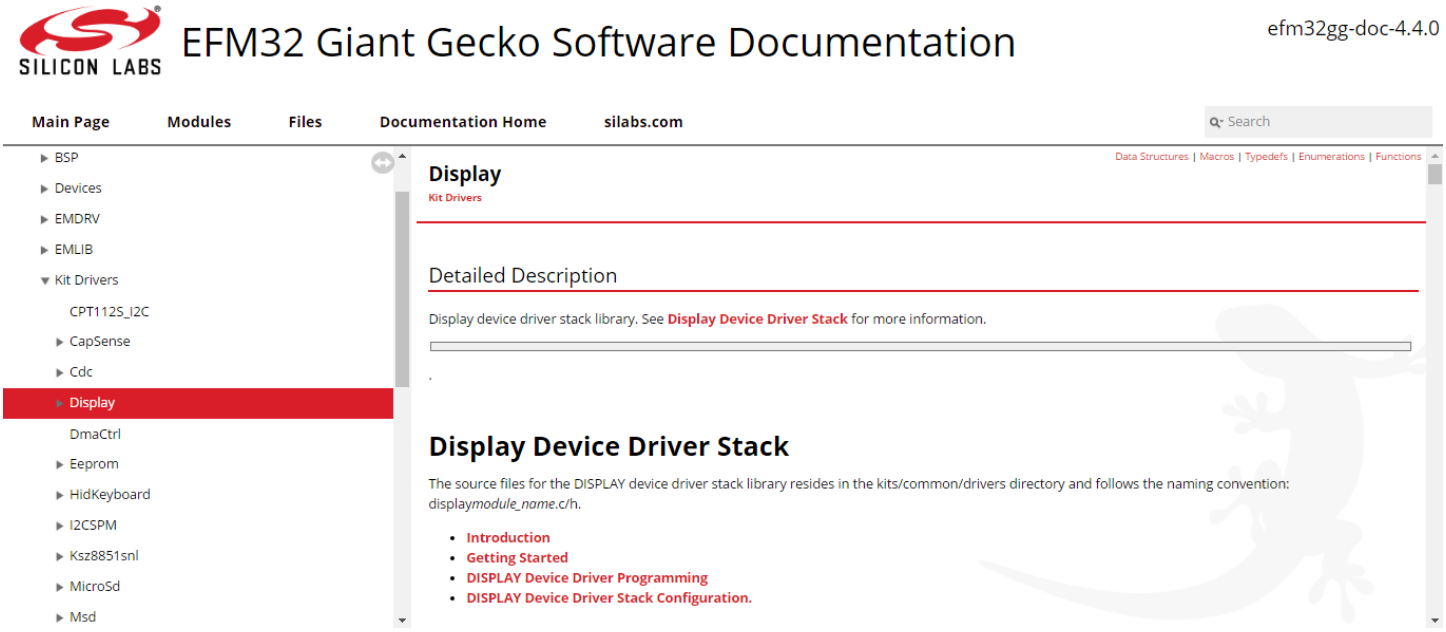


Figure 6.1. 显示器驱动程序的文档

6.1 第 1 步 — 配置显示器驱动程序

首先，通过 DISPLAY_Init() 初始化 DISPLAY 驱动程序。

该显示器驱动程序包括 TEXTDISPLAY，这是用于将文本打印到显示器设备的接口。使用 TEXTDISPLAY_New() 创建新的 TEXTDISPLAY 接口。

6.2 第 2 步 — 将文本写入记忆 LCD

TEXTDISPLAY 可实现用于将文本写入记忆 LCD 的基本函数。尝试 TEXTDISPLAY_WriteString() 和 TEXTDISPLAY_WriteChar()。

7. 示例 4 — 能源模式

本例展示如何进入不同的能源模式（EMx）以及如何使用 RTC 或 RTCC 中断进行唤醒。本例中使用名为 [**<kit_name>_4_energymodes**] 的项目。

7.1 带有 STK 的高级电能监测器

入门套件包括 VMCU 电源域的电流测量系统，用于为设备、LCD 显示器及入门套件应用部件分的其他元件供电。可使用 Simplicity Studio 中的 Energy Profiler 在 PC 上实时监测电流测量结果。

7.2 第 1 步 — 进入 EM1

执行等待中断指令并清除 SCB_SCR 寄存器中的 SLEEPDEEP 位，从而进入 EM1。该指令的内在函数（CMSIS 的一部分）如下所示：

```
SCB->SCR &= ~SCB_SCR_SLEEPDEEP_Msk;
__WFI();
```

执行完该指令后，观察电流消耗是否下降。

此外，EMU emlib 函数包括用于进入能源模式的函数，固件可使用这些函数而不是手动清除 SLEEPDEEP 位和执行 WFI 指令：

```
void EMU_EnterEM1()
```

7.3 第 2 步 — 进入 EM2

也通过执行 WFI 指令来进入 EM2 ，不同之处在于还要设置 SCB_SCR 寄存器中的 SLEEPDEEP 位并启用低频振荡器。首先启用低频振荡器（LFRCO 或 LFXO），然后再进入深度睡眠，从而进入 EM2。在本例中，通过使用如下 emlib 函数来启用 LFRCO 并等待其稳定：

```
void CMU_OscillatorEnable(CMU_Osc_Typedef osc, bool enable, bool wait)
```

现在，通过设置 SLEEPDEEP 并执行 WFI 指令来进入 EM2：

```
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
__WFI();
```

此外，EMU emlib 函数包括用于进入能源模式的函数，固件可使用这些函数而不是手动设置 SLEEPDEEP 位和执行 WFI 指令：

```
void EMU_EnterEM2(bool restore)
```

强烈建议使用该函数。该 emlib 函数还将避免或修订任何影响能源模式操作的勘误表问题。

Note: 当处于主动调试会话时，不允许设备低于 EM1。要测量 EM2 中的电流消耗，需结束调试会话并使用 STK 上的“复位”按钮将设备复位。

7.4 第 3 步 — 进入 EM3

首先禁用所有低频振荡器，然后再进入深度睡眠（通过使用与 EM2 相同的技术来实现），从而进入 EM3。通过使用如下 emlib 函数来禁用第 2 步中原本已启用的 LFRCO：

```
void CMU_OscillatorEnable(CMU_Osc_Typedef osc, bool disable, bool wait)
```

现在，通过设置 SLEEPDEEP 并执行 WFI 指令来进入 EM3：

```
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
__WFI();
```

此外，EMU emlib 函数包括用于进入能源模式的函数，固件可使用这些函数而不是手动禁用 LF 振荡器、设置 SLEEPDEEP 位和执行 WFI 指令：

```
void EMU_EnterEM3(bool restore)
```

强烈建议使用该函数。该 emlib 函数还将避免或修订任何影响能源模式操作的勘误表问题。

Note：当处于主动调试会话时，不允许设备低于 EM1。要测量 EM3 中的电流消耗，需结束调试会话并使用 STK 上的“复位”按钮将设备复位。

7.5 第 4 步 — 配置 MCU Series 1 的实时计数器和日历 (RTCC)

将实时计数器和日历 (RTCC) 配置为在 5 秒后中断，以从 EM2 中唤醒。首先，通过使用 CMU emlib 函数将时钟启用为 RTCC。要与 RTCC 等低能耗/频率外围设备进行通信，还需要为 LE 接口启用时钟 (cmuClock_HFLE)。

RTCC 的 emlib 初始化函数要求将配置结构作为输入：

```
void RTCC_Init(const RTCC_Init_TypeDef *init)
```

该结构已在代码中声明，但在通过 RTCC_Init 函数使用之前，固件必须在结构中设置 9 个参数：

```
rtccInit.cntWrapOnCCV1 = true;
```

用于 RTCC 捕获/比较信道的 emlib 初始化函数要求将配置结构作为输入：

```
void RTCC_ChannelInit(int ch, RTCC_CCChConf_TypeDef const *confPtr)
```

该结构已在代码中声明，但在通过 RTCC_ChannelInit 函数使用之前，固件必须在结构中设置 7 个参数：

```
rtccInitCompareChannel.chMode = rtccCapComChModeCompare;
```

接下来，在 RTCC 中设置捕获/比较值寄存器 1 (CC1_CCV)，当比较值与计数器值匹配时，这将设置中断标记 CC1。考虑到 RTCC 以 32.768 kHz 的频率运行，选择等于 5 秒的值：

```
void RTCC_ChannelCCVSet(int ch, uint32_t value)
```

现在，RTCC CC1 标记将在设置比较匹配上，但还必须设置对应的允许位以从 RTCC 中生成中断请求：

```
void RTCC_IntEnable(uint32_t flags)
```

RTCC 中断请求在比较器匹配上启用，但若要触发中断，必须在 Cortex-M 中启用 RTCC 中断请求行。要使用的 IRQn_Type 为 RTCC_IRQn。

```
NVIC_EnableIRQ(RTCC_IRQn);
```

RTCC 的中断处理程序已包含在代码中 (RTCC_IRQHandler)，但是为空。在该函数中，添加函数调用以清除 RTCC CC1 中断标记。如果固件不执行此操作，Cortex-M 将卡在中断处理程序中，因为中断永远不会无效置位。使用 RTCC emlib 函数来清除中断标记。

8. 摘要

恭喜！现在，您已了解节能编程的基础知识，包括寄存器和 GPIO 操作、STK 和 LCD 上基本函数的使用，以及如何处理设备中的不同能源模式和 emlib/CMSIS 函数。Simplicity Studio 中［**入门指南**］选项卡下方的［**软件示例**］和［**应用说明**］提供了更多可供探索的示例和应用说明。

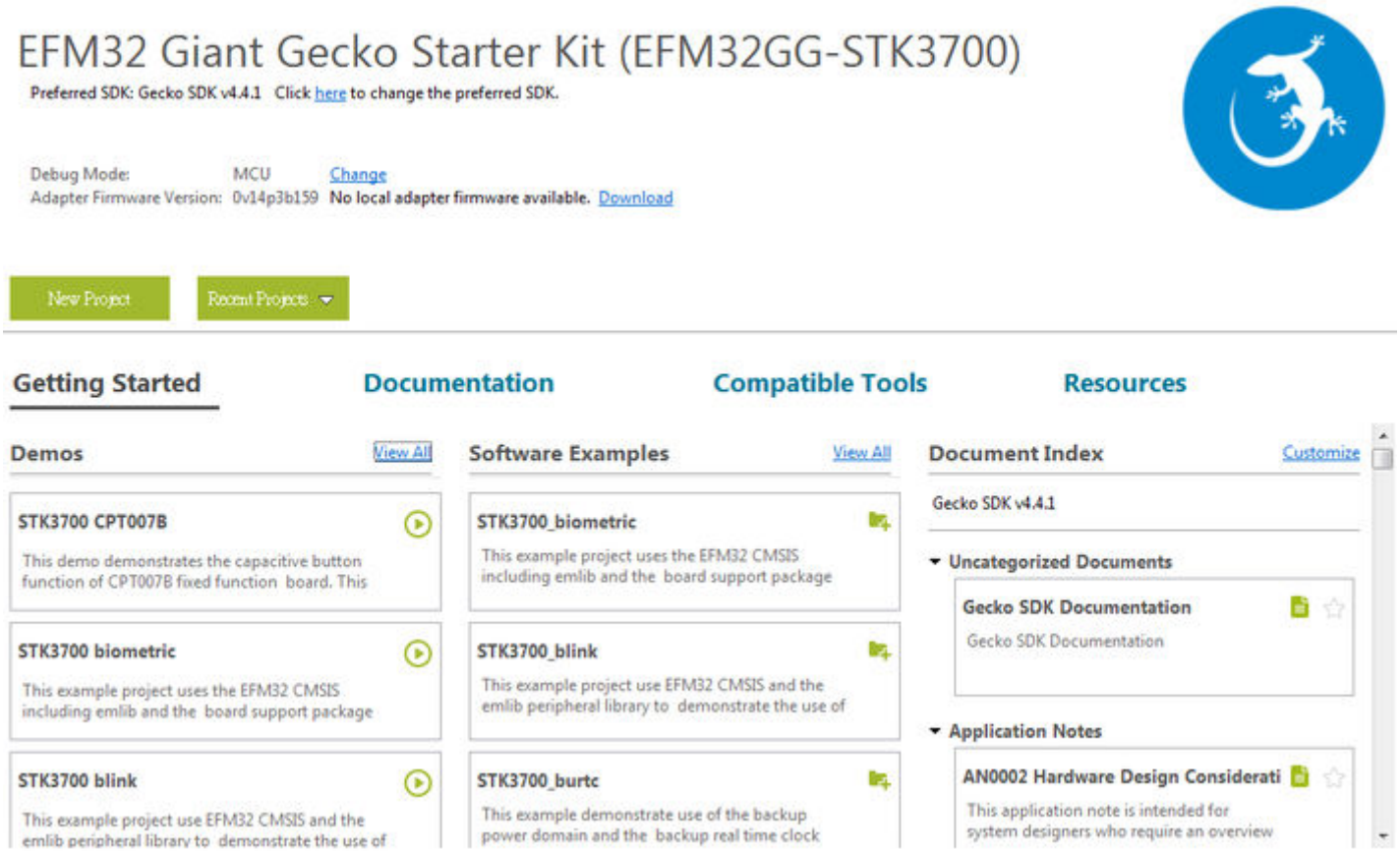


Figure 8.1. Simplicity Studio 中的软件示例和应用说明

9. 版本历史

修订版 1.22

2018 年 1 月

- 删除了关于 EFM32PG13/EFM32JG13 的参考
- 增加了 EFM32TG11 入门套件的示例。

修订版 1.21

2017 年 6 月

- 将 AN0009 分为 AN0009.0 和 AN0009.1，分别用于 MCU/无线 MCU 系列 0 和 MCU/无线 SoC 系列 1。
- 删除了示例 4 文本中的 RTC 配置说明。
- 添加了 EFM32PG12、EFM32GG11、EFR32MG1、EFR32MG12 和 EFR32MG13 入门套件的示例。

修订版 1.20

2017 年 1 月

- 更新了 Simplicity Studio V4。
- 删除了 EFM32G 开发套件的示例。
- 添加了 EFM32PG1 入门套件的示例。

修订版 1.19

2017 年 11 月

- 增加了对 EFM32 Gemstones 和 EFR32 无线 Gecko 产品组合的支持。

修订版 1.18

2014 年 5 月

- 更新了 CMSIS 3.20.5 的示例代码
- 更改了代码示例中的 Silicon Labs 许可
- 增加了 Simplicity IDE 的示例项目
- 删除了 Sourcery CodeBench Lite 的示例生成文件

修订版 1.17

2013 年 10 月

- 增加了缺失的头文件

修订版 1.16

2013 年 10 月

- 新封面布局
- 增加了对 Zero Gecko 入门套件 (EFM32ZG-STK3200) 的支持
- 在位域定义章节新增文本
- 修复了 STK3700 和 LCD 动画的问题

修订版 1.15

2013 年 5 月

- 增加了 ARM/GCC 和 Atollic TrueStudio 的软件项目。

修订版 1.14

2012 年 11 月

- 增加了对 Giant Gecko 入门套件 (EFM32GG-STK3700) 的支持
- 针对新套件驱动程序和 bsp 结构调整了软件项目

修订版 1.13

2012 年 4 月

- 针对新外围设备库命名和 CMSIS_V3 调整了软件项目

修订版 1.12

2012 年 3 月

- 在 LED 项目中增加了 efm32lib 文件 efm32_emu.c
- 修复了 CodeSourcery 项目的生成文件错误

修订版 1.11

2011 年 10 月

- 更新了 IDE 项目路径与新套件目录

修订版 1.10

2011 年 9 月

- 增加了对 Tiny Gecko 入门套件 (EFM32TG-STK3300) 的支持

修订版 1.03

2011 年 5 月

- 更新了项目，以与新的 bsp 版本保持一致

修订版 1.02

2010 年 11 月

- 更正了新 EFM32LIB 函数的解决方案 c 文件
- 更正了新 EFM32LIB 函数的 pdf 文档

修订版 1.01

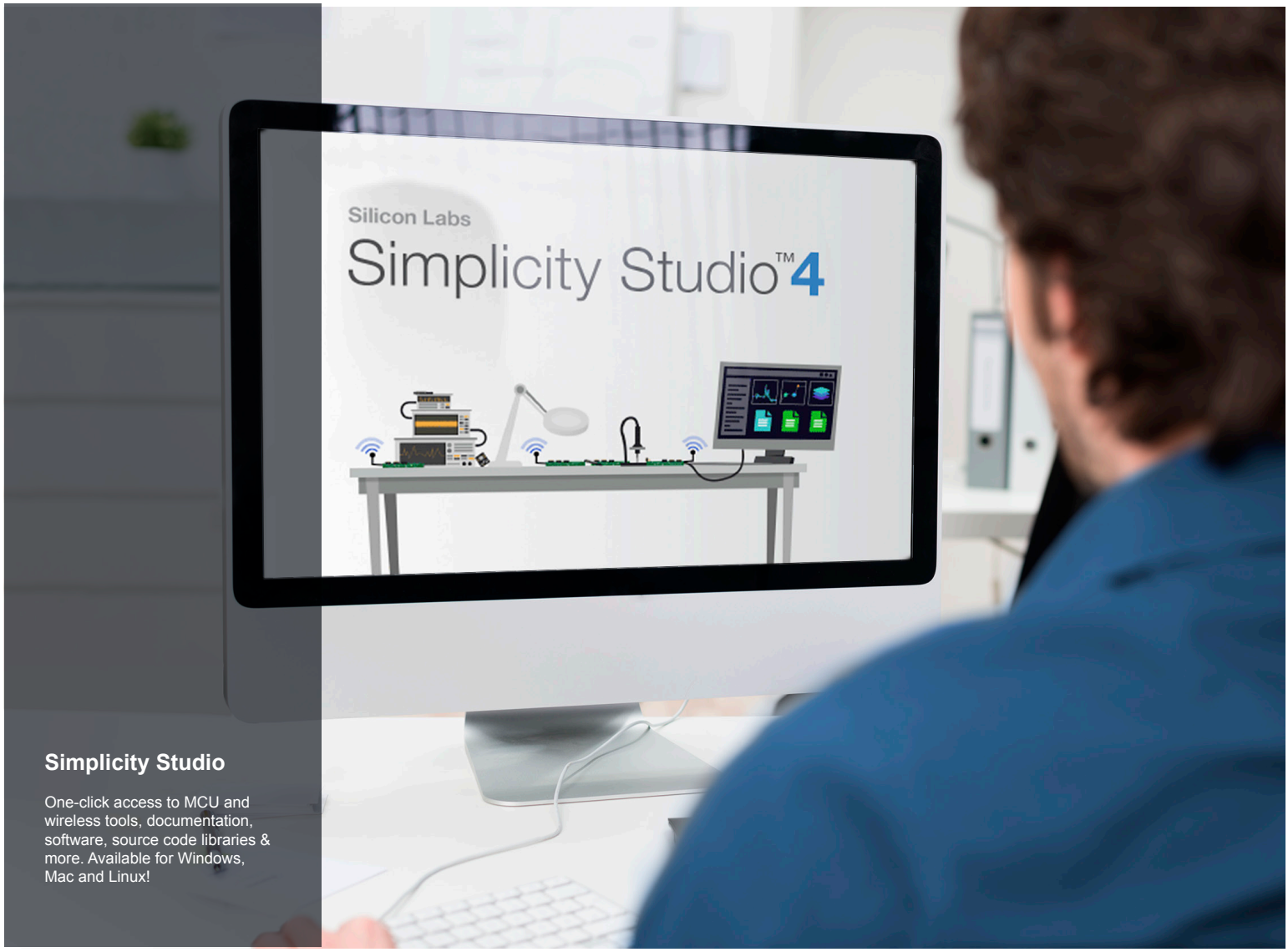
2010 年 11 月

- 更改了软件/文档以使用 segmentlcd.c 函数，lcdcontroller.c 已过时
- 增加了有关 CMSIS Doxygen 文档的章节
- 更改了示例文件夹结构，删除了 build 和 src 文件夹
- 将 chip init 函数更新至最新的 efm32lib 版本

修订版 1.00

2010 年 9 月

- 初始版本



Simplicity Studio

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

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com