Bluetooth® Low Energy プロトコルスタック
Embedded 構成サンプルプログラム

R01AN3319JJ0120
Rev.1.20
2022.01.31

要旨

RL78/G1D 単体で Bluetooth Low Energy 動作とアプリケーション動作を行う Embedded 構成サンプルプログラムの実装例を示します。ユーザは、Embedded 構成のプログラムを開発する際に、本サンプルプログラムをベースとして使用することができます。

開発者は、サンプルプログラムの理解を通じて、Embedded 構成のプログラムの実装方法を知ることができます。本書は、サンプルプログラムの使用方法、構成、機能詳細、実装詳細など、サンプルプログラムの理解を進める上で必要な知識を提供します。

サンプルプログラムは、Central role の動作を行うプログラムと Peripheral role の動作を行うプログラムの 2 種類があります。いずれも RL78/G1D 評価ボード上で動作し、BLE プロトコルスタックを利用するプログラムとして実装されています。

動作確認デバイス

RL78/G1D

関連資料

資料名	資料番号	
	和文	英文
Bluetooth Low Energy プロトコルスタック	-	-
ユーザーズマニュアル	R01UW0095J	R01UW0095E
API リファレンスマニュアル　基本編	R01UW0088J	R01UW0088E
クイックスタートガイド	R01AN2767J	R01AN2767E
Security Library	R01AN3777J	R01AN3777E
RL78/G1D	-	-
ユーザーズマニュアル 評価ボード	R30UZ0048J	R30UZ0048E

目次

1. 概要 4

2. サンプルプログラムのデモ 5
2.1 動作環境 5
2.2 環境構築 6
2.3 使用方法 8

3. 構成 11
3.1 サンプルプログラムの構成 11
3.2 ファイル構成 12

4. サンプルプログラムのビルド 13
4.1 共通手順 13
4.2 開発環境上でのビルド手順 14
4.2.1 e² studio 14
4.2.2 CS+ 14

5. サンプルプログラムの詳細 16
5.1 Sample Control Program 16
5.1.1 Connection 設定 16
5.1.2 同時接続台数設定 17
5.1.3 SW4 状態通知周期 17
5.1.4 セキュリティ設定 17
5.1.5 保存可能なボンディング情報の件数 17
5.1.6 セキュリティ機能の有効・無効設定 17
5.2 Console Driver 18
5.2.1 コンソール入力の処理 18
5.2.2 API 18
5.2.3 定義 19
5.3 Menu Driver 20
5.3.1 メニューの構成 20
5.3.2 メニューの操作 20
5.3.3 API 20
5.3.4 構造体 21
5.4 Sample Custom Service 23
5.5 Sample Custom Profile for Server role (SAMS) 24
5.5.1 API 24
5.5.1 Event 25
5.5.2 定義 26
5.6 Sample Custom Profile for Client Role (SAMC) 27
5.6.1 API 27
5.6.2 Event 29
5.6.3 定義 30
5.7 シーケンス図 31

6. Appendix 34
6.1 ROM サイズ・RAM サイズ 34

1. 概要

サンプルプログラムは、Sample Custom Profile を用いて、Embedded 構成のプログラムの実装例を示します。Sample Custom Profile は、サンプルプログラムのために独自に定義した簡易な Profile です。サンプルプログラムは、以下の実装を含みます。

- GAP（Generic Access Profile）を使用した基本的な Central role や Peripheral role の動作
- 暗号化によるセキュリティの確保
- RWKE（Renesas Wireless Kernel Extension）ユーザタスクやメッセージの使用
- Sample Custom Profile の追加
- 1 台の Central role のデバイスと複数の Peripheral role のデバイス間の同時接続
- RL78/G1D 評価ボード上の入出力デバイス（LED、プッシュボタンスイッチ）の使用

サンプルプログラムは、2 台以上の RL78/G1D 評価ボード：RTK0EN0001D01001BZ（以降、評価ボード）を使用します。1 台を Central role（以降、Central）、残りを Peripheral role（以降、Peripheral）として使用し、Central と Peripheral 間の通信を通じて、サンプルプログラムの動作を実現します。

本書は、以下の内容を含みます。

2 章では、サンプルプログラムのデモの環境構築と使用方法について説明します。

3 章では、サンプルプログラムのソフトウェア構成およびファイル・ディレクトリ構成を示します。

4 章では、サンプルプログラムのビルド方法ついて説明します。サンプルプログラムは、e² studio や CS+ を使用したビルドに対応しています。

5 章では、サンプルプログラムを構成するコンポーネントが提供する機能や API、設定項目などについて記載します。

2. サンプルプログラムのデモ

本章では、サンプルプログラムの動作環境、環境構築および使用方法について記載します。

2.1 動作環境

サンプルプログラムを使用する際の必要となる環境について記載します。

- ハードウェア環境
  - Personal Computer
    - PC/AT™ 互換機
      - プロセッサ:　1.6GHz 以上
      - メインメモリ:　1GByte 以上
      - インタフェース:　USB2.0（E1 エミュレータおよび RL78/G1D 評価ボード接続用）
  - デバイス
    - 2 台以上の RL78/G1D 評価ボード（RTK0EN0001D01001BZ）
    - USB ケーブル（A タイプ オス / mini-B タイプ オス）
- ツール
  - Renesas オンチップデバッギングエミュレータ E1（R0E000010KCE00）
- ソフトウェア環境
  - Windows®7 以降
  - e² studio V4.3.1.001 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
    または Renesas CS+ for CC V4.00.00 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
    または Renesas CS+ for CA, CX V3.02.00 / Renesas CA78K0R V1.72
  - Renesas Flash Programmer v3.02.00
  - Tera Term V4.89 （任意のターミナルソフトを使用可）
  - UART-USB 変換デバイスドライバ

    ※ 評価ボードと PC を接続する際に、UART-USB 変換 IC（FT232RL）のデバイスドライバを要求される場合があります。その際は、デバイスドライバを以下から入手してください。

    FTDI (Future Technology Devices International) – Drivers

    https://ftdichip.com/drivers/d2xx-drivers/

2.2 環境構築

図 2-1 にサンプルプログラムを使用する際の環境の構築手順を示します。

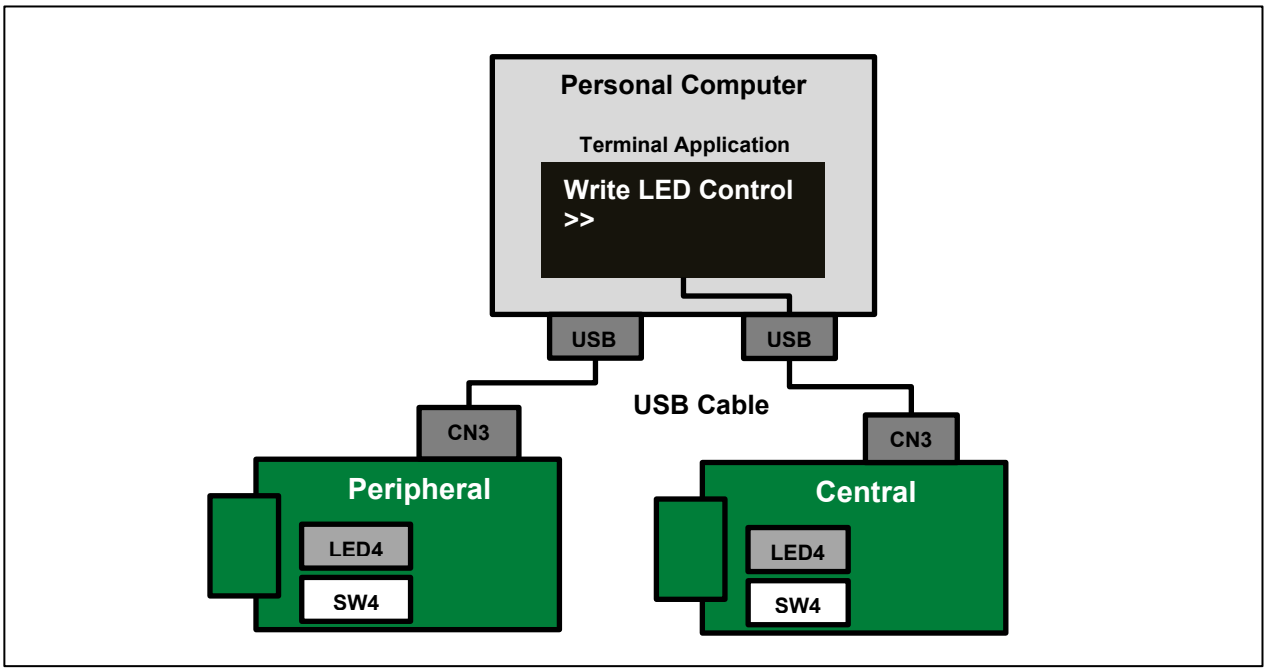


図 2-1 環境構築

サンプルプログラムの環境構築手順を以下に示します。

1. 2 台以上の評価ボードを用意します。1 台は Central として使用し、残りは Peripheral として使用します。Central は複数の Peripheral と同時に接続することが可能です（初期設定では 4 台まで可能、設定に関しては、5.1.2 章を参照）。Central と同時に接続したい Peripheral の台数に合わせて評価ボードを用意してください。

2. 表 2-1 にしたがって Central や Peripheral として使用する評価ボードのスイッチを設定してください。

表 2-1 評価ボードのスイッチ設定

スイッチ	設定	注意
SW7	OFF	ドットがある側が ON
SW8	OFF	
SW9	OFF	
SW10	ON	
SW11	OFF	
SW12	OFF	
SW13	ON	

3. パッケージに同梱の Hex ファイルを書き込みます。用意した評価ボードのうちの 1 台には Central 用の Hex ファイルを書き込みます。残りの評価ボードには、Peripheral 用の Hex ファイルを書き込みます。Hex ファイルの書き込み方法は、「クイックスタートガイド（R01AN2767）」の「5 章 プログラムを書き込む」を参照してください。

   - Central 用 Hex ファイル:
     ROM_File¥cc_rl¥Embedded¥RL78_G1D_CCE(EMBSMP_Central).hex

   - Peripheral 用 Hex ファイル:
     ROM_File¥cc_rl¥ Embedded¥RL78_G1D_CCE(EMBSMP_Peripheral).hex

4. PC にターミナルソフトをインストールします。ターミナルソフトは、任意のソフトウェアを使用できますが、本書では Tera Term V4.89 を使用します。

5. 表 2-2 にしたがってターミナルソフトの設定をします。

表 2-2 ターミナルソフト設定

設定項目	設定値
改行コード（受信）	LF
改行コード（送信）	CR
ボーレート	4,800 [bps]
データ長	8 [bit]
パリティ	none
ストップビット	1 [bit]
フロー制御	none

6. 図 2-1 に示すように、USB ケーブルを介して、Central を PC に接続します。次に PC 上でターミナルソフトを起動します。Central 上の SW5 を押下して、Central をリセットすると、ターミナルソフト上に操作メニューが表示されます。Peripheral は、USB ケーブルを介して PC に接続します。
Central は電源供給に加えて、ターミナルソフトから操作を行うために PC に接続します。Peripheral は、電源供給およびターミナルソフトにパスキーの表示を行うために PC に接続します。

7. Central、Peripheral ともに、起動すると LED1 および LED2 が点滅します。省電力機能が動作するため、点滅が停止したり、点滅周期が長くなったりしますが、これは通常の動作です。

2.3 使用方法

以下に、サンプルプログラムの使用方法を示します。

1. Peripheral は、電源を供給すると自動的に Advertise を開始します。

2. Central は、電源を供給するとターミナルソフト上に操作メニューを表示し、ユーザの指示を待ちます。操作メニューの遷移図を図 2-2 に示します。ユーザは、操作メニューの項目の先頭に付与された番号を、数値キー（0 から 9）を使用して入力後 Enter キー押下することで操作メニューを選択します。上位層のメニューに移動する場合は、Esc キーを押下します。

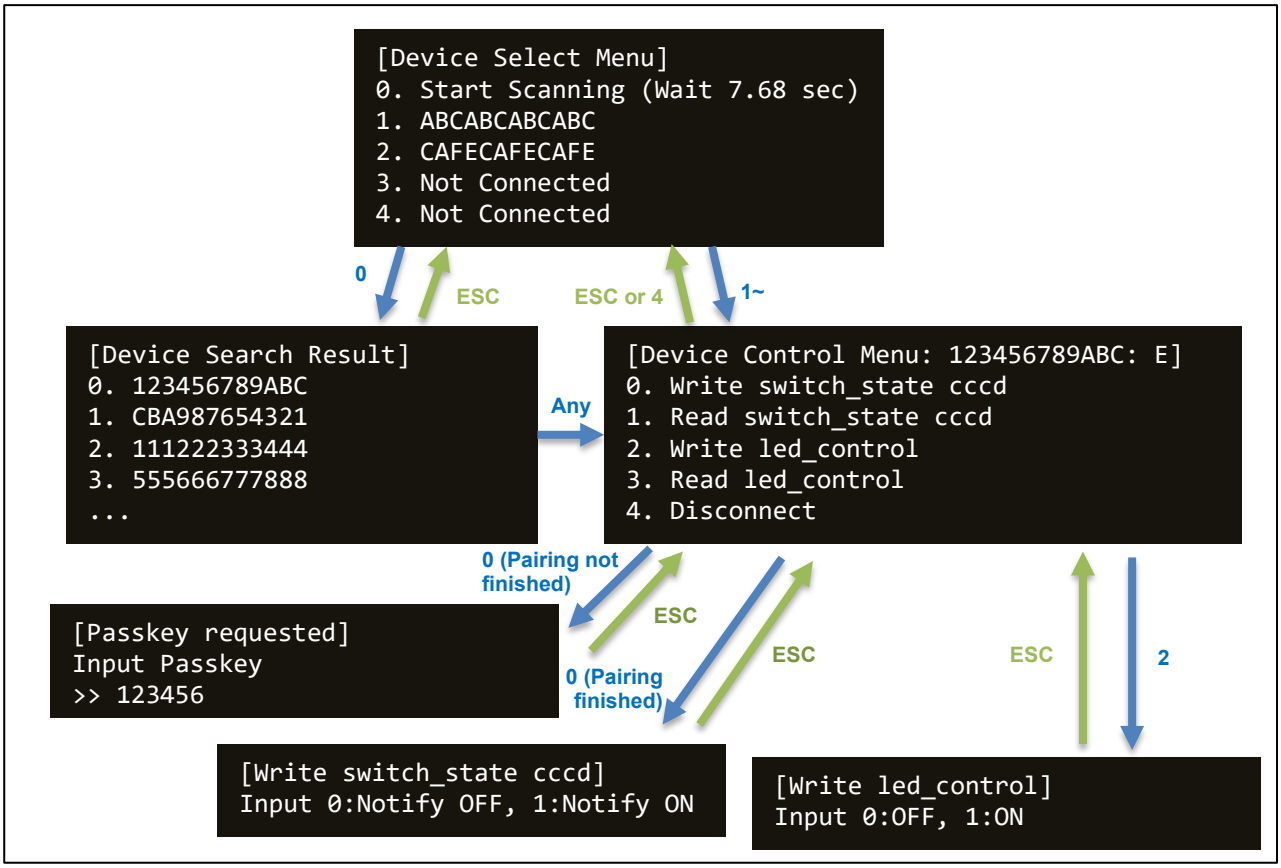


図 2-2 メニュー遷移

3. 「Device Select Menu」の「Start Scanning (Wait 7.68 sec)」を選択した場合、周辺の Peripheral を 7.68 秒間（RBLE_Device_Search API 仕様に準ずる）検索し、「Device Search Result」に発見した Peripheral のアドレスのリストを表示します。リストから接続したい Peripheral を番号で選択すると、その Peripheral との接続を開始します。

4. 接続が確立すると、その Peripheral と通信するための「Device Control Menu」が表示されます。「Device Control Menu」には、接続している Peripheral のアドレスに加えて、Peripheral との暗号化の実施状況が表示されます。暗号化が実施済みの場合は、「: E」と表示されます。ペアリングが未実施の場合は、「: No Sec」と表示されます。このメニューでは、以下の 5 つの操作を行えます。

   1. **Write switch_state cccd:** Peripheral からの通知の有効・無効を制御します。有効にする場合 1 を Write し、無効にする場合 0 を Write します。Peripheral からの通知が有効化されている場合、図 2-3 左図に示すように、Peripheral は 500ms 間隔で SW4 の押下・開放状態を Central に通知します。Central は、受信した SW4 の状態に応じて、Central 上の LED4 の点灯・消灯を制御します。cccd は、Client Characteristic Configuration Descriptor の略です。
      本 Characteristic の Permission には、Unauthenticated pairing が設定されているため、ペアリング完了

前にアクセスした場合は、Error が発生します。Central は、エラーを受け取ると、ペアリングを開始します。Peripheral のターミナルソフトに表示されたパスキーを Central のターミナルソフトに正しく入力すると、ペアリングが完了し、Peripheral との通信が暗号化されます。ペアリング時に Central / Peripheral 間で交換されたセキュリティ情報は、ペアリング完了時にボンディングにより RL78/G1D 内蔵データフラッシュに保存されるため、以降の再接続時にも使用できます。

2. **Read switch_state cccd:** Peripheral の通知の有効・無効の設定を取得します。有効の場合は 1 が Read され、無効の場合は 0 が Read されます。
3. **Write led_control:** Peripheral 上の LED4 の点灯・消灯を制御します。LED4 を点灯する場合は 1 を Write し、LED を消灯する場合は 0 を Write します。デフォルトの値は 0 です。
4. **Read led_control:** Peripheral 上の LED4 の点灯・消灯状態を取得します。LED4 が点灯している場合は 1 が Read され、消灯している場合は 0 が Read されます。
5. **Disconnect:** Peripheral との接続を切断します。切断を行うと、「Device Select Menu」に戻ります。

5. 複数台の Peripheral との接続する場合、次のような手順で操作します。
   1. 「Device Select Menu: Start Scanning (Wait 7.68)」を実行し、Peripheral の発見および接続を行います。
   2. 次に「Device Control Menu」上で Esc キーを押下し、「Device Select Menu」に戻ります。
   3. 再度「Device Select Menu: Start Scan (wait 7.68)」を実行することで、別の Peripheral を発見し、接続の確立を行うと、複数台の Peripheral との同時接続が行えます。
   4. 接続済みの Peripheral は、「Device Select Menu」から確認・選択できます。

   注) 「Device Select Menu」の「Start Scanning (Wait 7.68 sec)」を除いたメニュー数は、Central が同時に接続可能な Peripheral の台数を示します。図 2-2 の場合、4 台まで接続可能で、現状は 2 台接続されていることを示しています。

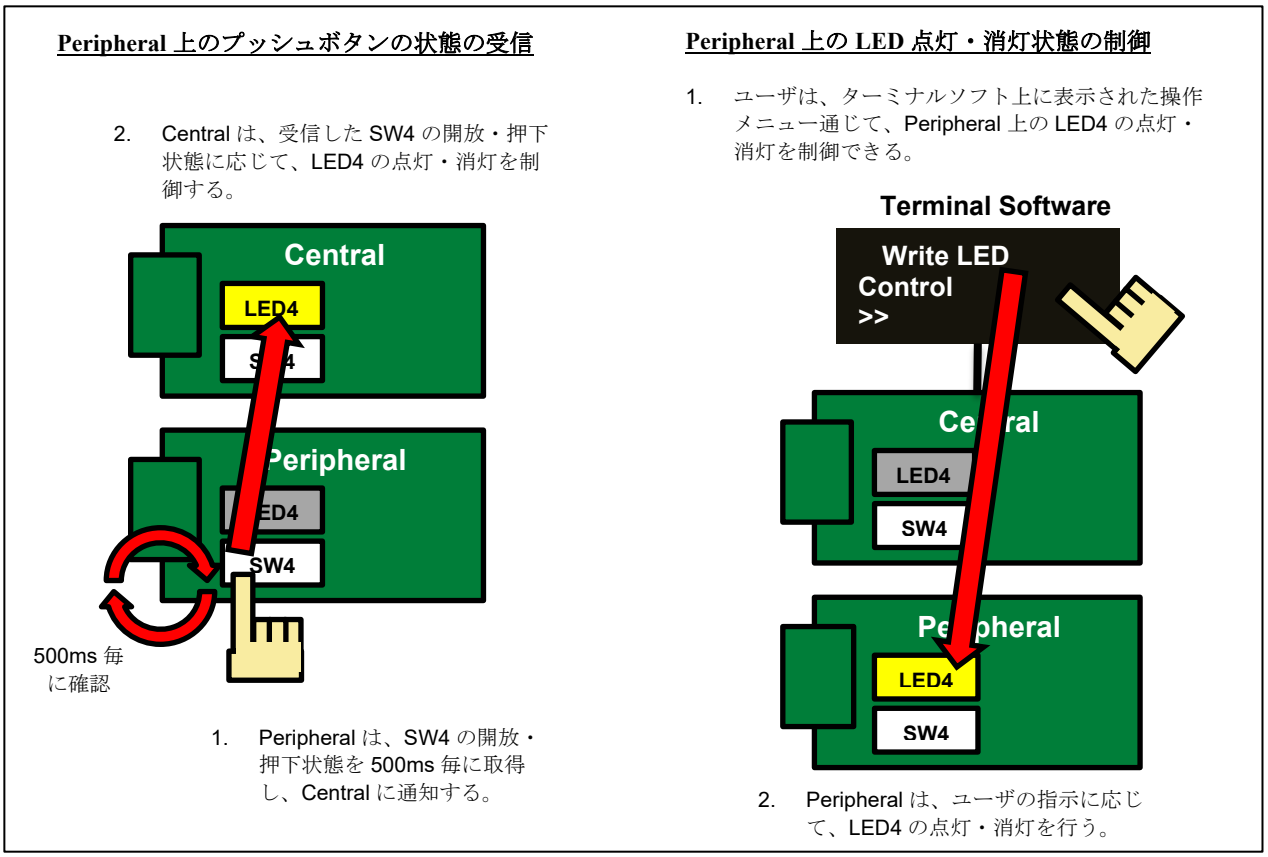


図 2-3 Central と Peripheal 間の操作

6. ペアリング完了時に保存されたセキュリティ情報は、SW2 を押下することにより削除できます。削除の対象は、非接続中のデバイスのセキュリティ情報です。接続中のデバイスに関連するセキュリティ情報は削除されません。すべてのセキュリティ情報を削除するには、すべての接続を切断したあとに、SW2 を押下してください。削除が完了すると、ターミナルソフトに「Bonding information is deleted.」と表示されます。

3. 構成

本章では、サンプルプログラムの構成と本パッケージのファイル・ディレクトリ構成について説明します。

3.1 サンプルプログラムの構成

図 3-1 にサンプルプログラムの構成を示します。サンプルプログラムを使用する際のソフトウェア・ハードウェアは、3 つの部分：サンプルプログラム（赤色）、BLE プロトコルスタック（青色）、デバイス（黒色）から構成されます。Central と Peripheral はそれぞれ異なる動作をするため、サンプルプログラムには、Central 向けと Peripheral 向けの 2 種類があります。セキュリティ機能を実現するため、Central / Peripheral の両方に Security Library（R01AN3777）を組み込んでいます。

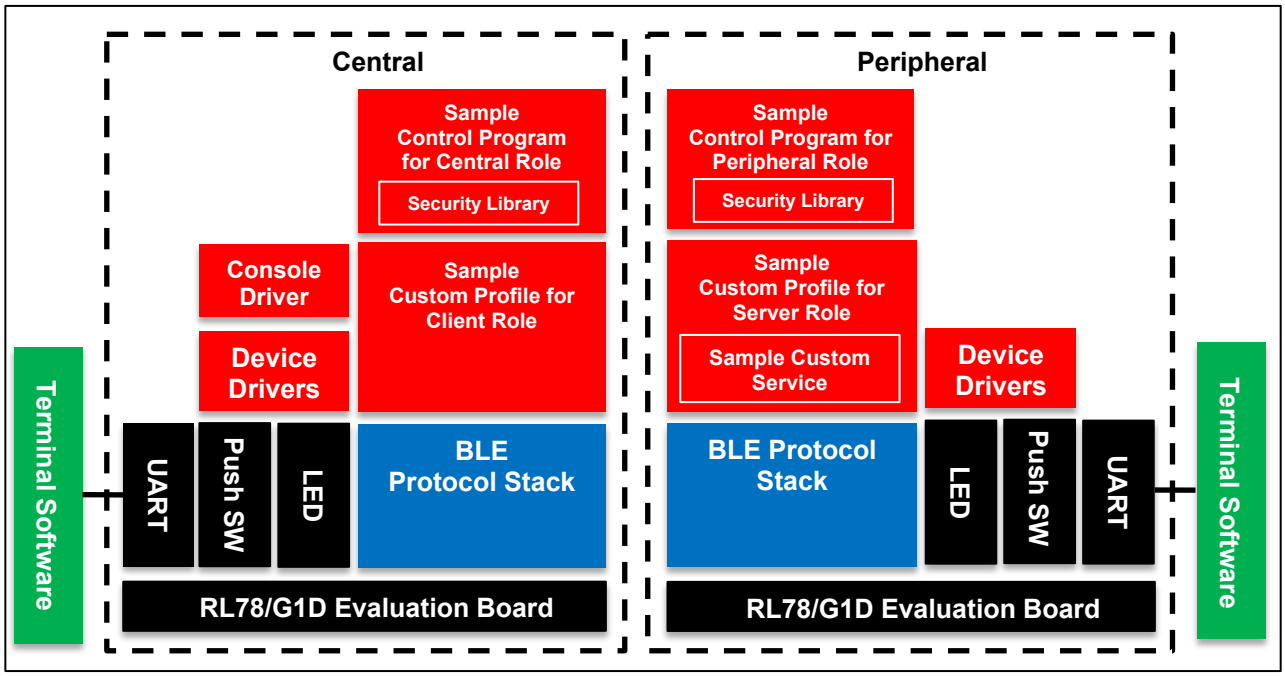


図 3-1 サンプルプログラムの構成

Central 向けサンプルプログラム

- Sample Control Program for Central role は、周辺の Peripheral の検索や接続の開始を行います。接続を確立後は、GATT Client role として動作します。
- Sample Custom Profile for Client role は、BLE プロトコルスタックが提供する GATT API をラップし、Sample Custom Service の使用を容易にするための API を提供します。
- Central は、LED の制御やコンソールによる文字列の入出力を行うため、LED および Console の Device Driver を含みます。

Peripheral 向けサンプルプログラム

- Sample Control Program for Peripheral role は、Advertise の実行や Central との接続を行います。Central と接続確立後、GATT Server role として動作し、Sample Custom Service を提供します。
- Sample Custom Profile for Server role は、BLE プロトコルスタックが提供する GATT API をラップし、Sample Custom Service の使用を容易にするための API を提供します。
- Peripheral は、LED の制御やプッシュボタンスイッチ状態の取得を行うため、LED およびプッシュボタンスイッチの Device Driver を含みます。またログやパスキーを表示するため、Console の Device Driver を含みます。

3.2　ファイル構成

パッケージのファイル構成を以下に示します。パッケージには、サンプルプログラムで使用するために BLE プロトコルスタック V1.20 に対して変更・追加したファイルのみを含みます。

```
Project_Source
    +---rBLE
    |   ¥---src
    |       +---sample_app                          Sample Control Program
    |       |   |   console.c                       Console Driver
    |       |   |   console.h
    |       |   |   menu.c                          Menu Driver
    |       |   |   menu.h
    |       |   |   rble_sample_app_central.c       Sample Control Program for Central Role
    |       |   |   rble_sample_app_central.h
    |       |   |   rble_sample_app_peripheral.c    Sample Console Program for Peripheral Role
    |       |   |   rble_sample_app_peripheral.h
    |       |   ¥---seclib
    |       |           seclib.c
    |       |           seclib.h                    Security Library
    |       |           secdb.c
    |       |           secdb.h
    |       ¥---sample_profile                      Sample Custom Profile
    |           ¥---sam
    |                   sam.h                       Sample Custom Profile Client/Server 共通定義
    |                   samc.c                      Sample Custom Profile for Client Role
    |                   samc.h
    |                   sams.c                      Sample Custom Profile for Server Role
    |                   sams.h
    ¥---renesas
        +---src
        |   +---arch
        |   |   ¥---rl78
        |   |           arch_main.c                 BLE ソフトウェアメインループ
        |   |           db_handle.h                 Sample Custom Service Handle 定義
        |   |           ke_conf.c                   RWKE タスク定義
        |   |           prf_config.c                Sample Custom Service 定義
        |   |           prf_config.h
        |   |           prf_sel.h                   使用する Profile の選択
        |   ¥---driver
        |       +---uart                            UART 4800bps 設定
        |       |       uart.c
        |       ¥---dataflash
        |               eel_descriptor_t02.c        Data Flash Library
        |               eel_descriptor_t02.h
        ¥---tools
            ¥---project
                +---CS_CCRL
                |   +---BLE_Embedded_for_Peripheral CS+ for CC プロジェクトファイル
                |   ¥---BLE_Embedded_for_Central
                +---CubeSuite
                |   +---BLE_Embedded_for_Peripheral CS+ for CA,CX プロジェクトファイル
                |   ¥---BLE_Embedded_for_Central
                +---e2studio
                    +---BLE_Embedded_for_Peripheral e2 studio プロジェクトファイル
                    ¥---BLE_Embedded_for_Central
```

4. サンプルプログラムのビルド

サンプルプログラムは、以下の開発環境を用いてビルドできます。

- e² studio V4.3.1.001 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
- Renesas CS+ for CC V4.00.00 / RL78 Family C Compiler Package V1 (without IDE) V1.03.00
- Renesas CS+ for CA, CX V3.02.00 / Renesas CA78K0R V1.72

4.1 共通手順

以下にサンプルプログラムのビルド手順を示します。

1. サンプルプログラムをビルドするためには、BLE プロトコルスタックおよび EEPROM エミュレーションライブラリを入手する必要があります。それぞれルネサスの Web ページからダウンロードしてください。
   - BLE プロトコルスタック
     - https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rl78-family
   - EEPROM エミュレーションライブラリ
     - RL78 ファミリ EEPROM エミュレーションライブラリ Pack02 パッケージ Ver.2.00(CA78K0R/CC-RL コンパイラ用)
       - https://www.renesas.com/software-tool/data-flash-libraries

       ※ 上記に示すダウンロードリンクは、Web サイトのリニューアルなどにより、予告なく変更になる場合があります。
2. BLE プロトコルスタックを解凍し、サンプルプログラムのソースコードディレクトリ（Project_Source）を BLE_Software_Ver_X_XX¥RL78_G1D¥Project_Source に上書きしてください。
3. EEPROM エミュレーションライブラリをインストールします。インストール手順は、「クイックスタートガイド（R01AN2767）」の「4.2 章 EEPROM エミュレーションライブラリのインストール」を参照してください。

4.2 開発環境上でのビルド手順

本章では、各開発環境を使用してサンプルプログラムをビルドする方法を示します。

4.2.1 e² studio

1. e² studio を起動します。
2. 「プロジェクトエクスプローラー」上で右クリックし、表示されたメニューから「インポート」を選択します。
3. 「インポート」ウィンドが表示されるので、「既存プロジェクトをワークスペースへ」を選択し、「次」をクリックします。
4. 「ルートディレクトリの選択」フォームに、に示すプロジェクトディレクトリを選択します。
5. 選択後、「プロジェクト」内に指定したプロジェクトが表示されていることを確認し、「終了」をクリックします。すると、「インポート」ウィンドが閉じられます。
6. 「プロジェクトエクスプローラー」上に表示されたプロジェクト上で右クリックし、「プロジェクトのビルド」を選択し、ビルドを開始します。
7. 表 4-1 に示すパスに Hex ファイルが生成されます。

4.2.2 CS+

1. 表 4-1 に示すプロジェクトファイルをダブルクリックします。これにより CS+が起動します。
2. 「プロジェクトツリー」内の「BLE_Emb（サブプロジェクト）」を右クリックし、ドロップダウンメニューから「BLE_Emb をビルド」を選択して、ビルドを開始します。
3. 表 4-1 に示すパスに Hex ファイルが生成されます。

表 4-1 プロジェクトファイルと Hex ファイルのパス

e² studio with CC-RL	
Project Directory	**Peripheral:** e2studio¥BLE_Embedded_for_Peripheral¥rBLE_Emb **Central:** e2studio¥BLE_Embedded_for_Central¥rBLE_Emb
Hex File	**Peripheral:** e2studio¥BLE_Embedded_for_Peripheral¥rBLE_Emb¥DefaultBuild¥rBLE_Emb_CCRL.hex **Central:** e2studio¥BLE_Embedded_for_Central¥rBLE_Emb¥DefaultBuild¥rBLE_Emb_CCRL.hex
CS+ with CC-RL	
Project File	**Peripheral:** CS_CCRL¥BLE_Embedded_for_Peripheral¥BLE_Embedded_for_Peripheral.mtpj **Central:** CS_CCRL¥BLE_Embedded_for_Central¥BLE_Embedded_for_Central.mtpj
Hex File	**Peripheral:** CS_CCRL¥BLE_Embedded_for_Peripheral¥rBLE_Emb¥DefaultBuild¥rBLE_Emb_CCRL.hex **Central:** CS_CCRL¥BLE_Embedded_for_Central¥rBLE_Emb¥DefaultBuild¥rBLE_Emb_CCRL.hex
CS+ with CA78K0R	
Project File	**Peripheral:** CubeSuite¥BLE_Embedded_for_Peripheral¥BLE_Embedded_for_Peripheral.mtpj **Central:** CubeSuite¥BLE_Embedded_for_Central¥BLE_Embedded_for_Central.mtpj
Hex	**Peripheral:** CubeSuite¥BLE_Embedded_for_Peripheral¥BLE_Emb¥DefaultBuild¥BLE_Emb.hex **Central:** CubeSuite¥BLE_Embedded_for_Central¥BLE_Emb¥DefaultBuild¥BLE_Emb.hex

(基準パス: ¥BLE_Software_Ver_X_XX¥RL78_G1D ¥Project_Source¥renesas¥tools¥project¥)

5. サンプルプログラムの詳細

本章は、サンプルプログラムの各コンポーネントの動作や設定内容、API について記載します。

5.1 Sample Control Program

Sample Control Program は、Advertising（Peripheral）や Scanning および接続の確立（Central）を行います。これらは、BLE プロトコルスタックが提供する GAP API を用いて実現しています。API の詳細については、「API リファレンスマニュアル：基本編（R01UW0088J）」を参照してください。サンプルプログラムが行う API 呼び出しに関しては、本書の 5.7 章を参照してください。

以下は、Sample Control Program の設定項目について記載します。

5.1.1 Connection 設定

Peripheral は起動後、または Central との接続切断後、自動的に Advertising を開始します。表 5-1 に Advertising 設定を示します。Advertising 設定を変更する場合は、「rBLE¥src¥sample_app¥rble_sample_peripheral.c」内の「app_advertise_param 変数」を参照してください。

表 5-1 Advertising 設定

Advertising Type	Connectable Undirected Advertising (ADV_IND)
Advertising Interval Min	20 [ms]
Advertising Interval Max	30 [ms]
Advertising Channel Map	All Channels (37, 38, 39 [ch])
Advertising Data	-
<<Flags>> (0x01)	LE General Discoverable Mode BR/EDR Not Supported
<<Complete Local Name>> (0x09)	“REL-BLE”
<<Complete List of 128-bit Service Class UUIDs>> (0x07)	Sample Custom Service UUID 5BC1B9F7-A1F1-40AF-9043-C43692C18D7A
Scan Response Data	none

Central は、周辺の Peripheral を見つけるために、Scanning を実施し、発見した Peripheral のリストを表示します。この際、Advertising Data に Sample Custom Service の UUID を含まない Peripheral は、表示しません。

Central は、ユーザが操作メニューから選択した Peripheral との接続を確立します。表 5-2 は、接続時の Connection 設定を示します。接続設定を変更する場合は、「rBLE¥src¥sample_app¥rble_sample_app_central.c」内の「app_connection_param」を参照してください。

表 5-2 Connection 設定

Scan Interval	60 [ms]
Scan Window Size	30 [ms]
Initiator Filter Policy	none
Minimum Connection Interval	30 [ms]
Maximum Connection Interval	50 [ms]
Connection Latency	0
Link Supervision Timeout	3 [s]
Minimum Connection Event Length	0 [ms]
Maximum Connection Event Length	50 [ms]

5.1.2 同時接続台数設定

Central が同時に接続可能な Peripheral の台数は、表 5-3 に示すマクロ定義により決定されます。マクロ定義は、統合開発環境のプロジェクト設定から変更できます。

表 5-3 同時接続台数のマクロ定義

マクロ定義	初期設定値	設定可能値
CFG_CON	4	1~7 (※)

※ BLE プロトコルスタックの仕様では、1~8 が設定可能ですが、本サンプルプログラムを使用する場合は、メモリ使用量の制限により、1~7 になっています。

5.1.3 SW4 状態通知周期

Peripheral が SW4 状態を Central に通知する周期は、RWKE が提供するタイマ管理機能を使用して生成します。通知周期は、表 5-1 に示すマクロ定義により決定されます。通知周期を変更したい場合は、この値を変更してください。設定可能な範囲は、タイマ管理機能が提供する「ke_timer_set」の「delay 引数」の仕様にしたがいます。

表 5-4 SW4 状態通知周期のマクロ定義

マクロ定義	初期設定値
APP_SWITCH_STATE_CHECK_INTERVAL	50 (500ms 周期)

5.1.4 セキュリティ設定

Central および Peripheral のセキュリティ設定を表 5-5 に示します。各パラメータの詳細は、「Security Library（R01AN3777）」を参照してください。

表 5-5 セキュリティ設定

Security Parameters	Central 設定	Peripheral 設定
role	RBLE_MASTER	RBLE_SLAVE
auth_req	RBLE_AUTH_REQ_MITM_BOND	RBLE_AUTH_REQ_MITM_BOND
iocap	RBLE_IO_CAP_KB_ONLY	RBLE_IO_CAP_DISPLAY_ONLY
rpa_generate	TRUE	FALSE

5.1.5 保存可能なボンディング情報の件数

ペアリング完了時に保存するボンディング情報の件数は、表 5-6 に示すマクロ定義により決定されます。保存可能なボンディング情報の件数を変更したい場合は、この値を変更してください。件数を変更した場合、Data Flash Library の修正が必要な場合があります。変更方法の詳細は、「Security Library（R01AN3777）」を参照してください。

表 5-6 ボンディング情報保存可能件数のマクロ定義

マクロ定義	Central	Peripheral
CFG_SECLIB_BOND_NUM	4	4

5.1.6 セキュリティ機能の有効・無効設定

暗号化やプライバシーのセキュリティ機能の有効・無効設定は、以下のマクロ定義により決定されます。デフォルトは、有効になっています。無効化するためには、本マクロ定義の削除および Security Library（seclib.c, secdb.c）をビルド対象外にしてください。

表 5-7 セキュリティ機能のマクロ定義

マクロ定義
USE_SECLIB

5.2 Console Driver

Console Driver は、以下の 2 つの機能を提供します。Console Driver は、内部で UART Driver を使用しています。

- コンソール入力: ユーザがターミナルソフトに入力した文字を UART 経由で受け取り、行単位にまとめた上で指定したタスクに送信する機能を提供します。
- コンソール出力: サンプルプログラムのログ出力や、操作メニューを表示するために、ターミナルソフトに文字を出力する機能を提供します。

5.2.1 コンソール入力の処理

Console Driver は、ターミナルソフトから受け取った文字を行単位にまとめて指定したタスクに送信します。行単位で送信するために、ターミナルから受け取った文字は一度内部バッファに格納し、改行もしくは Esc 文字を受け取った際に内部バッファの文字をまとめて送信します。

タスクへの行単位の文字の送信は、RWKE が提供するメッセージ機能を使用します。送信先のタスクには、CONSOLE_MSG_LINE_IN（5.2.3(a)）をメッセージ ID とする RWKE メッセージが通知されます。メッセージの内容として、CONSOLE_MSG 構造体（5.2.3(b)）を使用します。

5.2.2 API

(a) console_init

void console_init(bool enable_in, ke_task_id_t task_id)			
- コンソールの初期化を行います。			
Parameters:			
bool	enable_in	true	コンソール入力を使用
		false	コンソール入力を不使用
ke_task_id	task_id	コンソール入力を通知するタスク	

(b) console_enable_in

void console_enable_in(void)
- console_disable_in によって無効化したコンソール入力を有効化します。 - console_init にて、enable_in を false に設定した場合は、本 API を呼び出さないでください。

(c) console_disable_in

void console_disable_in(void)
- 本 API は、コンソール入力を一時的に無効化し、不要なコンソール入力の発生を避けます。 - Esc キーを押下すると、強制的にコンソール入力が有効化されます。

(d) Printf

void Printf(const char_t *fmt, ...)
- 標準ライブラリの printf と同等のコンソール出力機能を提供します。

5.2.3 定義

(a) CONSOLE_MSG_LINE_IN

コンソール入力を通知する際に使用する Message の ID です。

```
#define CONSOLE_MSG_LINE_IN (KE_FIRST_MSG(TASK_CON_APPL) + 2)
```

(b) CONSOLE_MSG

コンソール入力を通知する際に使用する Message のデータ構造です。len は、buf のバイト数を示します。CONSOLE_MSG を配置するメモリ領域は、len のサイズに合わせて確保する必要があります。サンプルプログラムでは、「sizeof(CONSOLE_MSG) + len」分のメモリ領域を ke_msg_alloc API により動的に割当てています。

```
typedef struct {
  uint8_t len;     // Message Data 長
  char_t  buf[1];  // Message Data
} CONSOLE_MSG;
```

5.3 Menu Driver

Menu Driver は、ターミナルソフト上に表示するメニューの制御を行います。

5.3.1 メニューの構成

図 5-1 に示すようにメニューは複数の要素の階層構造として表されます。

- メニューリスト（LIST）は、（ディレクトリのように）複数の子要素を持ちます。子要素は、他の LIST やメニュー項目（ITEM）です。LIST が選択されると、その子要素を表示します。
- ITEM は、（ファイルのように）特定の動作を持つ要素です。ITEM が選択されると、指定したハンドラが呼び出されます。
- SINGLE は、子要素に 1 つの ITEM のみを持つメニューリストです。SINGLE が選択されると、唯一の子要素の ITEM の動作を実行します。
- TERMINATOR は、実装上の都合により、LIST の子要素の終端に付加する必要があります。TERMINATOR はメニューには表示されません。

```
LIST
  |
  +---- LIST
  |     +---- ITEM
  |     +---- ITEM
  |     +---- TERMINATOR
  |
  +---- ITEM
  |
  +---- SINGLE
  |     +---- ITEM
  |     +---- TERMINATOR
  |
  +---- TERMINATOR
```

図 5-1 メニューの構成例

5.3.2 メニューの操作

各要素には、通し番号が付加されるため、ユーザはその番号入力し、Enter キーを押下することで要素の選択をします。メニューの上層への移動は、Esc キーを使用します。要素を選択後、次にメニューを表示するまでの間は、コンソール入力を無効します。これは、処理の実行中に不要なコンソール入力が発生するのを避けるためです。

5.3.3 API

(a) menu_show

void menu_show(MENU *menulist)		
- ターミナルソフトに表示する LIST を指定します。		
Parameters:		
MENU	menulist	表示する LIST

(b) menu_user_in

```
int_t menu_user_in(ke_msg_id_t const msgid, void const *param,
                   ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

- コンソール入力の通知を受け付けるための Kernel Message のハンドラです。
- 本 API を CONSOLE_MSG_LINE_IN に対するハンドラとして登録してください。

5.3.4 構造体

(a) MENU_TYPE

メニューの要素の種別を示します。

```
typedef enum {
  MENU_TYPE_TERMINATOR,
  MENU_TYPE_LIST,
  MENU_TYPE_SINGLE,
  MENU_TYPE_ITEM,
} MENU_TYPE;
```

(b) MENU

メニューの 1 つの要素を表す構造体です。MENU_TYPE によって設定すべき項目が異なります。

```
typedef struct MENU_t {
// for LIST, SINGLE, ITEM
MENU_TYPE        type;                    // メニュー種別
char_t        title[MENU_TITLE_SIZE];  // 要素選択時に表示される文字

// for LIST, SINGLE
struct MENU_t         *parent;      // 親要素、Esc キー押下時に参照
struct MENU_t         *children;   // 子要素
MENU_UPDATE_HANDLER   update;       // LIST の子要素を動的に変更するハンドラ
MENU_CANCEL_HANDLER   cancel        // Esc キー押下時に処理を実行するハンドラ

// for ITEM
MENU_HANDLER     handler;              // ITEM 選択時に呼び出されるハンドラ
```

(c) MENU_HANDLER

ITEM を選択した際に呼び出されるハンドラです。

```
typedef void (*MENU_HANDLER)(void *arg)
```

(d) MENU_UPDATE_HANDLER

LIST の子要素を動的に変更する際に呼び出されるハンドラです。LIST の表示直前に呼び出されます。サンプルプログラムでは、Peripheral の検索結果の表示時などに使用しています。

```
typedef void (*MENU_UPDATE_HANDLER)(void)
```

(e) MENU_CANCEL_HANDLER

Esc キーを押下した際に呼び出されるハンドラです。サンプルプログラムでは、パスキー入力時にパスキーの入力をキャンセルする際に使用しています。

```
typedef void (*MENU_CANCEL_HANDLER)(void)
```

5.4 Sample Custom Service

Peripheral は、Sample Custom Service を提供します。表 5-8 に Sample Custom Service 定義を示します。

表 5-8 Sample Custom Service 定義

Type	Value	Permission
Sample Custom Service		
Primary Service Declaration (0x2800)	5BC1B9F7-A1F1-40AF-9043-C43692C18D7A	Read
Switch State Characteristic		
Characteristic Declaration (0x2803)	Prop: Notification UUID: 5BC18D80-A1F1-40AF-9043-C43692C18D7A	Read
Characteristic Value	1 [octet]	Notification
Client Characteristic Configuration Descriptor (0x2902)	2 [octet]	Read, Write
LED Control Characteristic		
Characteristic Declaration (0x2803)	Prop: Read, Write UUID: 5BC143EE-A1F1-40AF-9043-C43692C18D7A	Read
Characteristic Value	1 [octet]	Read, Write

Switch State Characteristic

Switch State Characteristic Value に格納されたスイッチの押下・開放状態を、Notification により通知します。表 5-9 に Switch State Characteristic Value のビット定義を示します。Switch State Characteristic は、Notification の有効・無効を制御するため、Client Characteristic Configuration Descriptor を含みます。

表 5-9 Switch State Characteristic のビット定義

Bit	Definition	Key	Value
0	Switch State	0	RELEASE
		1	PUSH
1 to 7	Reserved For Future Use	-	-

LED Control Characteristic

LED Control Characteristic Value に Write された値に応じて LED の状態を制御します。表 5-10 に LED Control Characteristic Value のビット定義を示します。

表 5-10 LED Control Characteristic のビット定義

Bit	Definition	Key	Value
0	LED State	0	OFF
		1	ON
1 to 7	Reserved For Future Use	-	-

5.5 Sample Custom Profile for Server role (SAMS)

SAMS は、BLE プロトコルスタックが提供する GATT API（RBLE_GATT_xxx）をラップし、Sample Custom Service へのアクセスを容易にするための API・Event・構造体などを提供します。SAMS が使用する GATT API は、5.7 章に示すシーケンス図を参照してください。

API 名の先頭に付加されている「SAMPLE_」は、サンプルプログラム内で定義されていることを示します。サンプルプログラム以外では使用できません。

5.5.1 API

(a) SAMPLE_Server_Enable

```
RBLE_STATUS SAMPLE_Server_Enable (uint16_t conhdl, uint8_t con_type,
  SAMPLE_SERVER_PARAM *param, SAMPLE_SERVER_EVENT_HANDLER callback)
```

- 本 API は、SAMS を有効化します。API の実行完了は、SAMPLE_SERVER_EVENT_ENABLE_COMP により通知されます。
- con_type は、SAMS の設定値の初期化の実施可否を指定します。con_type に RBLE_PRF_CON_NORMAL を指定した場合、param で指定した値を用いて SAMS の初期化を実施します。
- param は、SAMS の設定値です。
- callback は、SAMS に関するイベントが発生した際に呼び出されます。

Parameters:

uint16_t	conhdl	Connection Handle	
uint8_t	con_type	RBLE_PRF_NORMAL	初期化を実施
		RBLE_PRF_DISCOVERY	初期化を非実施
SAMPLE_SERVER_PARAM *	param	SAMS 設定値	
SAMPLE_SERVER_EVENT_HANDLE	callback	SAMS Event Handler	

Return:

RBLE_OK	Success
RBLE_PARAM_ERR	Parameter Error
RBLE_STATUS_ERROR	Status Error

(b) SAMPLE_Server_Disable

```
RBLE_STATUS SAMPLE_Server_Disable (uint16_t conhdl)
```

- 本 API は、SAMS を無効化します。API の実行完了は、SAMPLE_SERVER_EVENT_DISABLE_COMP により通知されます。

Parameters:

uint16_t	conhdl	Connection Handle

Return:

RBLE_OK	Success
RBLE_PARAM_ERR	Parameter Error
RBLE_STATUS_ERROR	Status Error

(c) SAMPLE_Server_Send_Switch_State

RBLE_STATUS SAMPLE_Server_Send_Switch_State (uint16_t conhdl, uint8_t value)			
- 本 API は、SAMC に switch state Characteristic Value を Notify します。 - value は、Notify する switch state 値です。			
Parameters:			
uint16_t	conhdl	Connection Handle	
uint8_t	value	SAMPLE_SWITCH_STATE_ON	Switch は押下状態
		SAMPLE_SWITCH_STATE_OFF	Switch は開放状態
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

5.5.1 Event

(a) SAMPLE_SERVER_EVENT_ENABLE_COMP

SAMPLE_SERVER_ENABLE_COMP		
- SAMS の有効化の完了を通知するイベントです。		
Parameters:		
RBLE_STATUS	status	SAMS 有効化結果
uint16_t	conhdl	Connection Handle

(b) SAMPLE_SERVER_EVENT_DISABLE_COMP

SAMPLE_SERER_DISABLE_COMP		
- SAMS の無効化の完了を通知するイベントです。		
Parameters:		
SAMPLE_SERVER_PARAM	param	SAMS 設定値、無効化完了時に設定されていた値を返却
uint16_t	conhdl	Connection Handle

(c) SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND

SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND			
- LED Control Characteristic Value に対して Write が行われたことを通知するイベントです。			
Parameters:			
uint8_t	value	SAMPLE_PRF_LED_CONTROL_ON	LED ON
		SAMPLE_PRF_LED_CONTROL_OFF	LED OFF

(d) SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE

SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE			
- switch state CCCD (Client Characteristic Configuration Descriptor)に対して Write が行われたことを通知するイベントです。			
Parameters:			
uint16_t	char_code	SAMPLE_PRF_SWITCH_STATE_CCCD_CODE	Write の対象 Character
uint16_t	cccd_val	SAMPLE_PRF_STOP_NTFIND	Notification を無効化
		SAMPLE_PRF_START_NTF	Notification を有効化

5.5.2 定義

(a) SAMPLE_SERVER_EVENT_TYPE

SAMS のイベント識別子の定義です。

```
typedef enum {
  SAMPLE_SERVER_EVENT_ENABLE_COMP = 0,
  SAMPLE_SERVER_EVENT_DISABLE_COMP,
  SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND,
  SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE,
} SAMPLE_SERVER_EVENT_TYPE;
```

(b) SAMPLE_CLIENT_PARAM

SAMS の設定値の定義です。

```
typedef struct {
  uint16_t switch_state_cccd;// switch state CCCD 値
} SAMPLE_CLIENT_PARAM;
```

(c) SAMPLE_SERVER_EVENT_HANDLER

SAMS のイベントハンドラの定義です。

```
typedef void (*SAMPLE_SERVER_EVENT_HANDLER)(SAMPLE_SERVER_EVENT *event);
```

(d) SAMPLE_SERVER_EVENT

SAMS のイベント構造体の定義です。

```
typedef struct {
  SAMPLE_SERVER_EVENT_TYPE type;// イベント種別
  RBLE_STATUS status;     // SAMS 実行結果
  uint16_t conhdl;        // イベントが発生した Connection Handle
  union {
    struct {              // SAMPLE_SERVER_EVENT_DISABLE_COMP
      SAMPLE_SERVER_PARAM param; // SAMS の設定値
    } disable_comp;
    struct {             // SAMPLE_SERVER_EVENT_CHG_LED_CONTROL_IND
      uint8_t value;     // Led Control Characteristic Value の変更後の値
    } change_led_control_ind;
    struct {            // SAMPLE_SERVER_EVENT_WRITE_CHAR_RESPONSE
      uint16_t cccd;    // Switch State CCCD の変更後の値
    } write_char_resp;
  } param;
} SAMPLE_SERVER_EVENT;
```

5.6 Sample Custom Profile for Client Role (SAMC)

SAMC は、BLE プロトコルスタックが提供する GATT API（RBLE_GATT_xxx）をラップし、Sample Custom Service へのアクセスを容易にするための API・Event・構造体などを提供します。Sample Custom Profile for Client role が使用する GATT API は、5.7 章に示すシーケンス図を参照してください。

SAMC は複数台の Peripheral との同時接続に対応しています。

5.6.1 API

API 名の先頭に付加されている「SAMPLE_」は、サンプルプログラム内で定義されていることを示します。本サンプルプログラム以外では使用できません。

(a) SAMPLE_Client_Init

```
void SAMPLE_Client_Init (void)
```

- 本 API は、SAMC の初期化を行います。RBLE_GAP_EVENT_RESET_RESULT イベント発生時に 1 度だけ実行してください。

(b) SAMPLE_Client_Enable

```
RBLE_STATUS SAMPLE_Client_Enable (uint16_t conhdl,
  SAMPLE_CLIENT_CON_TYPE con_type, SAMPLE_CLIENT_CONTENT *content,
  SAMPLE_CLIENT_EVENT_HANDLE callback)
```

- 本 API は、SAMC を有効化します。API の実行完了は、SAMPLE_CLIENT_EVENT_ENABLE_COMP により通知されます。
- con_type は、SAMS が提供する Service の Discovery の実施可否を指定します。con_type に RBLE_PRF_CON_DISCOVERY を指定した場合、Discovery を実施します。RBLE_PRF_CON_NORMAL を指定した場合、Discovery を実施せず、content で指定したハンドル値を使用します。
- content は、Service の Attribute 情報です。con_type に RBLE_PRF_CON_NORMAL が指定された場合に使用します。
- callback は、SAMC に関するイベントが発生した際に呼び出されます。

Parameters:

uint16_t	conhdl	Connection Handle	
uint16_t	con_type	RBLE_PRF_NORMAL	Discovery を実施
		RBLE_PRF_DISCOVERY	Discovery を実施せず content を使用
SAMPLE_CLIENT_CONTENT *	content	Service の Handle 値や Property	
SAMPLE_CLIENT_EVENT_HANDLE	callback	SAMC Event Handler	

Return:

RBLE_OK	Success
RBLE_PARAM_ERR	Parameter Error
RBLE_STATUS_ERROR	Status Error

(c) SAMPLE_Client_Disable

RBLE_STATUS SAMPLE_Client_Disable (uint16_t conhdl)		
- 本 API は、SAMC を無効化します。API の実行完了は、SAMPLE_CLIENT_EVENT_DISABLE_COMP により通知されます。		
Parameters:		
uint16_t	conhdl	Connection Handle
Return:		
RBLE_OK	Success	
RBLE_STATUS_ERROR	Status Error	

(d) SAMPLE_Client_Write_Led_Control

RBLE_STATUS SAMPLE_Client_Write_Led_Control (uint8_t value)			
- 本 API は、LED Control Characteristic Value に対して Write を発行します。			
Parameters:			
uint8_t	value	SAMPLE_PRF_LED_CONTROL_ON	LED ON
		SAMPLE_PRF_LED_CONTROL_OFF	LED OFF
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

(e) SAMPLE_Client_Write_Char

RBLE_STATUS SAMPLE_Client_Write_Char (uint16_t char_code, uint16_t cccd_val)			
- 本 API は、char_code で指定した CCCD(Client Characteristic Configuration Descriptor)に対して Write を発行します。			
Parameters:			
uint16_t	char_code	SAMPLE_CLIENT_WR_SWITCH_STATE_CCCD_CODE	Write を発行する対象
uint16_t	cccd_val	RBLE_PRF_STOP_NTFIND	Notification を無効化
		RBLE_PRF_START_NTF	Notification を有効化
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

(f) SAMPLE_Client_Read_Char

RBLE_STATUS SAMPLE_Client_Read_Char (uint16_t conhdl, uint8_t char_code)			
- 本 API は、char_code で指定した Characteristic に対して Read を発行します。			
Parameters:			
uint16_t	conhdl	Connection Handle	
uint8_t	char_code	SAMPLE_CLIENT_RD_LED_CONTROL_CODE	LED Control Char Value
		SAMPLE_CLIENT_RD_SWITCH_STATE_CCCD_CODE	Switch state CCCD
Return:			
RBLE_OK	Success		
RBLE_STATUS_ERROR	Status Error		

5.6.2 Event

(a) SAMPLE_CLIENT_EVENT_ENABLE_COMP

SAMPLE_SERVER_ENABLE_COMP		
- SAMC の有効化の完了を通知するイベントです。		
Parameters:		
RBLE_STATUS	status	SAMC 有効化結果
SAMPLE_CLIENT_CONTENT*	param	Discovery により得た Attribute 情報
uint16_t	conhdl	Connection Handle

(b) SAMPLE_CLIENT_EVENT_DISABLE_COMP

SAMPLE_CLIENT_DISABLE_COMP		
- SAMC の無効化の完了を通知するイベントです。		
Parameters:		
RBLE_STATUS	status	SAMC の無効化結果
uint16_t	conhdl	Connection Handle

(c) SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND

SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND		
- Switch State Characteristic Value の Notification 通知の受信イベントです。		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	value	受信した Switch State Characteristic Value の値

(d) SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE

SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE		
- Write コマンドの実行完了イベントです。		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	status	Write Command の実行結果

(e) SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE

SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE		
- Read コマンドの実行完了イベントです。		
Parameters:		
uint16_t	conhdl	Connection Handle
uint8_t	status	Read Command の実行結果
RBLE_GATT_INFO_DATA	data	Read した値

5.6.3 定義

(a) SAMPLE_CLIENT_EVENT_TYPE

SAMC のイベント識別子の定義です。

```
typedef enum {
  SAMPLE_CLIENT_EVENT_ENABLE_COMP,
  SAMPLE_CLIENT_EVENT_DISABLE_COMP,
  SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND,
  SAMPLE_CLIENT_EVENT_WRITE_CHAR_RESPONSE,
  SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE,
} SAMPLE_CLIENT_EVENT_TYPE;
```

(b) SAMPLE_CLIENT_EVENT_HANDLE

SAMC のイベントハンドラです。

```
typedef void (*SAMPLE_CLIENT_EVENT_HANDLE)(SAMPLE_CLIENT_EVENT *event);
```

(c) SAMPLE_CLIENT_CONTENT

Discovery により得た Attribute 情報を格納する構造体です。

```
typedef struct {
  uint16_t start_hdl;               // Service の開始 Handle
  uint16_t end_hdl;                 // Service の終了 Handle
  uint16_t switch_state_char_hdl;   // Switch State Char の Handle
  uint16_t switch_state_val_hdl;    // Switch State Value Char の Handle
  uint16_t switch_state_prop;       // Switch State Char の Property
  uint16_t switch_state_cccd_hdl;   // Switch State Char の CCCD
  uint16_t led_control_char_hdl;    // LED Control Char の Handle
  uint16_t led_control_val_hdl;     // LED Control Value Char の Handle
  uint16_t led_control_prop;        // LED Control Char の Property
} SAMPLE_CLIENT_CONTENT;
```

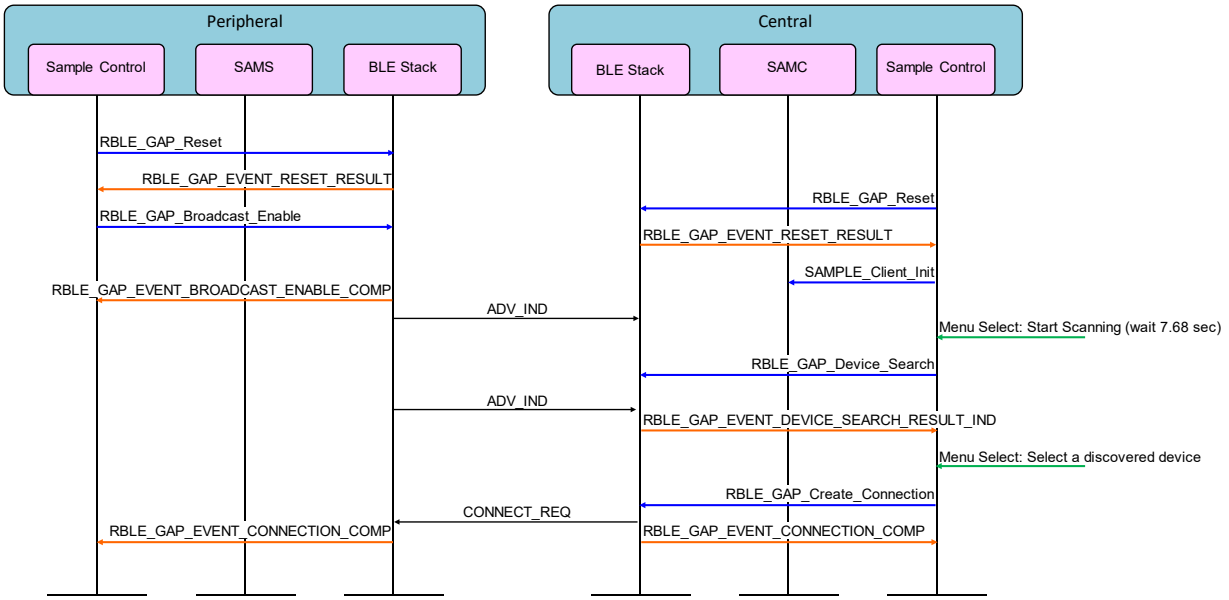
(d) SAMPLE_CLIENT_EVENT

SAMC のイベント構造体の定義です。

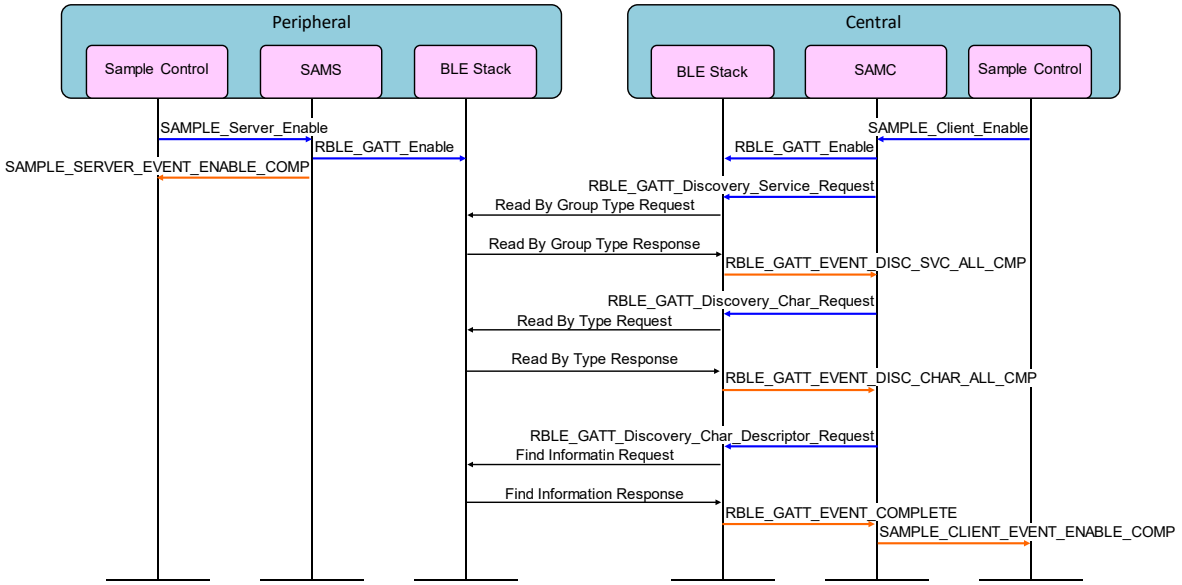
```
typedef struct {
  SAMPLE_CLIENT_EVENT_TYPE type;// イベント種別
  RBLE_STATUS status;         // SAMC 実行結果
  uint16_t conhdl;            // イベントが発生した Connection Handle
  union {
    struct {            // SAMPLE_CLIENT_EVENT_ENABLE_COMP
      SAMPLE_CLIENT_CONTENT samc;   // Service の Discovery 結果
  } enable_comp;
  struct {          // SAMPLE_CLIENT_EVENT_SWITCH_STATE_IND
    uint8_t value;         // Notification により通知された値
  } switch_state_ind;
  struct {          // SAMPLE_CLIENT_EVENT_READ_CHAR_RESPONSE
    uint8_t value;         // Read 結果
  } read_char_resp;
} param;
} SAMPLE_CLIENT_EVENT;
```

5.7 シーケンス図

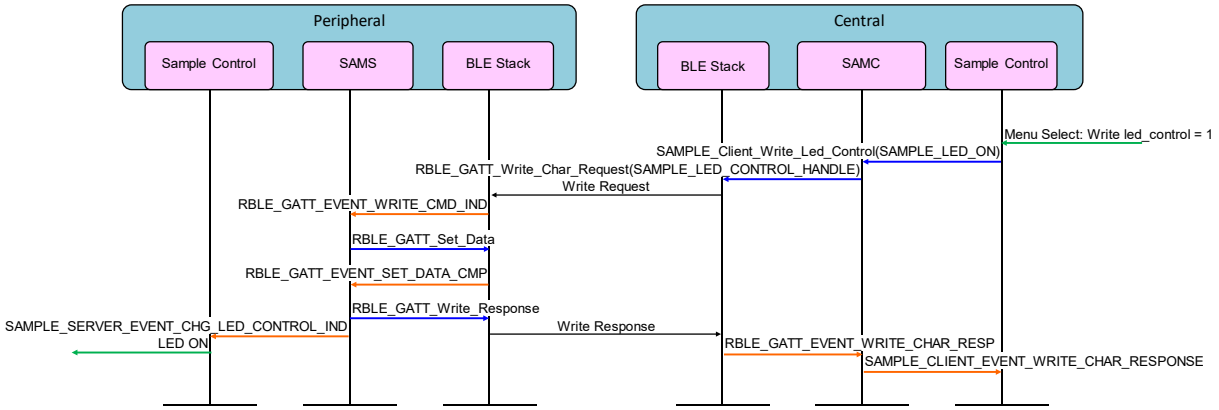
・ 起動から接続の確立



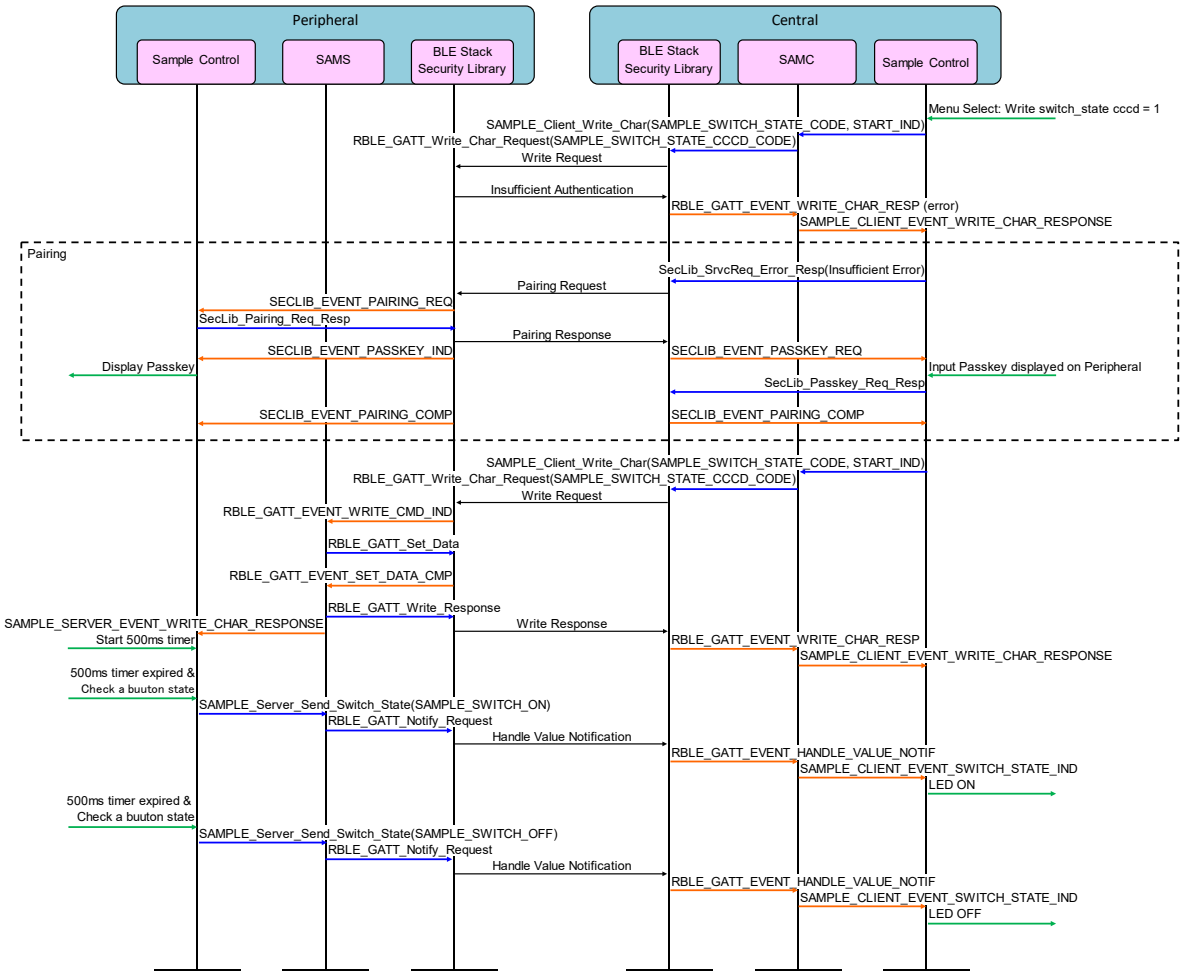
・ Sample Custom Profile の有効化



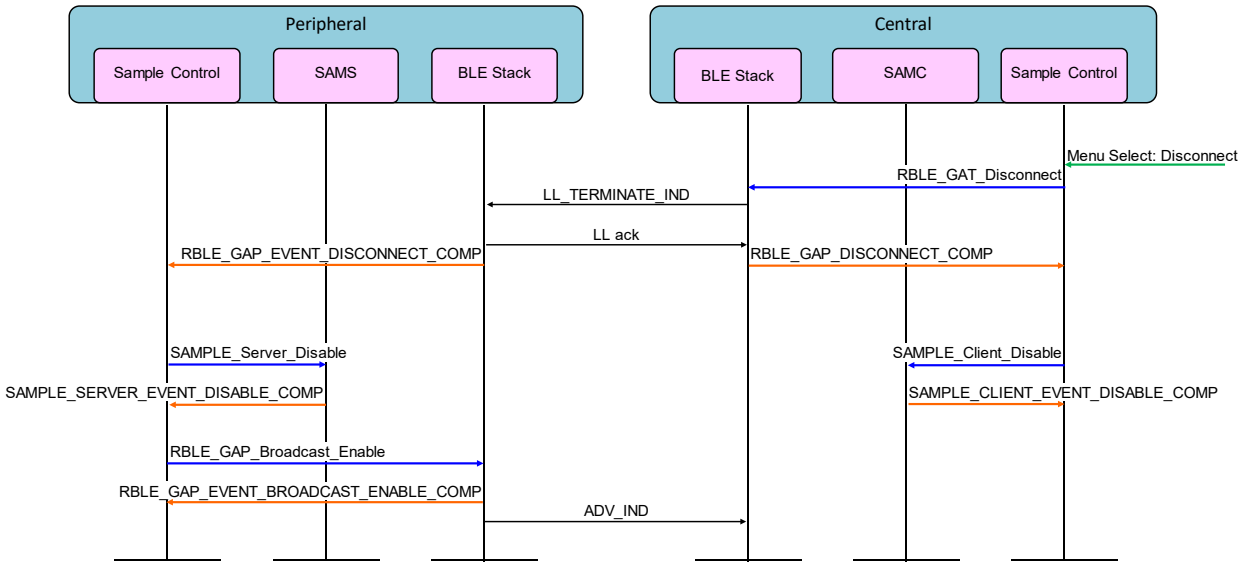
・ LED4 の制御



・ SW4 状態の通知



・ 接続の切断および Sample Custom Profile の無効化



6. Appendix

6.1 ROM サイズ・RAM サイズ

図 3-1 に示す赤色部分のソフトウェアが使用する ROM サイズ・RAM サイズを表 6-1 に示します。本サイズは、同時接続台数の最大値を 4 台に設定している場合を記載しています。

表 6-1 ROM サイズ・RAM サイズ

Compiler	Central		Peripheral	
	ROM size	RAM size	ROM size	RAM size
RL78 Family C Compiler Package V1 V1.03.00	160,524	12,739	154,400	7,307
Renesas CA78K0R V1.72	131,719	12,699	127,178	7,219

(in bytes)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

https://www.renesas.com/

お問合せ先

https://www.renesas.com/contact/

すべての商標および登録商標は，それぞれの所有者に帰属します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
v1.00	2016.07.14	-	初版
v1.10	2016.10.07	5 14 34	2.1 動作環境 ： サポートしている開発環境の記載の追加、ソフトウェアライブラリの記載が 4.1 章と重複するため削除 4.1 共通手順: IARv2 の記載の追加 6.1 ROM サイズ・RAM サイズ ：IARv2 の記載の追加
V1.20	2017.03.01	 4 6 8 11 12 17 31	いずれもセキュリティ機能の追加に伴う修正 1 概要 2.2 環境構築 2.3 使用方法 3.1 サンプルプログラムの構成 3.2 ファイル構成 5.1.2 同時接続台数設定 5.1.4 セキュリティ設定 5.1.5 保存可能なボンディング情報の件数 5.7 シーケンス図
v1.20	2022.01.31	-	Bluetooth Low Energy プロトコルスタックでの IAR サポート終了に伴う修正。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

RENESAS

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

© 2016 Renesas Electronics Corporation. All rights reserved.
Colophon 4.0