AN0009.0: EFM32 and EZR32 Series 0 スタート・ガイド

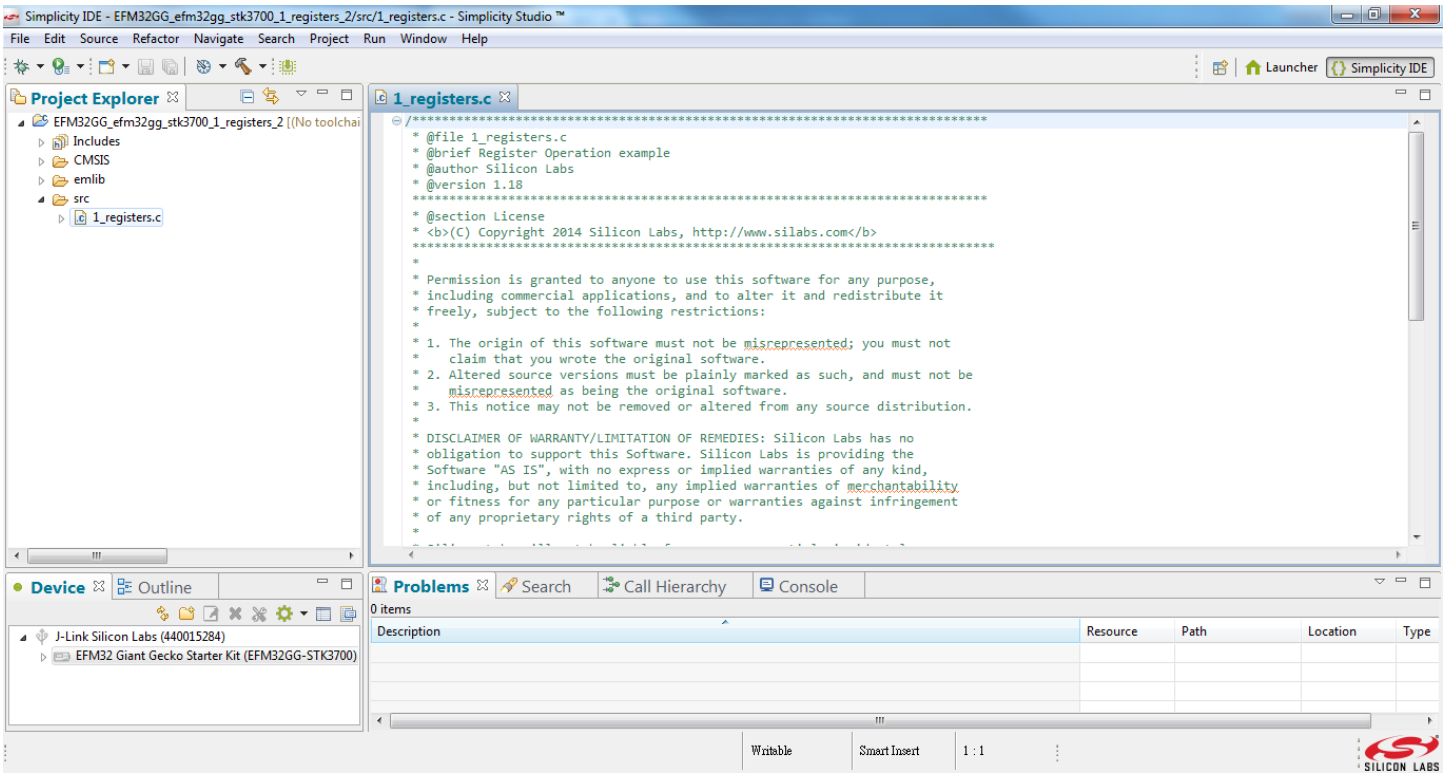
このアプリケーション・ノートでは、EFM32 and EZR32 Series 0 で使用可能なソフトウェア例、ライブラリ、ドキュメント、ソフトウェア・ツールについて紹介します。

このドキュメントには、これらのデバイスで使用できるツールに関する基本情報に加え、スターター・キット・ハードウェア、emlib ファームウェア・ライブラリ、Simplicity Studio ソフトウェア・ツールに慣れるためのファームウェアに関する基本演習もいくつか記載されています。

このドキュメントでは、主に、デバイスの MCU 部分を取り上げます。ワイヤレス製品 (EZR32) については、製品のユーザ・ガイドに記載されているワイヤレスに関するスタート・ガイド追加情報を参照してください。製品のハードウェアの詳細については、キットのユーザ・ガイドに記載されています。Simplicity Studio の一般的な詳細については、『*AN0822 : Simplicity Studio™ User Guide*』に記載されています。アプリケーション・ノートは、Silicon Labs のウェブサイト (www.silabs.com/32bit-appnotes) または Simplicity Studio でご確認いただけます。

要点

- Simplicity Studio には、EFM32 and EZR32 Series 0 を使用した開発に必要なものがすべて揃っています。
- 学習内容 :
  - レジスタの基本操作
  - emlib 関数の使用
  - LED の点滅とボタンの読み取り
  - LCD コントローラ
  - エネルギー・モード
  - リアルタイム・カウンタの操作



第 1 章　デバイスの互換性

このアプリケーション・ノートは複数のデバイス・ファミリを対象にしています。一部の機能はデバイスによって異なります。

MCU Series 0 consists of:

- EFM32 Gecko (EFM32G)
- EFM32 Giant Gecko (EFM32GG)
- EFM32 Wonder Gecko (EFM32WG)
- EFM32 Leopard Gecko (EFM32LG)
- EFM32 Tiny Gecko (EFM32TG)
- EFM32 Zero Gecko (EFM32ZG)
- EFM32 Happy Gecko (EFM32HG)

Wireless MCU Series 0 consists of:

- EZR32 Wonder Gecko (EZR32WG)
- EZR32 Leopard Gecko (EZR32LG)
- EZR32 Happy Gecko (EZR32HG)

第 2 章　はじめに

2.1　前提条件

このアプリケーション・ノートのサンプルを使用するには、サポート対象の Silicon Labs デバイスにアクセスできる必要があります。サポート対象のデバイスには、EFM32 Series 0 スターター・キットが含まれています。このチュートリアルを始める前に、以下のいずれかを行って対応する IDE を使用できるよう準備してください。

- Silabs.com (http://www.silabs.com/simplicity) から Simplicity Studio をインストールする、または
- IAR を使用する場合は、キットをサポートするために最新の Segger J-Link ドライバをインストールする (https://www.segger.com/jlink-software.html)

Simplicity Studio のメイン・ウィンドウの左上の [Update Software（ソフトウェアの更新）] ボタンをクリックして、利用可能なすべてのパッケージがインストールされていて最新であることを確認してください。

Note: Simplicity Studio にはフル機能の統合 IDE が含まれているだけでなく、IAR などのサードパーティ製 IDE も一部サポートしています。このため、これらのサンプルを簡単に利用し、さらに EFM32 and EZR32 Series 0 のソリューションの開発に役立つすべての機能にアクセスできるように、どの IDE を使用するかにかかわらず、Simplicity Studio をインストールすることをお勧めします。

2.2 このアプリケーション・ノートの使い方

このアプリケーション・ノートのソース・コードは、ルート・フォルダ an0009_efm32_getting_started にあります。

Simplicity Studio の [Getting Started（スタート・ガイド）] タブの [Application Notes（アプリケーション・ノート）] ダイアログ・ボックスを使用すると、ソース・コードやプロジェクトのサンプルに簡単にアクセスできます。どのアプリケーション・ノートをクリックしても [Application Notes（アプリケーション・ノート）] ダイアログが開きます。

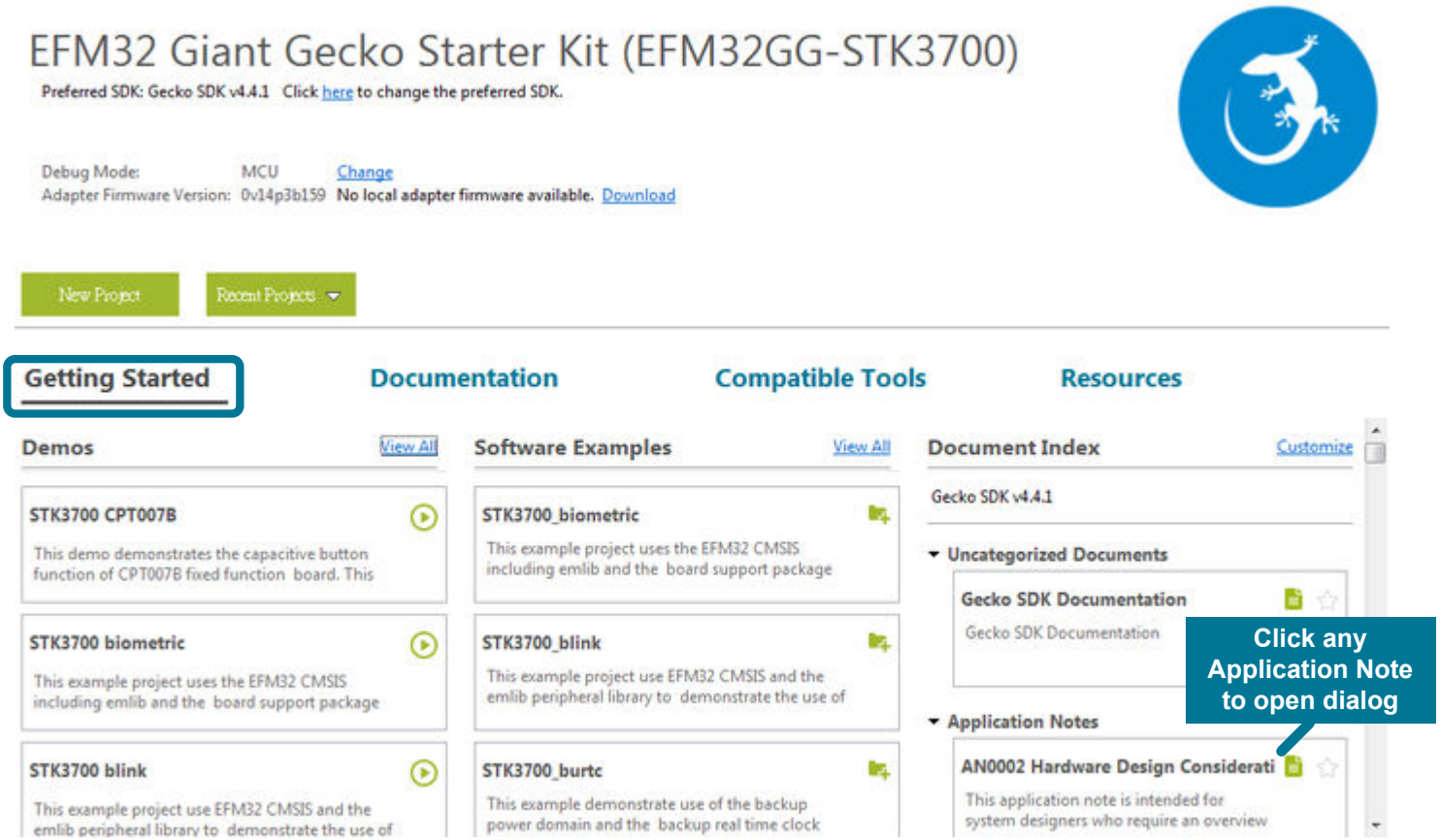


図 2.1. Simplicity Studio のアプリケーション・ノート

[Application Notes（アプリケーション・ノート）] ダイアログで、[AN0009 Getting Started with EFM32（AN0009 EFM32 スタート・ガイド）] エントリを探して選択し、[Import Project...（プロジェクトのインポート...）] ボタンをクリックして利用可能なサンプル・プロジェクトのリストを表示します。プロジェクト名を確認してキットと互換性のあるサンプルを探し、プロジェクトを 1 つ選択して IDE にインポートします（デフォルトでは Simplicity IDE ですが、この構成を必要に応じて別の IDE に変更できます）。

また、複数の IDE 向けのプロジェクトは、iar、arm など、これらのサンプルをホストしているファイルシステム内の個別のフォルダにあり、Simplicity Studio の [Applications Notes（アプリケーション・ノート）] ダイアログの [Open Folder（フォルダを開く）] ボタンを使用してアクセスできます。これらのプロジェクトは、適切な IDE に手動でロードできます。また、すべての IAR プロジェクトが efm32.eww という 1 つの共通ワークスペースに集められます。プロジェクトは、各種キットごとに多少異なるため、必ず使用するキットの名前が先頭についているプロジェクトを開いてください。

Note: このアプリケーション・ノートのコード・サンプルは完成版ではありません。演習を行いながら少しずつコードを書き込んでいく必要があります。また、サンプルごとに完成版のコード・ファイル（末尾に *_solution.c が付いている）もあります。

第 3 章　レジスタの操作

この章では、[CMSIS] と [emlib] のソフトウェア・ライブラリに用意されている定義とライブラリ関数を使用して EFM32 and EZR32 Series 0 デバイスの C コードを記述する基本的な方法について説明します。

3.1　アドレス

EFM32 and EZR32 Series 0 デバイスは、複数の異なるペリフェラル（CMU、RTC、ADC...）で構成されています。デバイスのペリフェラルの中には、クロック管理ユニット（CMU）のように、1 つのインスタンスとしてのみ存在するものがあります。タイマ（TIMERn）などのペリフェラルは複数のインスタンスとして存在し、名前の末尾にはインスタンス番号を示す数字（n）が付いています。通常、ペリフェラルの 2 つのインスタンスは同等ですが、メモリ・マップの異なる領域に配置されます。ただし、一部のペリフェラルは、インスタンスごとに異なる機能セットを備えています。たとえば、USART0 には IrDA インターフェイスがありますが、USART1 にはありません。このような相違点については、デバイスのデータシートとリファレンス・マニュアルに記載されています。

EFM32 and EZR32 Series 0 ペリフェラルはメモリ・マップされます。このため、ペリフェラル・インスタンスにはそれぞれ、読み取り/書き込み操作でアクセス可能なレジスタを含む専用のアドレス領域があります。ペリフェラル･インスタンスとメモリ領域については、デバイスのデータシートに記載されています。ペリフェラル・インスタンスの開始アドレスは、ベース・アドレスと呼ばれます。デバイス・シリーズのリファレンス・マニュアルには、各ペリフェラル内のレジスタの詳しい説明が記載されています。各レジスタのアドレスは、ペリフェラル・インスタンスのベース・アドレスからのオフセットとして指定されます。

3.2 レジスタの説明

EFM32 and EZR32 Series 0 デバイスでは、ペリフェラルへの書き込み/読み取りアクセスに 32 ビット・バスを使用し、ペリフェラル内の各レジスタには、0 ～ 31 の番号が付いた 32 個のビットがあります。未使用のビットは予約済みとして指定されています。修正しないでください。ペリフェラルが使用するビットはシングル・ビット（下図の OUTEN ビットなど）か、またはビット・フィールドでグループ化できます（下図の PRSSEL ビット・フィールドなど）。各ビット・フィールドは、以下の属性で記述されます。

- ビット位置
- 名前
- リセット値
- アクセス・タイプ
- 説明

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.

図 3.1. サンプル・レジスタの説明

3.3 アクセス・タイプ

各レジスタには、そのレジスタ内のすべてのビット・フィールドのアクセス・タイプが設定されています。アクセス・タイプとは、ビット・フィールドに対する読み取りまたは書き込み操作への反応を記述したものです。以下の表は、デバイス内のレジスタのさまざまなアクセス・タイプを説明しています。

表 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 および emlib

Cortex マイクロコントローラ・ソフトウェア・インターフェイス規格（CMSIS）は、すべての ARM Cortex デバイスに共通のコーディング規格です。Silicon Labs が提供している CMSIS ライブラリには、ヘッダー・ファイル、定義（ペリフェラル、レジスタ、およびビット・フィールド用）、およびすべてのデバイスのスタートアップ・ファイルが含まれます。さらに CMSIS には、すべての Cortex デバイスに共通する機能（割り込み処理や組み込み関数など）も含まれています。ハード・コードされたアドレスとデータ値を使用してレジスタに書き込むことができますが、コードの移植性と読みやすさを確保するために定義の使用をお勧めします。

これらの定義を使用するには、プロジェクトが c ファイルに em_device.h をインクルードする必要があります。これは、すべての EFM32 and EZR32 Series 0 デバイスに共通のヘッダ・ファイルです。このファイル内で、該当するデバイスのヘッダ・ファイルの内容が、プロジェクトに定義されているプリプロセッサ・シンボルに従ってプロジェクト・ビルドにインクルードされます。

EFM32 and EZR32 Series 0 デバイスのプログラミングを簡素化するために、Silicon Labs では [emlib] という完全な C 関数ライブラリを開発・管理しています。このライブラリにより、デバイスのすべてのペリフェラルとコア機能に対する効率的かつ明確で堅固なアクセスと制御が可能になります。このライブラリは、以下のパス（ここで、v1.1 は Gecko SDK スイートのバージョン番号）にある emlib フォルダ内の em_xxx.c（em_dac.c など）ファイルと em_xxx.h（em_dac.h など）ファイル内にあります。

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

このアプリケーション・ノートに付属するソース・ファイルには、それぞれ em_chip.h がインクルードされていて、CHIP_Init() の呼び出しは、すべての main() 関数の先頭付近にあります。em_device.h の内容と同様に、CHIP_Init() 関数内で行う操作は、使用する具体的な部品によって異なりますが、既知のエラッタの修正を含めるか、それ以外の場合はデバイス全体の動作の一貫性を確保してください。したがって、CHIP_Init() 関数を実行する前に、メイン関数内でコードを実行しないでください。

3.4.1 CMSIS ドキュメント

EFM32 and EZR32 Series 0[CMSIS] ライブラリと [emlib] に関する詳しい Doxygen ドキュメントは、Simplicity Studio メイン・ウィンドウの [Documentation（ドキュメント）]の [Gecko SDK Documentation（Gecko SDK ドキュメント）]ボックスで対応するデバイスまたはキットを選択すると入手できます。このドキュメントは、Silicon Labs ウェブサイト（http://devtools.silabs.com/dl/documentation/doxygen/）または GitHub（https://siliconlabs.github.io/Gecko_SDK_Doc/）からも入手できます。

3.4.2 ペリフェラル構造体

emlib ヘッダ・ファイルでは、以下のサンプルのように、ペリフェラル・タイプごとのレジスタ定義が構造体にグループ化されています。

```
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;
```

レジスタ・アドレスは、ペリフェラル・インスタンスのベース・アドレスと追加オフセットで構成されています。[emlib] のペリフェラル構造体はレジスタへの書き込みを簡素化し、基礎となるこれらのアドレスとオフセットを抽象化します。したがって、DAC0 ペリフェラル・インスタンスの CH0DATA への書き込みは、以下のように行うことができます。

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

同様に、レジスタの読み取りは、以下のように行うことができます。

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

3.4.3 ビット・フィールド定義

すべてのデバイスには、ペリフェラルごとに関連ビット・フィールドが定義されています。これらの定義は efm32xx_xxx.h（efm32tg_dac.h など）ファイル内にあり、該当する [emlib] ペリフェラル・ヘッダ・ファイルに自動的にインクルードされます。

```
#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)
```

すべてのレジスタ・ビット・フィールドについて、関連シフト、マスク、デフォルト値のビット・フィールドも定義されています。

```
#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 レジスタ・アクセスの例

制御レジスタのビットを設定する際、ファームウェアでレジスタの他のビットを誤ってクリアしないようにする必要があります。このためには、以下の例に示すように、ビット・ファームウェアを含むマスクを、元の内容との論理和を取れるように設定する必要があります。

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

これをまとめて表記すると、次のようになります。

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

ビットをクリアする場合は、レジスタで、クリアするビットを除き、設定するすべてのビットと値の論理積を取ります。

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

複数のビットを含むビット・フィールドに新しい値を設定する場合、単純な OR 関数は使用しないでください。マスクとの論理和を取った元のビット・フィールドの内容から、間違った結果が生成される恐れがあります。代わりに、新しい値で論理和を取る前にビット・フィールド全体（かつ、そのビット・フィールドのみ）を必ずクリアしてください。

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

3.4.5 グループ化されたレジスタ

各ペリフェラル内で、一部のレジスタがグループ化されています。このようなグループの例としては、下図のデータ出力レジスタ (DOUT) など、各 GPIO ポートに関連付けられたレジスタがあります。各 GPIO ポート（A、B、C ...）には、DOUT レジスタが含まれ、以下の説明はこれらすべてに共通しています。GPIO_Px_DOUT の x は、ポートのワイルド・カードを示しています。

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 Data output on pin.	0x0000	RW	**Data Out**

図 3.2. GPIO のグループ化されたレジスタ

CMSIS 定義で、ポート・レジスタは配列 P[x] でグループ化されています。この配列を使用する場合は、ポート文字の代わりに番号を使用してインデックス付けする必要があります（A=0、B=1、C=2...）。ポート C の DOUT レジスタへのアクセスは次のように行います。

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

第 4 章　例 1 - レジスタの操作

この例では、CMSIS 定義を使用したレジスタの書き込み/読み取り方法を示します。また、このチュートリアルでは、Simplicity Studio や IAR Embedded Workbench でデバッガを使用してレジスタの内容を監視し、操作する方法についても示します。この例では Simplicity Studio と IAR のみについて示しますが、これら以外のサポート対象の IDE でもこの作業を行うことができます。

Simplicity Studio の場合：

1. キットを PC に接続し、Simplicity Studio を開きます。
2. キットが表示されたら、[Device（デバイス）] ウィンドウで EFM32 Giant Gecko スターター・キット (EFM32GG-STK3700) などのキットをクリックします。
3. [Getting Started（スタート・ガイド）] タブの下でいずれかのアプリケーション・ノートをクリックし、[Application Notes（アプリケーション・ノート）] ダイアログを開きます。
4. [AN0009 Getting Started with EFM32 (AN0009 EFM32 スタート・ガイド）] を探して、リスト内のドキュメントをクリックし、[Import Project...（プロジェクトのインポート）] ボタンをクリックします。
5. ダイアログで [<kit_name>_1_registers.slsproj] オプションを選択し、[OK] をクリックします。
6. 1_registers.c ファイルをダブルクリックして、エディタ・ビューにファイルを開きます。カスタム・コードを追加する位置にマーカーがあります。

IAR の場合：

1. efm32 ワークスペースを開きます (an¥an0009_efm32_getting_started¥iar¥efm32.eww)。
2. IAR Embedded Workbench で [<kit_name>_1_registers] プロジェクトを選択します。
3. ソース・ファイル内の 1_registers.c のメイン関数のカスタム・コードを追加する位置にマーカーがあります。

4.1　ステップ 1 - タイマ・クロックの有効化

この例では、TIMER0 を使用します。高周波数 RC 発振器 (HFRCO) はデフォルトの周波数帯域で動作しますが、すべてのペリフェラル・クロックは無効になっています。したがって、使用する前に TIMER0 のクロックを有効にする必要があります。リファレンス・マニュアルの CMU に関する章を参照すると、CMU ペリフェラルの HFPERCLKEN0 レジスタの TIMER0 ビットを設定して、TIMER0 のクロックを有効にできることがわかります。

4.2　ステップ 2 - タイマの起動

タイマを起動するには、TIMER0 の CMD レジスタで START ビットに 1 を書き込みます。

4.3　ステップ 3 - 待機しきい値

カウンタが 1000 になり、次に進むまで待機する while ループを作成します。

4.4 観察

Simplicity IDE の場合、左側の [Project（プロジェクト）] ビューでプロジェクトをクリックして、[<kit_name>_1_registers] プロジェクトがアクティブであることを確認します。次に、[Debug（デバッグ）] ボタンを押し、コードを自動的にビルドしてデバイスにダウンロードします。[Registers（レジスタ）] ビューをクリックし、TIMER0 の STATUS レジスタを探します。レジスタを展開して、RUNNING ビットが 0 に設定されていることを確認してください。

コード・ビューの左側のペインをダブルクリックして、ファームウェアがタイマを起動する前の位置にブレークポイントを配置し、[Resume（再開）] ボタンをクリックします。次に、[Step Over（ステップ・オーバー）] ボタンを使用して式を 1 回ステップ・オーバーして、[Registers（レジスタ）] ビューで RUNNING ビットが 1 に設定されていることを確認します。シングル・ステップを続行すると、CNT レジスタの値が増加することがわかります。[Registers（レジスタ）] ビューで直接入力して、CNT レジスタに別の値を書き込んでみてください。

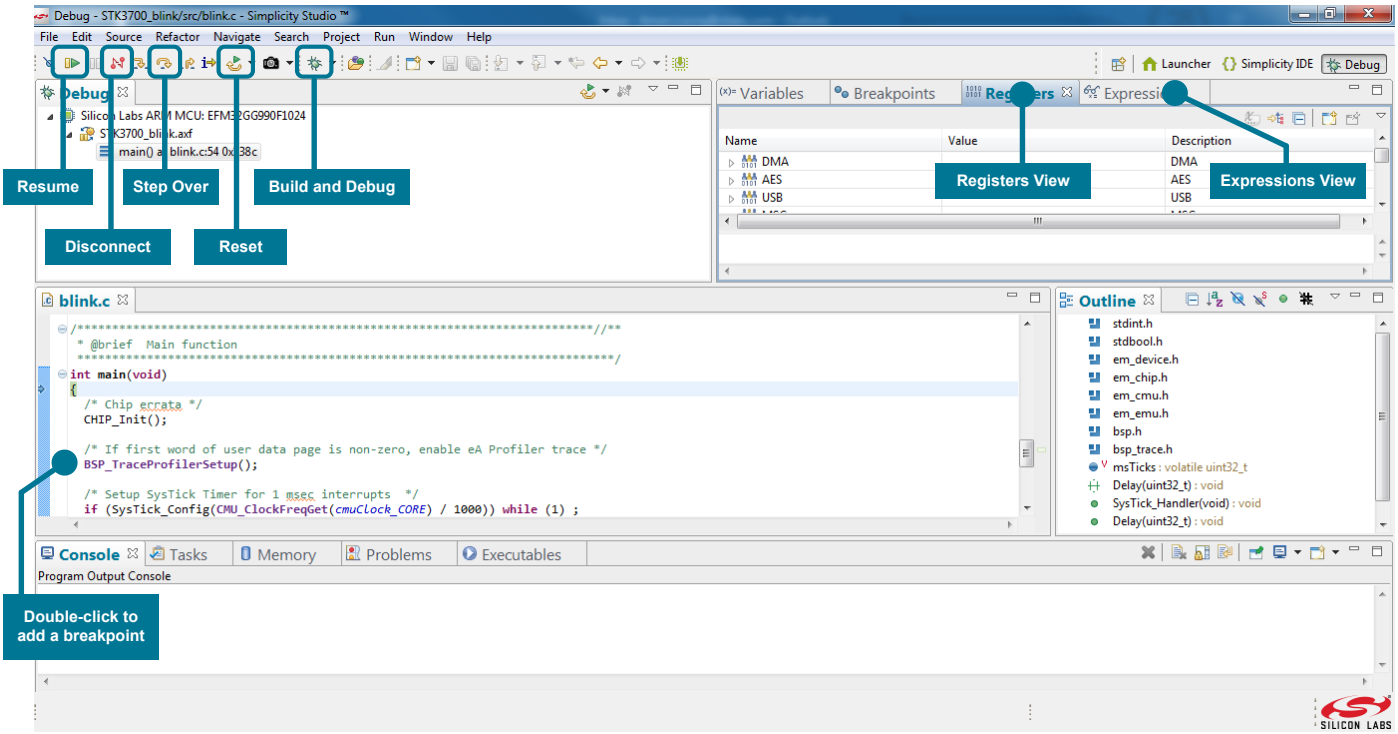


図 4.1. Simplicity IDE のデバッグ・ビュー

IAR の場合は、ワークスペース・ウィンドウの下部の対応するタブを押して、[<kit_name>_1_registers] プロジェクトがアクティブになっていることを確認します。次に、[Download & Debug（ダウンロードとデバッグ）] ボタンを押して、[View（表示）]->[Register（レジスタ）] の順に進み、TIMER0 の STATUS レジスタを探します。レジスタを展開して、RUNNING ビットが 0 に設定されていることを確認してください。

ファームウェアがタイマを起動する行の前にカーソルを置き、[Run to Cursor（カーソル位置まで実行）] を押します。次に、[Step Into（ステップ・イン）] ボタンをクリックして式に移動し、[Register（レジスタ）] ビューで RUNNING ビットが 1 に設定されていることを確認します。続けて [Step Into（ステップ・イン）] ボタンをクリックし、CNT レジスタの値が増加していることを確認します。[Register（レジスタ）] ビューで直接入力して、CNT レジスタに別の値を書き込んでみてください。

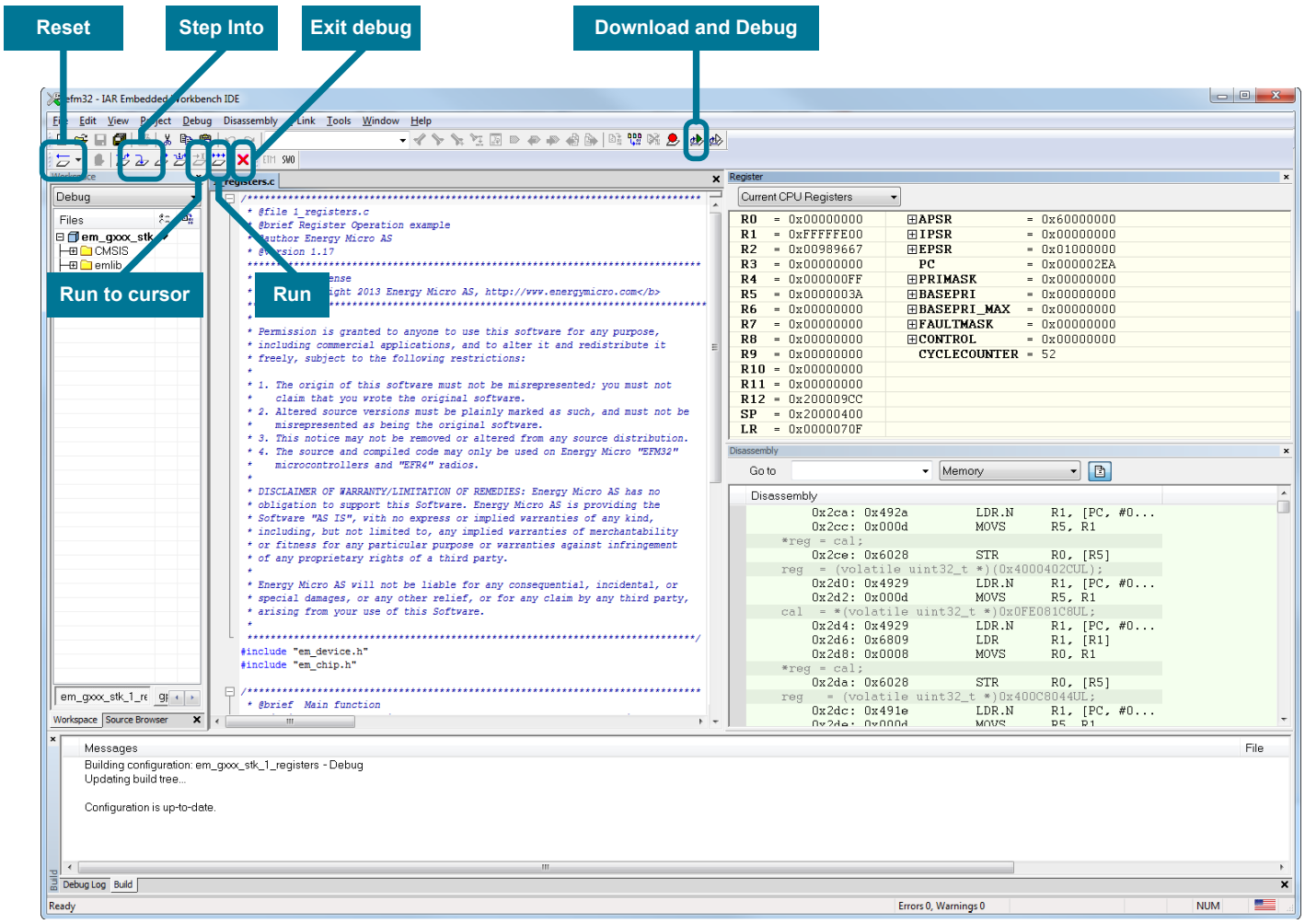


図 4.2. IAR のデバッグ・ビュー

第 5 章　例 2 – STK の LED の点滅

この例では、GPIO ピンを使用して STK の LED を点灯し、ボタンが押されるたびに LED 構成が変わるようにします。レジスタに直接アクセスするのではなく、emlib 関数を使用してペリフェラルを構成します。

AN0009 アプリケーション・ノートには Simplicity Studio の [<kit_name>_2_leds] プロジェクトと IAR efm32 ワークスペースが含まれており、これらをこの例で使用します。emlib C ファイルはプロジェクトにインクルードされます。対応するヘッダ・ファイルは、C ファイルの先頭でインクルードされます。

どのような emlib 関数があるのかと、それらの使用方法の詳細を確認するには、Simplicity Studio の [Documentation（ドキュメント）] の下にある [Gecko SDK Documentation（Gecko SDK ドキュメント）] を使用して API ドキュメントを開いて参照してください（または http://devtools.silabs.com/dl/documentation/doxygen/ にアクセスしてください）。適切なデバイス（Giant Gecko など）のソフトウェア・ドキュメント・リンクをクリックし、[Modules（モジュール）->EMLIB] を開いて [CMU] ペリフェラルを選択します。ウィンドウの右上にある [Functions（関数）] リンクをクリックして、このペリフェラルの関数のリストを探します。これらの関数を使用すると、クロック管理ユニットを容易に操作できます。

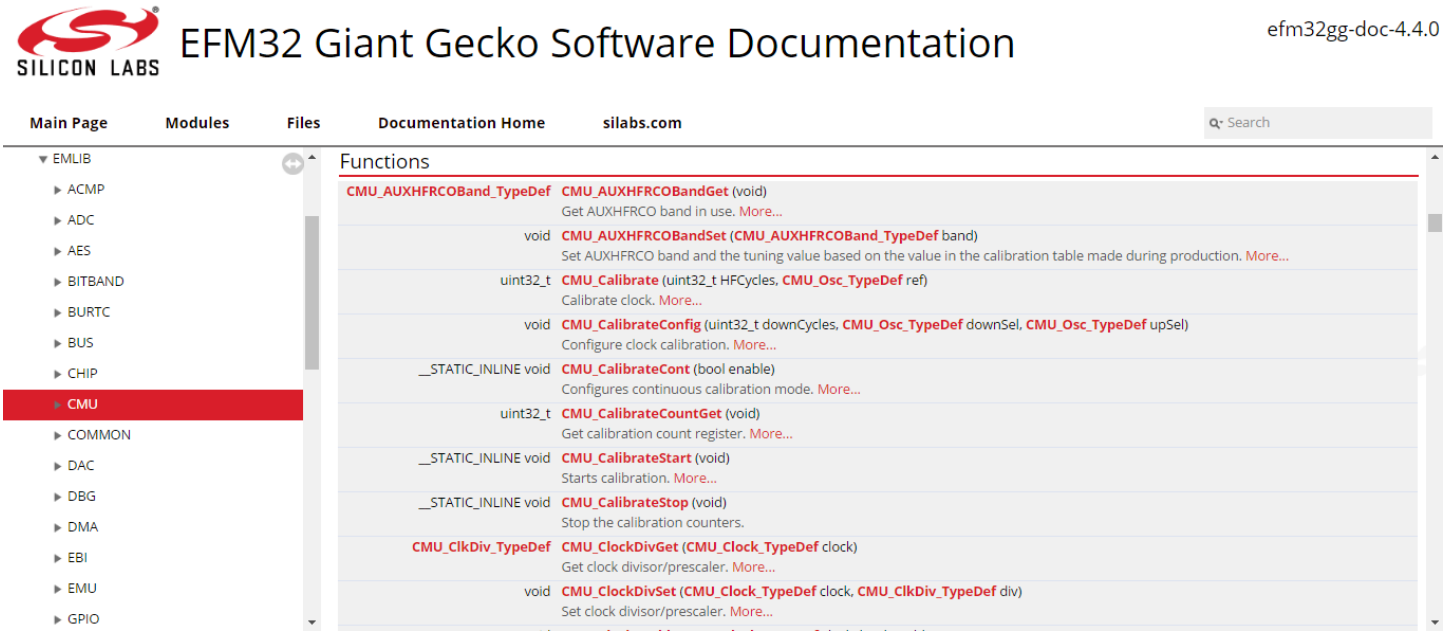


図 5.1.　CMU に固有の emlib 関数のドキュメント

5.1 ステップ 1 – GPIO クロックの有効化

CMU 関数のリストには、GPIO のクロックを有効にする以下の関数があります。

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

関数をクリックすると、その関数の使い方の説明が表示されます。[CMU_Clock_TypeDef] リンクをクリックすると、クロック引数に使用可能な列挙子のリストが表示されます。GPIO を有効にするには、以下を追加します。

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

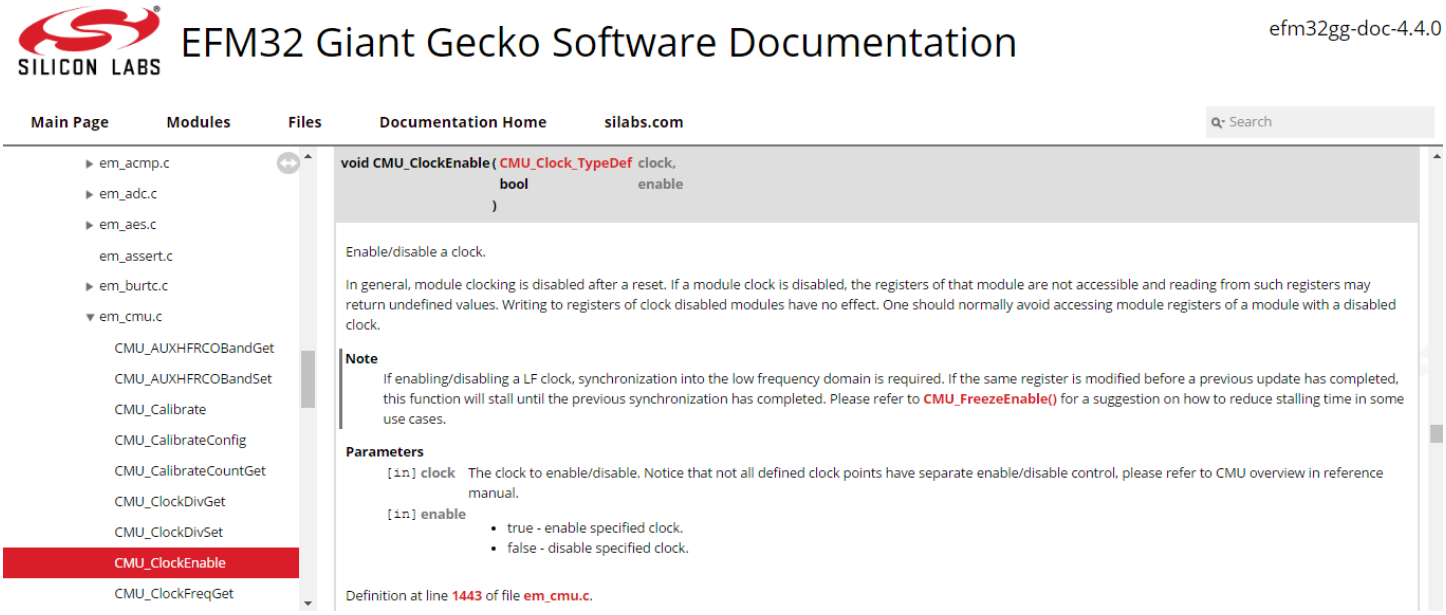


図 5.2. CMU_ClockEnable 関数の説明

5.2 ステップ 2 – LED の GPIO ピンの構成

[Device (デバイス)] ウィンドウまたは [Solutions (ソリューション)] ウィンドウで対応するキットを選択している場合、そのキットのユーザ・マニュアルは、Simplicity Studio の [Documentation (ドキュメント)] から入手可能です。このドキュメントには、ユーザ LED を含むすべてのピンの使用法が説明されています。次の EFM32 Series 0 キットでは、これらの LED は以下のように接続されています。

- EFM32-Gxxx-STK: 4 LEDs on port C, pins 0-3
- EFM32TG-STK3300: 1 LED on port D, pin 7
- EFM32GG-STK3700: 2 LEDs on port E, pins 2-3
- EFM32ZG-STK3200: 2 LEDs on port C, pins 10-11

使用するキットの情報については、ユーザ・マニュアルを参照してください。

GPIO に使用できる関数のうち、以下の関数を使用すると、GPIO ピンのモードを構成できます。

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

この関数を使用して、LED ピンをプッシュプル出力として構成し、DOUT 初期値を 0 に設定します。

5.3 ステップ 3 – ボタンの GPIO ピンの構成

STK のユーザ・マニュアルを参照して、使用しているキットでプッシュ・ボタン 0 (PB0) が接続されている場所を探します。このボタンは、次のサンプル・キットでは以下のように接続されています。

- EFM32-Gxxx-STK: Port B, pin 9
- EFM32TG-STK3300: Port D, pin 8
- EFM32GG-STK3700: Port B, pin 9
- EFM32ZG-STK3200: Port C, pin 8

ボタンの状態を検出できるように、このピンを入力として構成します。

5.4 ステップ 4 – ボタンが押されたときの LED ステータスの変更

PB0 が押されるたびに LED を切り替えるループを作成します。ボタンが押されるたびに LED が 1 回だけ切り替わるように、ボタンが押されたことだけでなく、ボタンから指が離れたこともチェックするようにします。PB0 は、外部レジスタによって High に設定されます。

5.5 その他のタスク – LED アニメーション

LED がゆっくり点灯/消灯する、連続して点灯する（複数の LED がある場合）など、LED のさまざまな点滅パターンを作成してみましょう。デバイスは一般的に、デフォルトで HFRCO 周波数帯域で動作するため、LED がリアルタイムで変化することを確認できるように遅延関数を追加します。例 1 の TIMER0 のコードを使用し、次の関数を作成します。

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

TIMER0_CTRL の PRESC ビット・フィールドを使用してクロック周波数を目的の値まで下げます。

第 6 章　例 3a – セグメント LCD コントローラ

この例では、セグメント LCD のある STK が必要です。この例では、セグメント LCD コントローラを使用して LCD ディスプレイに情報を表示する方法を説明します。LCD コントローラには自律アニメーション機能があり、この機能についても説明します。AN0009 アプリケーション・ノートには、Simplicity Studio の [<kit_name>_3_lcd] プロジェクトと IAR efm32 ワークスペースが含まれており、これらをこの例で使用します。

6.1　ステップ 1 – LCD コントローラの初期化

LCD コントローラ・ドライバは、スターター・キット・ライブラリにあります。まず、segmentlcd.h にある SegmentLCD_Init() 初期化関数を実行して LCD コントローラを設定します。

6.2　ステップ 2 – LCD ディスプレイへの書き込み

デフォルトでは、すべての LCD セグメントは初期化後にオフに切り替わります。LCD コントローラ・ドライバには、ディスプレイ上のさまざまなセグメント・グループを制御する関数がいくつか含まれています。その例の一部を以下に示します。

```
void SegmentLCD_Number(int value)
void SegmentLCD_Write(char *string)
void SegmentLCD_Symbol(lcdSymbol s, int on);
```

ディスプレイ上にカスタムのテキスト、数字、記号を配置し、これらを少し動かしてみましょう。例 2 の遅延関数を使用します。遅延関数は、ソリューション・ファイルにもあります。

6.3 ステップ 3 – セグメントのアニメーションを作成

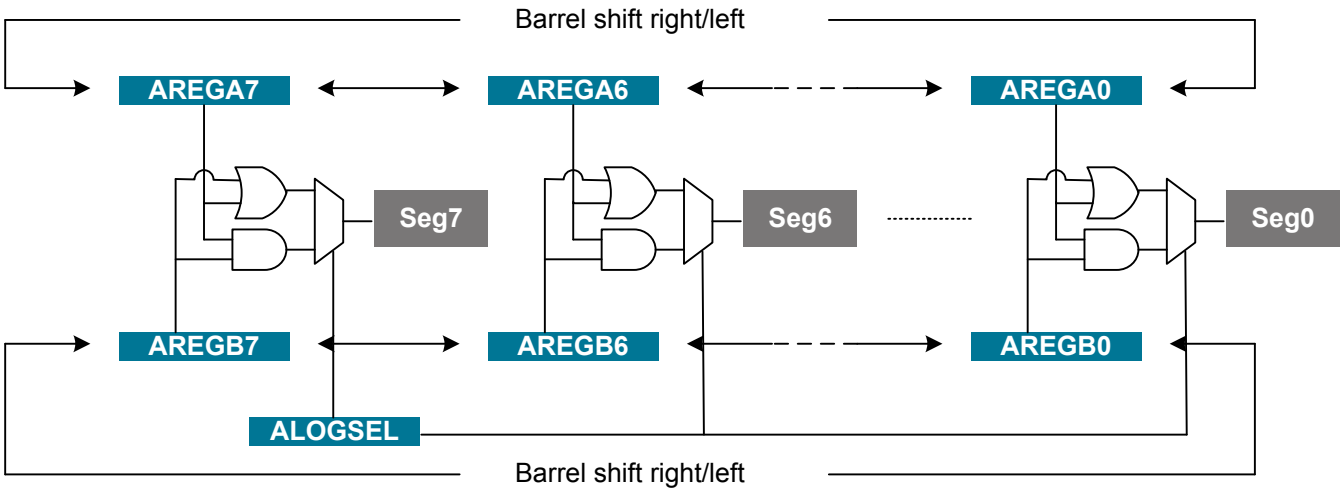


図 6.1. アニメーション機能

LCD コントローラには、最大で 8 個のセグメント（LCD ディスプレイに 8 セグメント・リングを表示）のアニメーションを自律的に動かすことができるアニメーション機能があります。アニメーション・セグメントに表示されるデータは、セグメントごとに 2 個のレジスタ・ビットの論理関数（AND または OR）です。2 個のレジスタ配列（LCD_AREGA、LCD_AREGB）は、フレーム・カウンタがオーバーフローするたびに左または右にシフトするバレルになるように設定することができます。フレーム・カウンタは、構成可能なフレーム数を超えるとオーバーフローするように設定できます。ファームウェアは、レジスタに直接書き込むことで LCD レジスタを操作します。以下のレジスタを設定する必要があります。

LCD_BACTRL：

- Frame Counter Enable（フレーム・カウンタ有効化）ビットを設定する
- FCTOP フィールドを設定してフレーム・カウンタ期間を構成する
- Animation Enable（アニメーション有効化）ビットを設定する
- AND または OR を論理関数として選択する
- AREGA と AREGB のシフト方向を構成する
- Giant Gecko STK（STK3700）の場合は、ALOC ビットも SEG8TO15 に設定する

LCD_AREGA/LCD_AREGB：

- アニメーションに使用するデータをこれらのレジスタに書き込む。

アニメーション・セグメントの構成を使用してビットを操作し、LCD ディスプレイに表示される結果を確認します。

第 7 章　例 3b – メモリ LCD

この例では、メモリ LCD のある STK が必要です（Happy Gecko、Zero Gecko、Pearl Gecko、EFR32xG ワイヤレス STK など）。この例では、メモリ LCD ドライバを構成してメモリ LCD にテキストを書き込む方法を説明します。この例では、[<kit_name>_3_lcd_mem] というソフトウェア・プロジェクトを使用します。

Gecko SDK には、メモリ LCD 用のドライバが含まれています。このディスプレイ・ドライバのドキュメントは、対応するキットの [Software Documentation（ソフトウェア・ドキュメント）] の [Kit Driver（キット・ドライバ）] セクションの下にある [Display（ディスプレイ）] にあります。

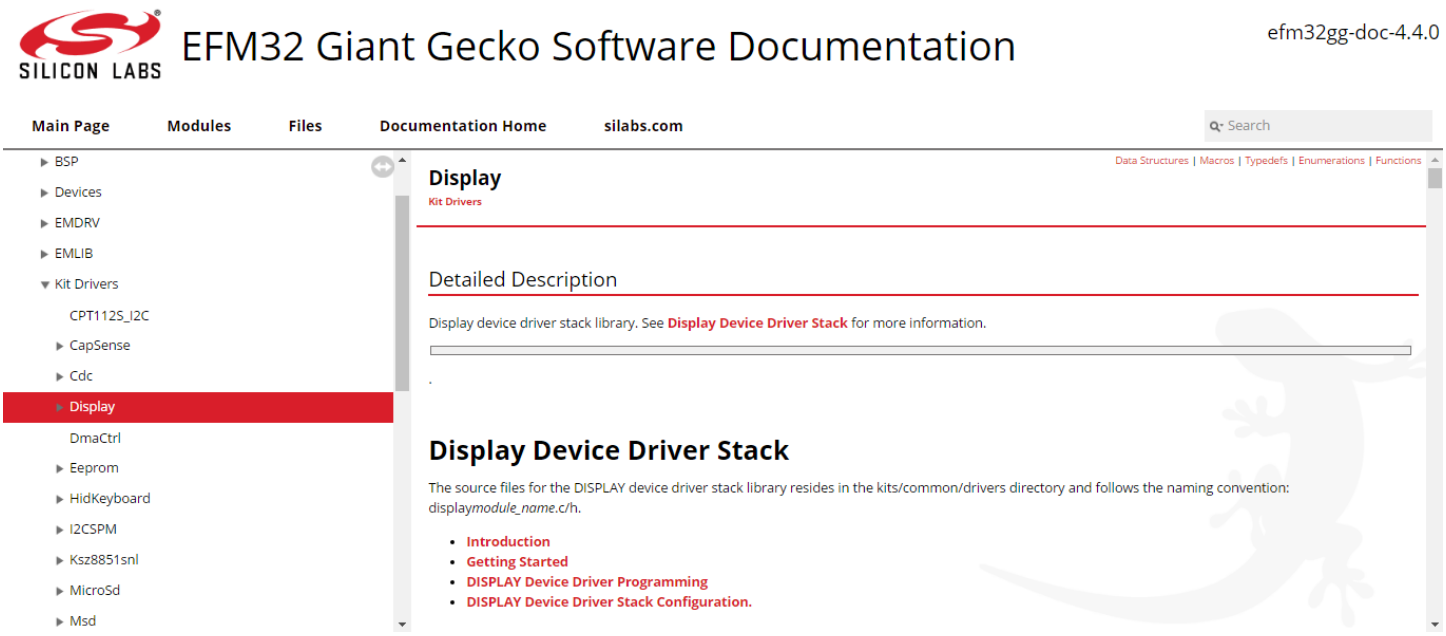


図 7.1.　ディスプレイ・ドライバのドキュメント

7.1　ステップ 1 – ディスプレイ・ドライバの構成

まず、DISPLAY_Init() を使用してディスプレイ・ドライバを初期化します。

ディスプレイ・ドライバには、ディスプレイ・デバイスにテキストを出力するためのインターフェイスである TEXTDISPLAY が含まれています。TEXTDISPLAY_New() を使用して新しい TEXTDISPLAY インターフェイスを作成します。

7.2　ステップ 2 – メモリ LCD にテキストを書き込む

TEXTDISPLAY は、メモリ LCD にテキストを書き込むための基本関数を実装します。TEXTDISPLAY_WriteString() と TEXTDISPLAY_WriteChar() を試してみましょう。

第 8 章　例 4 – エネルギー・モード

この例では、さまざまなエネルギー・モード（EMx）に切り替え、RTC 割り込みや RTCC 割り込みを使用してウェイク・アップさせる方法を説明します。この例では、[<kit_name>_4_energymodes] というプロジェクトを使用します。

8.1　STK の高度エネルギー・モニタ

スターター・キットには、VMCU 電源ドメインの電流測定機能が含まれています。この電源ドメインは、スターター・キットのアプリケーション部分でデバイスや LCD ディスプレイ、その他のコンポーネントに電力を供給するために使用されます。リアルタイムの電流測定は、Simplicity Studio に用意されている Energy Profiler を使用して PC 上で監視できます。

8.2　ステップ 1 – EM1 への切り替え

EM1 に切り替えるには、SCB_SCR レジスタ・クリアの SLEEPDEEP ビットを使用して割り込み待ち命令を実行します。この命令の組み込み関数（CMSIS の一部）を以下に示します。

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

この命令の実行後、消費電流の低下を観察します。

また、EMU の emlib 関数には、エネルギー・モードに切り替えるための関数が含まれています。これらの関数をファームウェアで使用することで、SLEEPDEEP ビットをクリアして WFI 命令を手動で実行する必要がなくなります。

```
void EMU_EnterEM1()
```

8.3　ステップ 2 – EM2 への切り替え

EM2 への切り替えも WFI 命令を実行して行いますが、SCB_SCR レジスタの SLEEPDEEP ビットも設定し、低周波数発振器も有効にします。EM2 に切り替えるには、ディープ・スリープに移行する前に、まず、低周波数発振器（LFRCO または LFXO）を有効にします。この例では、次の emlib 関数を使用して LFRCO を有効にして安定化するまで待ちます。

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

次に SLEEPDEEP を設定し、WFI 命令を実行して EM2 に切り替えます。

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

また、EMU の emlib 関数には、エネルギー・モードに切り替えるための関数が含まれています。これらの関数をファームウェアで使用することで、SLEEPDEEP ビットを設定して WFI 命令を手動で実行する必要がなくなります。

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

この関数を利用することを強くお勧めします。また、この emlib 関数を使用すると、エネルギー・モードの動作に影響を与えるエラッタの問題も回避できます。

Note: アクティブなデバッグ・セッション中、デバイスは EM1 を下回ることはできません。EM2 で消費電流を測定するには、デバッグ・セッションを終了し、STK のリセット・ボタンを使用してデバイスをリセットしてください。

8.4 ステップ 3 – EM3 への切り替え

EM3 に切り替えるには、まず、ディープ・スリープに移行する前にすべての低周波数発振器を無効にします（EM2 と同じ方法で行います）。以下の emlib 関数を使用して、ステップ 2 で有効にした LFRCO を無効にします。

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

次に SLEEPDEEP を設定し、WFI 命令を実行して EM3 に切り替えます。

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

また、EMU の emlib 関数には、エネルギー・モードに切り替えるための関数が含まれています。これらの関数をファームウェアで使用することで、LF 発振器を無効にし、SLEEPDEEP ビットを設定し、WFI 命令を手動で実行する必要がなくなります。

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

この関数を利用することを強くお勧めします。また、この emlib 関数を使用すると、エネルギー・モードの動作に影響を与えるエラッタの問題も回避できます。

Note: アクティブなデバッグ・セッション中、デバイスは EM1 を下回ることはできません。EM3 で消費電流を測定するには、デバッグ・セッションを終了し、STK のリセット・ボタンを使用してデバイスをリセットしてください。

8.5 ステップ 4 – MCU Series 0 のリアルタイム・カウンタ（RTC）の構成

EM2 からウェイク・アップするには、5 秒後に割り込みが入るようにリアルタイム・カウンタ（RTC）を構成します。まず、CMU の emlib 関数を使用して、RTC のクロックを有効にします。RTC などの低エネルギー/周波数のペリファラルと通信するには、LE インターフェイス（cmuClock_HFLE）のクロックも有効にします。

RTC の emlib 初期化関数には、入力として構成構造体が必要です。

```
void RTC_Init(const RTC_Init_TypeDef *init)
```

この構造体はすでにコードで宣言されていますが、これを RTC_Init 関数とともに使用するには、ファームェアでこの構造体に 3 つのパラメータを設定する必要があります。

```
rtcInit.comp0Top = true;
```

次に、RTC でコンペア値 0（COMP0）を設定します。このコンペア値は、コンペア値がカウンタ値と一致したときに割り込みフラグ COMP0 を設定します。RTC が 32.768 kHz で動作するとして、5 秒に相当する値を選択します。

```
void RTC_CompareSet(unsigned int comp, uint32_t value)
```

次に、コンペア・マッチに RTC COMP0 フラグを設定しますが、対応するイネーブル・ビットも、RTC からの割り込み要求を生成するように設定する必要があります。

```
void RTC_IntEnable(uint32_t flags)
```

コンパレータ・マッチで RTC 割り込み要求が有効になっていますが、割り込みをトリガするには、RTC 割り込み要求ラインを Cortex-M で有効にする必要があります。使用する IRQn_Type は RTC_IRQn です。

```
NVIC_EnableIRQ(RTC_IRQn);
```

RTC の割り込みハンドラはコードにすでに含まれていますが（RTC_IRQHandler）、空です。この関数で、RTC COMP0 割り込みフラグをクリアする関数呼び出しを追加します。ファームウェアでこれを行わないと、割り込みがディアサートされないため、Cortex-M は割り込みハンドラで停止します。割り込みフラグをクリアする RTC emlib 関数を探してください。

8.6 追加タスク – EM2 のセグメント LCD コントローラ

追加タスクとして、LCD コントローラを有効にし（使用中のキットにセグメント LCD があると仮定）、EM2 に移動する前に LCD ディスプレイに何か書きます。前述の EM2 の例のセグメント・アニメーションを使用します。

Note: メモリ LCD ディスプレイ・ドライバでは RTC または RTCC が使用されるため、この追加タスクは、セグメント LCD ではなくメモリ LCD を備えているキットには該当しません。

第 9 章　概要

お疲れ様でした。レジスタと GPIO 操作、STK と LCD における基本関数の使用方法に加え、デバイスと emlib/CMSIS 関数のさまざまなエネルギー・モードの処理など、省エネ・プログラミングの基本を理解していただきました。Simplicity Studio の [Getting Started（スタート・ガイド）] タブにある [Software Examples（ソフトウェア例）] と [Application Notes（アプリケーション・ノート）] では、多くの例とアプリケーション・ノートが参照できます。

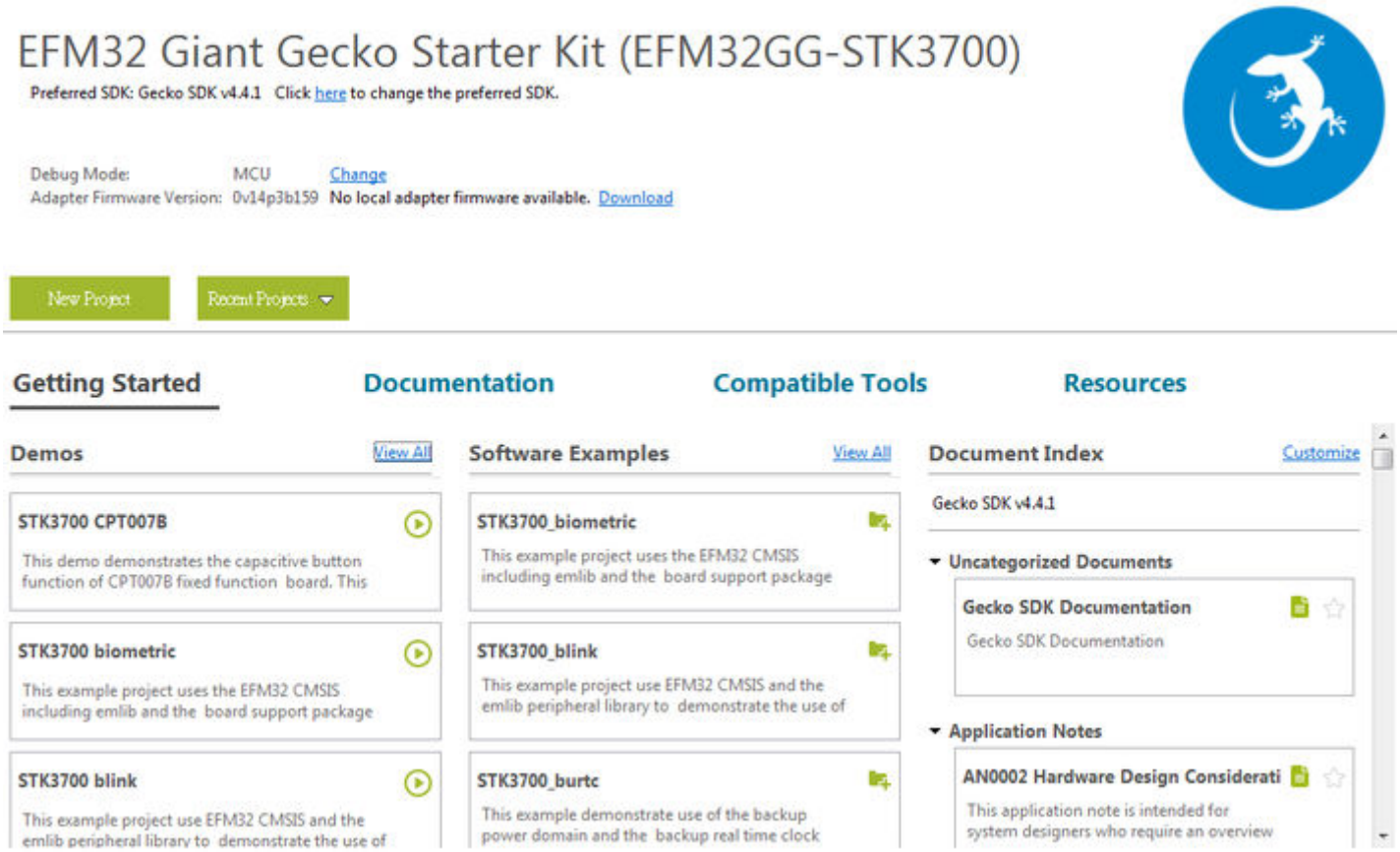


図 9.1.　Simplicity Studio のソフトウェア例とアプリケーション・ノート

第 10 章　修正履歴

改訂 1.22

2018 年 1 月

- EFM32PG13/EFM32JG13 の記述を削除。
- EFM32TG11 スターター・キットの例を追加。

改訂 1.21

2017 年 6 月

- AN0009 を AN0009.0（MCU/ワイヤレス MCU シリーズ 0 用）と AN0009.1（MCU/ワイヤレス SoC シリーズ 1 用）に分割。
- 例 4 テキストから RTC 構成手順を削除。
- EFM32PG12、EFM32GG11、EFR32MG1、EFR32MG12、および EFR32MG13 スターター・キットの例を追加。

改訂 1.20

2017 年 1 月

- Simplicity Studio V4 に合わせて更新。
- EFM32G 開発キットの例を削除。
- EFM32PG1 スターター・キットの例を追加。

改訂 1.19

2017 年 11 月

- EFM32 Gemstones および EFR32 Wireless Gecko のポートフォリオのサポートを追加。

改訂 1.18

2014 年 5 月

- CMSIS 3.20.5 のコード・サンプルを更新
- コード・サンプルで Silicon Labs ライセンスを変更
- Simplicity IDE のサンプル・プロジェクトを追加
- Sourcery CodeBench Lite のサンプル・メイクファイルを削除

改訂 1.17

2013 年 10 月

- 欠落していたヘッダ・ファイルを追加

改訂 1.16

2013 年 10 月

- 新しいカバー・レイアウト
- Zero Gecko スターター・キット (EFM32ZG-STK3200) のサポートを追加
- 「ビット・フィールド定義」の章に新しいテキストを追加
- STK3700 および LCD アニメーションに関する問題を修正

改訂 1.15

2013 年 5 月

- ARM-GCC および Atollic TrueStudio のソフトウェア・プロジェクトを追加。

改訂 1.14

2012 年 11 月

・ Giant Gecko スターター・キット (EFM32GG-STK3700) のサポートを追加
・ 新しいキットドライバと BSP 構成に合わせてソフトウェア・プロジェクトを変更

改訂 1.13

2012 年 4 月

・ 新しいペリフェラル・ライブラリの命名規則と CMSIS_V3 に合わせてソフトウェア・プロジェクトを変更

改訂 1.12

2012 年 3 月

・ efm32lib ファイル efm32_emu.c を LED プロジェクトに追加
・ CodeSourcery プロジェクトのメイクファイルエラーを修正

改訂 1.11

2011 年 10 月

・ 新しいキットのディレクトリに合わせて IDE プロジェクト・パスを更新

改訂 1.10

2011 年 9 月

・ Tiny Gecko スターター・キット (EFM32TG-STK3300) のサポートを追加

改訂 1.03

2011 年 5 月

・ 新しい BSP バージョンに合わせてプロジェクトを更新

改訂 1.02

2010 年 11 月

・ 新しい EFM32LIB 関数のソリューション c ファイルを修正
・ 新しい EFM32LIB 関数の PDF ドキュメントを修正

改訂 1.01

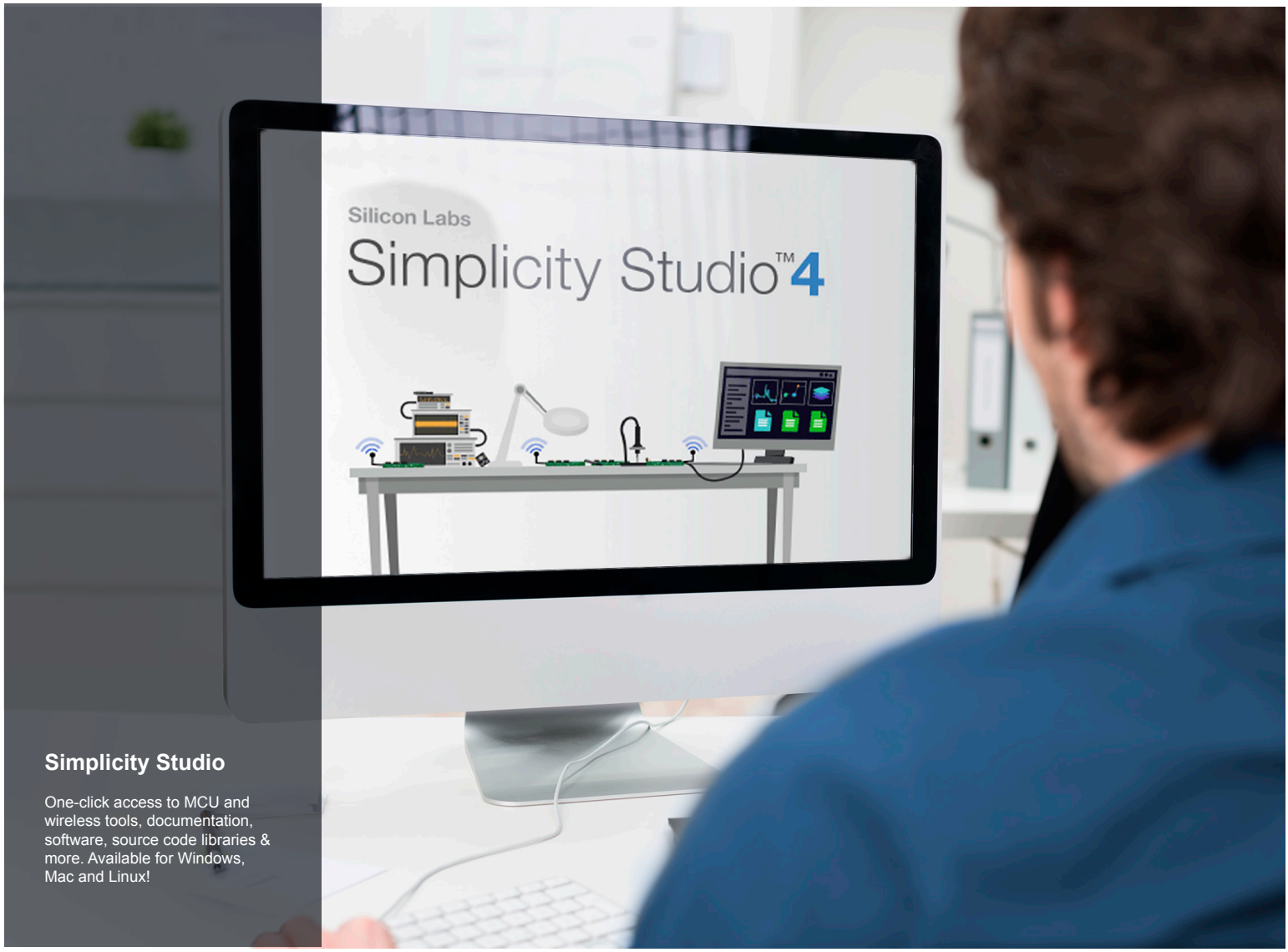
2010 年 11 月

・ lcdcontroller.c が廃止されるため、segmentlcd.c 関数を使用するようにソフトウェア/ドキュメントを変更
・ CMSIS Doxygen ドキュメントに関するセクションを追加
・ サンプル・フォルダの構造を変更し、ビルドおよび src フォルダを削除
・ chip init 関数を最新の efm32lib バージョンに更新

改訂 1.00

2010 年 9 月

・ 初期改定



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