RL78/G1D ビーコンスタック
接続確立とビーコンデータ更新 サンプルプログラム

R01AN3313JJ0111
Rev.1.11
2018.03.30

要旨

本ソフトウェアは Bluetooth® Low Energy に対応した RL78/G1D 上で動作し、ビーコンとして情報発信するための Advertising 動作と、動作設定や Advertising データを更新するためのスマートフォンとの接続動作を実行するサンプルプログラムです。

本サンプルプログラムはビーコンアプリケーションとコネクトアプリケーションを切り替えて実行します。ビーコンアプリケーションは低消費電力で Advertising を実行します。コネクトアプリケーションは GAP Peripheral ロールとして動作し、Pairing による暗号化の確立と、Custom Profile によるデータ通信を実行します。ビーコンの設定と Advertising データを更新するための Custom Profile がサンプルとして実装されており、Custom Profile 仕様は拡張可能です。

動作確認デバイス

RL78/G1D 評価ボード (RTK0EN0001D01001BZ)

関連資料

資料名	資料番号	
	和文	英文
RL78/G1D		
ユーザーズマニュアル ハードウェア編	R01UH0515J	R01UH0515E
RL78/G1D 評価ボード		
ユーザーズマニュアル	R30UZ0048J	R30UZ0048E
E1 エミュレータ		
ユーザーズマニュアル	R20UT0398J	R20UT0398E
ユーザーズマニュアル別冊 (RL78 接続時の注意事項)	R20UT1994J	R20UT1994E
Renesas Flash Programmer V3.02 フラッシュ書き込みソフトウェア		
ユーザーズマニュアル	R20UT3841J	R20UT3841E
CC-RL コンパイラ		
ユーザーズマニュアル	R20UT3123J	R20UT3123E
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
RL78/G1D ビーコンスタック		
ユーザーズマニュアル	R01UW0171J	R01UW0171E
RL78 ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01		
ユーザーズマニュアル	R01US0050J	R01US0050E
RL78 ファミリ EEPROM エミュレーション・ライブラリ Pack02		
ユーザーズマニュアル	R01US0068J	R01US0068E

目次

1. 概要

サンプルプログラムはビーコン動作と RF 評価動作のいずれかを実行します。ビーコン動作では、スマートフォンや Bluetooth Low Energy に対応したデバイスを使用し、Advertising パケットの送信とカスタムプロファイルによる Advertising データの更新が可能です。RF 評価動作では、RF テスタを使用し、RL78/G1D の RF 特性評価が可能です。

サンプルプログラムのシステム構成を**図 1-1** に示します。サンプルプログラムはビーコンアプリケーション、コネクトアプリケーション、Direct Test Mode(DTM)アプリケーション、ビーコンスタック、Bluetooth Low Energy(BLE)プロトコルスタック、コードフラッシュライブラリ、データフラッシュライブラリで構成されます。またサンプルプログラムは RL78/G1D 評価ボードで動作します。

ビーコンアプリケーションは、ビーコンスタックを使用し、情報発信のための Advertising パケット送信を実行します。

コネクトアプリケーションは、 BLE プロトコルスタックを使用し、対向デバイスとの接続確立とカスタムプロファイルによる通信を実行します。また、コードフラッシュライブラリとデータフラッシュライブラリを使用し、フラッシュメモリへのデータ書き込みを実行します。ビーコン設定はコードフラッシュメモリに書き込まれ、Pairing 情報はデータフラッシュメモリに書き込まれるため、電源切断後も保持されます。

DTM アプリケーションは、BLE プロトコルスタックを使用し、RF 特性評価のための Direct Test Mode を実行します。

ビーコンスタックは、Advertising 機能を実行するための API をアプリケーションに提供します。

BLE プロトコルスタックは、Bluetooth Low Energy 機能を実行するための API をアプリケーションに提供します。

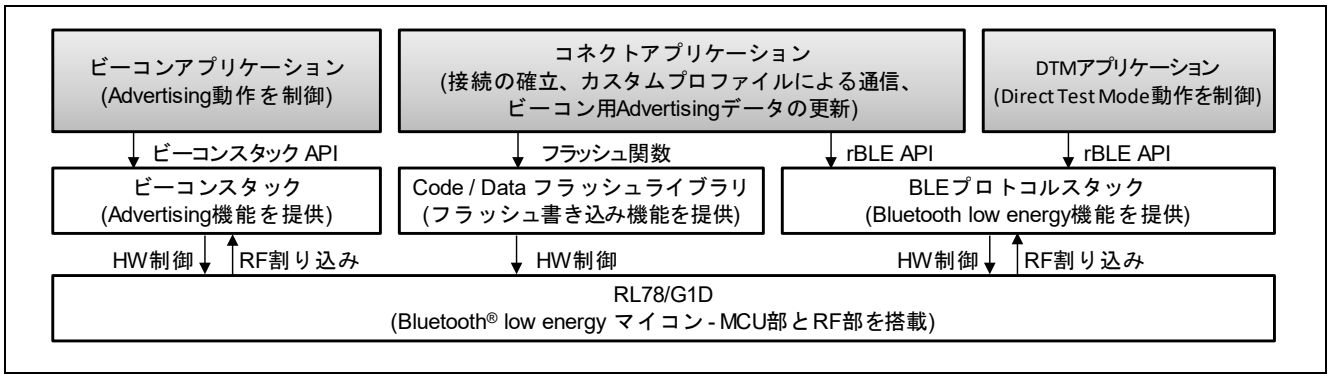


図 1-1　システム構成

ビーコンアプリケーション、コネクトアプリケーション、DTM アプリケーションの仕様については 5 章「仕様」を参照してください。

ビーコンスタックの仕様については『RL78/G1D ビーコンスタック ユーザーズマニュアル』(R01UW0171)を参照してください。

BLE プロトコルスタックの仕様については『Bluetooth Low Energy プロトコルスタック ユーザーズマニュアル』(R01UW0095)、『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

評価ボードの詳細については『RL78/G1D 評価ボード ユーザーズマニュアル』(R30UZ0048)を参照してください。

1.1　ビーコン動作

ビーコン動作の概要を**図 1-2** に示します。

評価ボードの DIP スイッチ SW6-1 を OFF に設定して電源供給を開始すると、ビーコンアプリケーションが起動します。ビーコンアプリケーションは、ビーコンスタックを利用して Non-connectable undirected Advertising パケットを送信します。評価ボードのスイッチ SW2 を押下すると、ビーコンアプリケーションが終了し、コネクトアプリケーションが起動します。コネクトアプリケーションは、BLE プロトコルスタックを利用して Connectable undirected Advertising パケットを送信し、対向デバイスとの接続が可能となります。評価ボードのスイッチ SW2 を再押下または 30 秒間接続がなければ、コネクトアプリケーションが終了し、再度ビーコンアプリケーションを起動します。

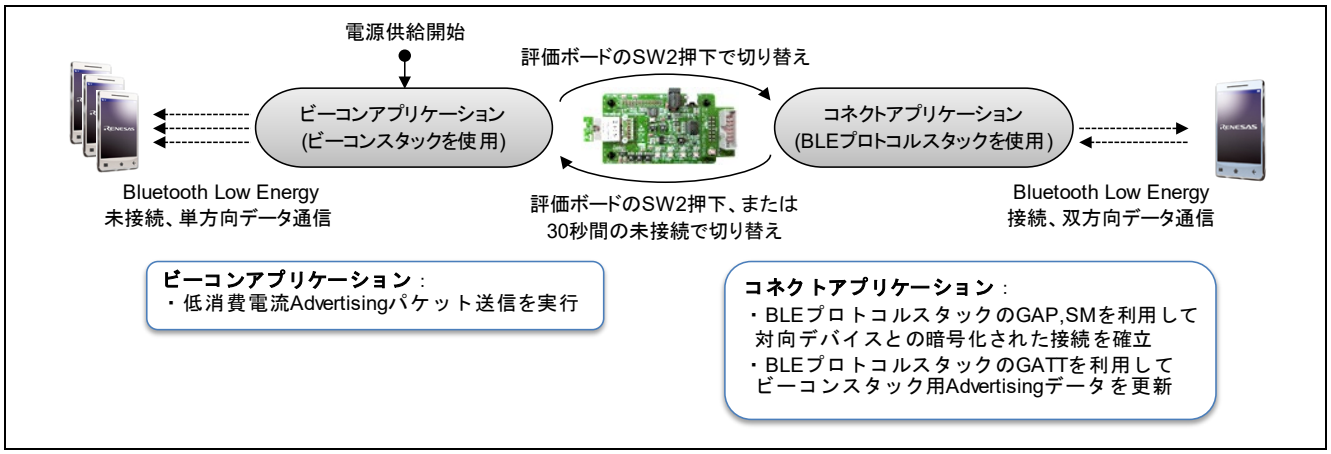


図 1-2　ビーコン動作の概要

1.2　RF 評価動作

RF 評価動作の概要を**図 1-3** に示します。

評価ボードの DIP スイッチ SW6-1 を ON に設定して電源供給を開始すると、DTM アプリケーションが起動します。DTM アプリケーションは、RF テストコマンドの UART 入力により Direct Test Mode を実行し、実行結果を RF テストイベントとして UART 出力します。

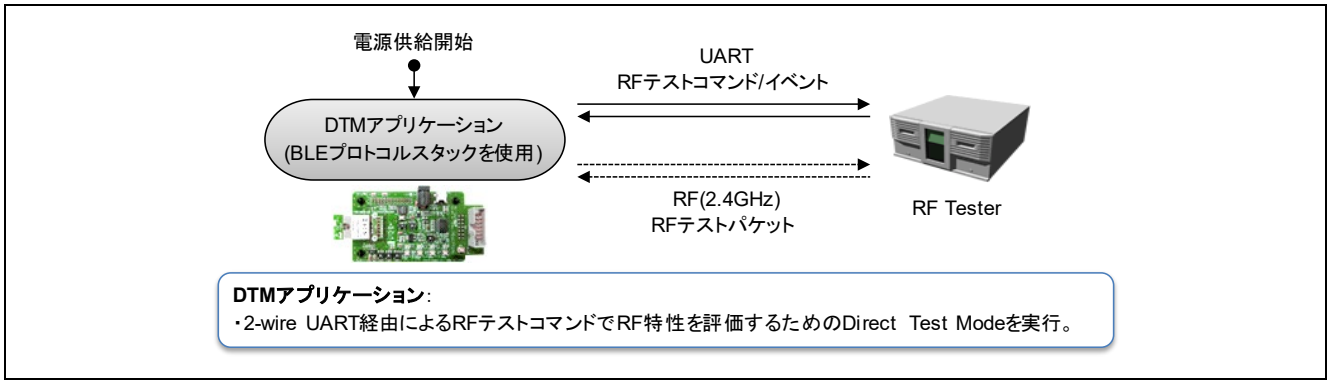


図 1-3　RF 評価動作の概要

2. 動作環境

サンプルプログラムのビルドと動作確認で使用する環境を示します。

- ハードウェア環境
  - ホストマシン
    - 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 評価ボードの接続用)

- デバイス
    - 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)
    - iOS デバイス または Android デバイス

- 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 Service Pack1
    - Renesas CS+ for CC V5.00.00 / Renesas CC-RL V1.04.00
      または Renesas e^2 studio Version 5.3.0.023 / Renesas CC-RL V1.04.00
    - Renesas Flash Programmer v3.02.01
    - Tera Term Pro (またはシリアルポートと接続可能なターミナルソフト)
    - 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。その際にはドライバを以下から入手してください。
    - FTDI (Future Technology Devices International) - Drivers
      http://www.ftdichip.com/Drivers/D2XX.htm

- ソフトウェアライブラリ
    - BLE プロトコルスタック : Bluetooth Low Energy Protocol Stack V1.20
    - ビーコンスタック : RL78/G1D Beacon Stack V2.10
    - コードフラッシュライブラリ : Flash Self Programming Library Type01 Ver2.21
    - データフラッシュライブラリ : EEPROM Emulation Library Pack02 for CC-RL Compiler Ver1.01

    上記のソフトウェアライブラリはルネサス WEB サイトからダウンロード可能です。ライブラリの入手については 4.1 節「ライブラリの入手」を参照してください。

3. ファイル構成

サンプルプログラムにはビーコンスタックライブラリとビーコンアプリケーション、コネクトアプリケーション、DTM アプリケーションのソースコードが含まれます。なお下記のライブラリは含まれません。これらのライブラリは入手後、適切なフォルダに配置する必要があります。

- BLE プロトコルスタックライブラリ
- コードフラッシュライブラリ
- データフラッシュライブラリ

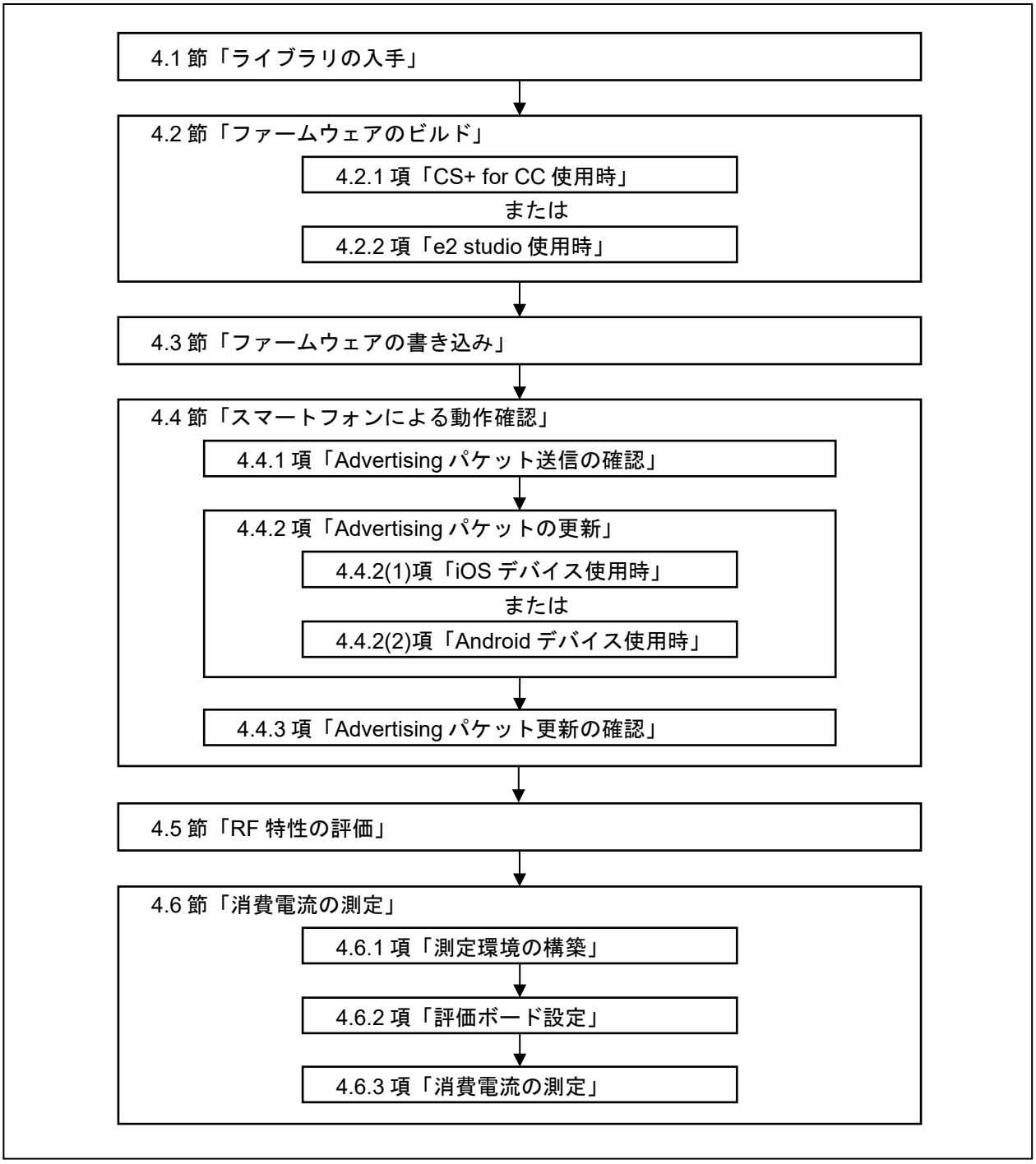
サンプルプログラムのリリースパッケージに含まれるファイルとフォルダの構成を以下に示します。

```
RL78G1D_BeaconCombination
  ├ROM_File
  │    R5F11AGJ_BcnCmb.hex                  サンプルプログラム – ファームウェア (R5F11AGJ)
  │    R5F11AGJ_BcnCmb_no_sw.hex            サンプルプログラム – ファームウェア (R5F11AGJ) (EVABOARD_SWITCH_EN=0)
  ├RUC_File
  │    r5f11agg_syscfg.ruc                  システム動作設定 - ユニークコードファイル (R5F11AGG)
  │    r5f11agh_syscfg.ruc                  システム動作設定 - ユニークコードファイル (R5F11AGH)
  │    r5f11agj_syscfg.ruc                  システム動作設定 - ユニークコードファイル (R5F11AGJ)
  └Project_Source
    ├library
    │  │  r_arch.h                          アーキテクチャ - ヘッダファイル
    │  │  r_compiler.h                      コンパイラ依存 - ヘッダファイル
    │  │  r_iodefine.h                      SFR 定義 - CC-RL 用ヘッダファイル
    │  │  r_ll.h                            低レベル処理 - ヘッダファイル
    │  │  r_port.h                          ポートアクセス - ヘッダファイル
    │  ├beacon
    │  │  BLE_BEACON_CC.lib                 ビーコンスタック - CC-RL 用ライブラリ
    │  │  r_bcn_api.h                       ビーコンスタック API - ヘッダファイル
    │  ├protocol
    │  │  │  (empty)                        (BLE プロトコルスタックライブラリを配置)
    │  │  └dummy
    │  │        types.h                     dummy ヘッダファイル
    │  ├codeflash
    │  │     (empty)                        (コードフラッシュライブラリを配置)
    │  └dataflash
    │        (empty)                        (データフラッシュライブラリを配置)
    └application
      ├src
      │  │  cstart.asm                      スタートアップ - CC-RL 用アセンブラファイル
      │  │  r_config.h                      コンフィグレーション - ヘッダファイル
      │  │  r_main.c                        エントリポイント - ヘッダファイル
      │  ├beacon
      │  │  r_beacon_main.c                 ビーコンアプリケーション・メインループ - コードファイル
      │  │  r_beacon_isr.c                  ビーコンアプリケーション 割り込み - コードファイル
      │  │  r_beacon.h                      ビーコンアプリケーション - ヘッダファイル
      │  │  r_beacon.c                      ビーコンアプリケーション - コードファイル
      │  ├connect
      │  │  │  r_connect_main.c             コネクトアプリケーション・メインループ - コードファイル
      │  │  │  r_connect.h                  コネクトアプリケーション - ヘッダファイル
      │  │  │  r_connect.c                  コネクトアプリケーション - コードファイル
      │  │  │  r_profile.h                  Custom Profile - ヘッダファイル
      │  │  │  r_profile.c                  Custom Profile - コードファイル
      │  │  │  r_dtm_main.c                 DTM アプリケーション・メインループ - コードファイル
      │  │  │  r_dtm.h                      DTM アプリケーション - ヘッダファイル
      │  │  │  r_dtm.c                      DTM アプリケーション - コードファイル
      │  │  ├resource
      │  │  │     r_rble_core.h             BLE プロトコルスタック・rBLE Core レイヤリソース - ヘッダファイル
      │  │  │     r_rble_core.c             BLE プロトコルスタック・rBLE Core レイヤリソース - コードファイル
      │  │  │     r_gatt.h                  BLE プロトコルスタック・GATT レイヤリソース - ヘッダファイル
```

r_gatt.c	BLE プロトコルスタック・GATT レイヤリソース - コードファイル
r_host.h	BLE プロトコルスタック・Host レイヤリソース - ヘッダファイル
r_host.c	BLE プロトコルスタック・Host レイヤリソース - コードファイル
r_controller.c	BLE プロトコルスタック・Controller レイヤリソース - コードファイル
r_kernel.h	BLE プロトコルスタック・Kernel リソース - ヘッダファイル
r_kernel.c	BLE プロトコルスタック・Kernel リソース - コードファイル
r_stack.h	BLE プロトコルスタック - ヘッダファイル
optional	
r_optional.c	BLE プロトコルスタック・オプション機能 - コードファイル
r_reserved.c	BLE プロトコルスタック・未使用機能 - コードファイル
driver	
codeflash	
r_codeflash.h	コードフラッシュドライバ - ヘッダファイル
r_codeflash.c	コードフラッシュドライバ - コードファイル
dataflash	
r_dataflash.h	データフラッシュドライバ - ヘッダファイル
r_dataflash.c	データフラッシュドライバ - コードファイル
r_eel_descriptor_t02.h	データフラッシュドライバ EEPROM Emulation descriptor - ヘッダファイル
r_eel_descriptor_t02.c	データフラッシュドライバ EEPROM Emulation descriptor - コードファイル
r_fdl_descriptor_t02.h	データフラッシュドライバ descriptor - ヘッダファイル
r_fdl_descriptor_t02.c	データフラッシュドライバ descriptor - コードファイル
input	
r_input.h	外部入力割り込みドライバ - ヘッダファイル
r_input.c	外部入力割り込みドライバ - コードファイル
plf	
r_plf.h	プラットフォームドライバ - ヘッダファイル
r_plf.c	プラットフォームドライバ - コードファイル
uart	
r_uart.h	UART ドライバ - ヘッダファイル
r_uart.c	UART ドライバ - コードファイル
project	
cs_cc	
BLE_Software	
BLE_Software.mtpj	CS+ for CC 用プロジェクトファイル
R5F11AGG_BcnCmb	
R5F11AGG_BcnCmb.mtsp	CS+ for CC 用サブプロジェクトファイル (R5F11AGG)
R5F11AGH_BcnCmb	
R5F11AGH_BcnCmb.mtsp	CS+ for CC 用サブプロジェクトファイル (R5F11AGH)
R5F11AGJ_BcnCmb	
R5F11AGJ_BcnCmb.mtsp	CS+ for CC 用サブプロジェクトファイル (R5F11AGJ)
e2_cc	
BLE_Software	
R5F11AGG_BcnCmb	
.project	e² studio 用プロジェクト構成ファイル (R5F11AGG)
.cproject	e² studio 用プロジェクト設定ファイル (R5F11AGG)
.info	e² studio 用 IDE 情報ファイル (R5F11AGG)
.DefaultBuildlinker	e² studio 用リンカ設定ファイル (R5F11AGG)
R5F11AGH_BcnCmb	
.project	e² studio 用プロジェクト構成ファイル (R5F11AGH)
.cproject	e² studio 用プロジェクト設定ファイル (R5F11AGH)
.info	e² studio 用 IDE 情報ファイル (R5F11AGH)
.DefaultBuildlinker	e² studio 用リンカ設定ファイル (R5F11AGH)
R5F11AGJ_BcnCmb	
.project	e² studio 用プロジェクト構成ファイル (R5F11AGJ)
.cproject	e² studio 用プロジェクト設定ファイル (R5F11AGJ)
.info	e² studio 用 IDE 情報ファイル (R5F11AGJ)
.DefaultBuildlinker	e² studio 用リンカ設定ファイル (R5F11AGJ)

4. 動作確認

サンプルプログラムの動作確認について示します。動作確認の手順は 6 つのステップ（ライブラリの入手・ファームウェアのビルド・ファームウェアの書き込み・スマートフォンによる動作確認・RF 特性の評価・消費電流の測定）があります。



4.1　ライブラリの入手

サンプルプログラムのファームウェアをビルドする前に、下記のライブラリをルネサス WEB サイトからダウンロードする必要があります。

- BLE プロトコルスタック:
  - Bluetooth Low Energy Protocol Stack V1.20
    https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rl78-family
- コードフラッシュライブラリ:
  - RL78 ファミリフラッシュセルフプログラミングライブラリ Type01 パッケージ Ver.3.00 (CA78K0R/CC-RL コンパイラ用)
    https://www.renesas.com/software-tool/code-flash-libraries-flash-self-programming-libraries
- データフラッシュライブラリ:
  - 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

ライブラリのダウンロード後、下記のファイルをサンプルプログラムの適切なフォルダにコピーします。

- BLE プロトコルスタック:
  - BLE_Software_Ver_x_xx¥RL78_G1D¥Project_Source¥renesas¥lib¥BLE_rBLE_lib_CCRL.lib
  - BLE_Software_Ver_x_xx¥RL78_G1D¥Project_Source¥renesas¥lib¥BLE_HOST_lib_CCRL.lib
  - BLE_Software_Ver_x_xx¥RL78_G1D¥Project_Source¥renesas¥lib¥BLE_CONTROLLER_LIB_CCRL.lib
  - BLE_Software_Ver_x_xx¥RL78_G1D¥Project_Source¥rBLE¥src¥include¥rble_api.h
  - BLE_Software_Ver_x_xx¥RL78_G1D¥Project_Source¥rBLE¥src¥include¥rble.h
- コードフラッシュライブラリ:
  - FSLRL78 Type01¥V2.21B¥CCRL_V2.21¥CCRL¥V2.21¥librl78¥fsl.lib
  - FSLRL78 Type01¥V2.21B¥CCRL_V2.21¥CCRL¥V2.21¥incrl78¥fsl.h
  - FSLRL78 Type01¥V2.21B¥CCRL_V2.21¥CCRL¥V2.21¥incrl78¥fsl_types.h
- データフラッシュライブラリ:
  - EELRL78 Pack02¥V1.01¥librl78¥eel.lib
  - EELRL78 Pack02¥V1.01¥librl78¥fdl.lib
  - EELRL78 Pack02¥V1.01¥incrl78¥eel.h
  - EELRL78 Pack02¥V1.01¥incrl78¥eel_types.h
  - EELRL78 Pack02¥V1.01¥incrl78¥fdl.h
  - EELRL78 Pack02¥V1.01¥incrl78¥fdl_types.h

上記のファイルをサンプルプログラムの下記のライブラリフォルダに配置します。

RL78G1D_BeaconCombination
└Project_Source
└library

├protocol

BLE_rBLE_lib_CCRL.lib	Protocol Stack rBLE Layer - library file
BLE_HOST_lib_CCRL.lib	Protocol Stack Host Layer - library file
BLE_CONTROLLER_LIB_CCRL.lib	Protocol Stack Controller Layer - library file
rble_api.h	Protocol Stack rBLE API - header file
rble.h	Protocol Stack rBLE definitions - header file

├codeflash

fsl.lib	Code Flash Library - library file
fsl.h	Code Flash Library - header file
fsl_types.h	Code Flash Library type definition - header file

└dataflash

eel.lib	Data Flash Library EEPROM Emulation - library file
eel.h	Data Flash Library EEPROM Emulation - header file
eel_types.h	Data Flash Library EEPROM Emulation type definition - header file
fdl.lib	Data Flash Library - library file
fdl.h	Data Flash Library - header file
fdl_types.h	Data Flash Library type definition - header file

4.2　ファームウェアのビルド

4.1 節に記載されたライブラリの入手後、サンプルプログラムファームウェアのビルドが可能となります。サンプルプログラムのビルドには、CS+ for CC または e^2 studio を使用します。

サンプルプログラムをデフォルト設定でビルドすると、リリースパッケージに含まれる HEX 形式のファームウェアファイル R5F11AGJ_BcnCmp.hex が生成されます。またリリースパッケージに含まれるファイルを使用することで、本手順の省略が可能です。

4.2.1　CS+ for CC 使用時

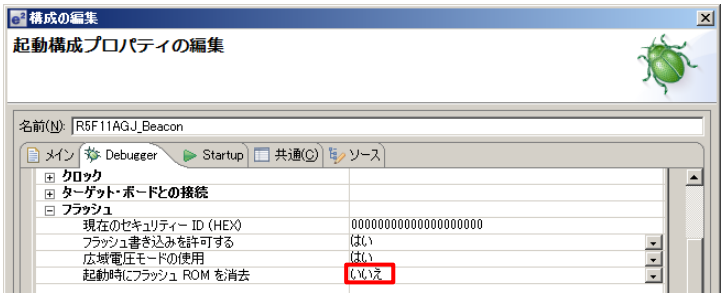
1. CS+ for CC を起動し、[ファイル]→[ファイルを開く]を選択し、下記のパスにある BLE_Software.mtpj プロジェクトファイルを開きます。
   - Project_Source¥application¥project¥cs_cc¥BLE_Software¥
2. [ビルド]→[リビルド・プロジェクト]を選択し、ビルドが成功することを確認します。
3. 下記のパスに R5F11AGJ_BcnCmb.hex が生成されていることを確認します。
   - Project_Source¥application¥project¥cs_cc¥BLE_Software¥R5G11AGJ_BcnCmb¥DefaultBuild¥

4.2.2　e^2 studio 使用時

1. e^2 studio を起動し、下記のパスをワークスペースとして選択します。
   - Project_Source¥
2. [ファイル]→[インポート]を選択し、インポートダイアログを表示します。
3. [一般]→[既存プロジェクトをワークスペースへ]を選択し、[次へ]をクリックします。
4. [ルートディレクトリの選択]で下記のパスを選択し、[プロジェクト]に表示される R5F11AGJ_BcnCmb を選択します。
   - Project_Source¥
5. [終了]をクリックし、インポートダイアログを閉じます。
6. [ようこそ]を閉じます。
7. プロジェクト・エクスプローラーで R5F11AGJ_BcnCmb を選択します。
8. [プロジェクト]→[プロジェクトのビルド]を選択し、ビルドが成功することを確認します。
9. 下記のパスに R5F11AGJ_BcnCmb.hex が生成されていることを確認します。
   - Project_Source¥application¥project¥e2_cc¥BLE_Software¥R5F11AGJ_BcnCmb¥DefaultBuild¥

※e^2 studio のデバッガのデフォルト設定は、起動時にフラッシュメモリを消去します。e^2 studio での開発時、RL78/G1D モジュールに書き込まれている出荷時検査フラグ、デバイスアドレスの消去を防止するため、デバッガ接続時に下記の設定変更を行ってください。また下記の設定変更は、RL78/G1D モジュールと E1 エミュレータの接続を外した状態で行ってください。

- [起動構成プロパティの編集]ダイアログの[Debugger]タブを選択し、[起動時にフラッシュ ROM を消去]で[いいえ]を設定



4.3　ファームウェアの書き込み

サンプルプログラムのファームウェア書き込みには、ホストマシンと E1 エミュレータを**図 4-1** のように使用します。E1 エミュレータと RL78/G1D 評価ボードは、ユーザインタフェースケーブルで接続します。ホストマシンと E1 エミュレータ、ホストマシンと RL78/G1D 評価ボードはそれぞれ USB ケーブルで接続します。

E1 エミュレータの詳細については『E1/E20 エミュレータ ユーザーズマニュアル』(R20UT0398)および『E1/E20 エミュレータ,E2 エミュレータ Lite ユーザーズマニュアル別冊 (RL78 接続時の注意事項)』(R20UT1994)を参照してください。

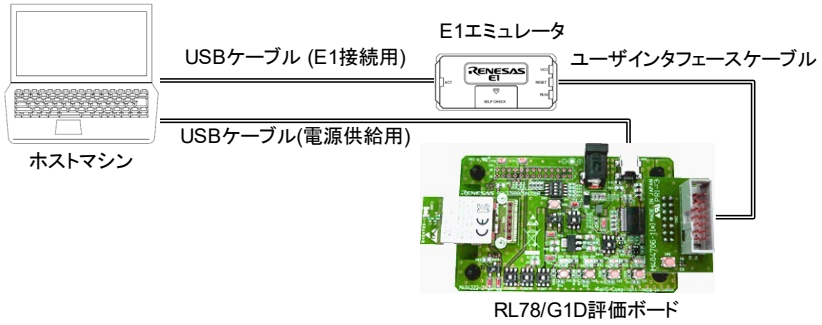


図 4-1　ファームウェア書き込み時の評価ボード操作

サンプルプログラム動作確認時のスライドスイッチ設定を**表 4-1** に示します。

表 4-1　サンプルプログラム動作確認時のスライドスイッチ設定

スイッチ	設定	説明
SW7	2-3 接続(右側) (デフォルト設定)	AC 電源アダプタを接続するための DC ジャック(J1)または USB インタフェース(CN3)からレギュレータ経由で電源供給
SW8	2-3 接続(右側)	USB から電源供給 ※AC 電源アダプタから電源供給する場合は 1-2 接続(左側)
SW9	1-2 接続(左側)	外部拡張インタフェースと接続
SW10	1-2 接続(左側) (デフォルト設定)	モジュールに電源供給
SW11	2-3 接続(右側) (デフォルト設定)	E1 エミュレータ 3.3V 以外から電源供給
SW12	2-3 接続(右側) (デフォルト設定)	(デフォルト固定)
SW13	2-3 接続(右側)	USB 切断

サンプルプログラムファームウェアの書き込みには、Renesas Flash Programmer(RFP)を使用します。

RFP のユニークコード埋め込み機能を使用することにより、同一ファームウェアとデバイスごとに異なるシステム動作設定を書き込むことが可能です。ユニークコード埋め込み機能については『Renesas Flash Programmer V3.02 フラッシュ書き込みソフトウェア ユーザーズマニュアル』(R20UT3841)の 2.3.6 項「[ユニークコード]タブ」を参照してください。

サンプルプログラムファームウェアの RL78/G1D 評価ボードへの書き込み手順を以下に示します。

1. **表 4-1** を参照し、評価ボードのスライドスイッチを設定します。
2. E1 エミュレータを評価ボードに接続後、E1 エミュレータと PC を接続します。
3. 評価ボードを PC または AC-USB 電源アダプタと USB で接続し、電源供給を開始します。
4. RPF を起動し、下記の手順でプロジェクトを作成します。

   ※プロジェクト作成後は、作成したプロジェクトを使用することで本手順を省略可能です。

   4-1. [ファイル]→[新しいプロジェクトを作成]を選択します。

   4-2. [新しいプロジェクトの作成]ダイアログの[プロジェクト情報]で[RL78]を選択し、任意のプロジェクト名を入力後、[接続]をクリックします。

   4-3. ログ出力ウィンドウに「操作が成功しました」と表示されることを確認します。
5. [プログラムファイル]で R5F11AGJ_BcnCmb.hex ファームウェアファイルを指定します。
6. 下記の手順でコードフラッシュメモリの Block254, 255 消去を防止します。

   ※RL78/G1D モジュールでは Block254 に出荷時検査フラグ、Block255 にデバイスアドレスが書き込まれています。

   6-1. [操作設定]タブを選択し、[消去オプション]で[ブロック選択消去]を選択します。

   6-2. [ブロック設定]タブを選択し、Block254, 255 の[Erase]、[P.V]のチェックを外します。

7. システム動作設定ファイルを書き込む場合は、下記の手順でユニークコードとして指定します。

   7-1. [ユニークコード設定]タブを選択します。

   7-2. [有効にする]にチェックを入れます。

   7-3. [ユニークコードファイル]で下記のユニークコードファイルを指定します。

   - RUC_File¥r5f11agj_syscfg.ruc

   7-4. [操作]タブに戻ります。
8. [スタート]を押下して書き込み開始後、「操作が完了しました」と表示されることを確認します。
9. 電源、E1 エミュレータを評価ボードから取り外します。

4.4　スマートフォンによる動作確認

スマートフォンによるサンプルプログラムの動作確認では、3 つのステップ（Advertising パケット送信の確認・Advertising パケットの更新・Advertising パケット更新の確認）があります。

サンプルプログラムの動作確認では、評価ボードのスイッチ(SW)と LED をユーザインタフェースとして使用します。サンプルプログラムが使用する SW と LED を**図 4-2** に示します。DIP スイッチ SW6-1 の状態はサンプルプログラムの動作開始時に読み込みます。SW2 はビーコン動作時に外部入力割り込みのトリガとして使用します。

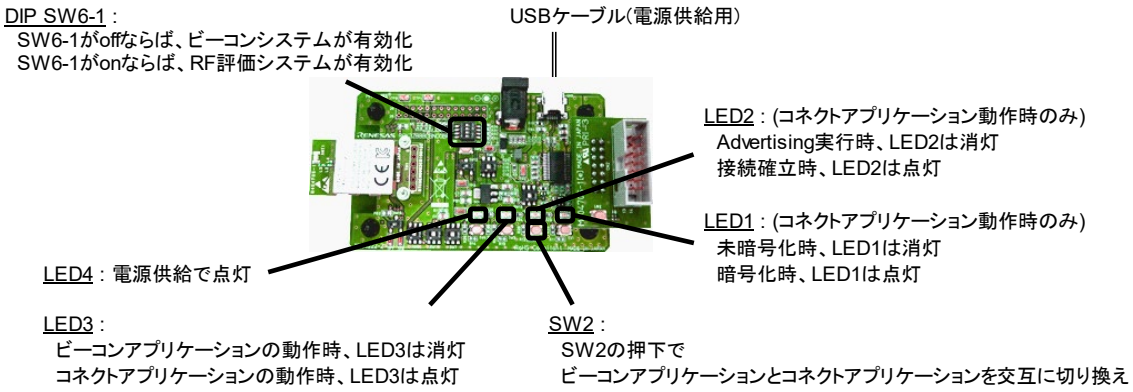
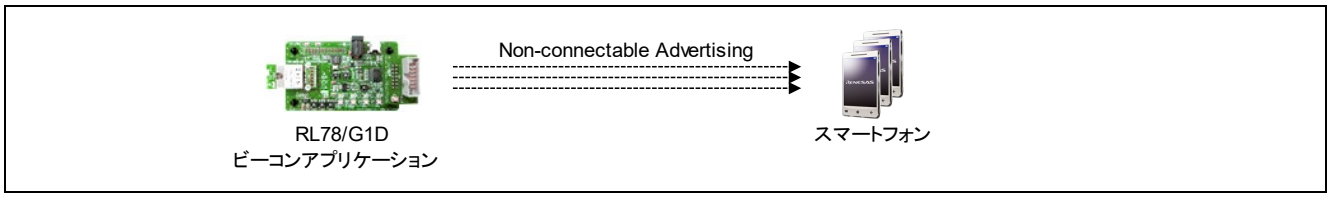


図 4-2　動作確認時の評価ボード操作

4.4.1　Advertising パケット送信の確認

サンプルプログラムのビーコンアプリケーションを実行し、スマートフォンを使用して Advertising パケットの送信動作を確認します。

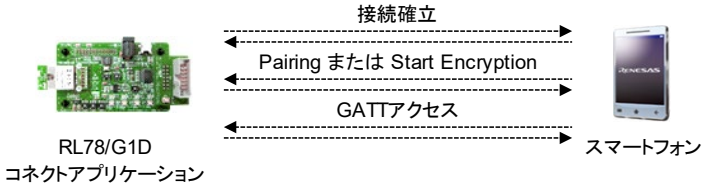


まずサンプルプログラムファームウェアを書き込んだ評価ボードに、DC ジャック(J1)または USB インタフェース(CN3)から電源供給を開始します。DIP スイッチ SW6-1 が OFF の状態で電源供給を開始すると、ビーコンアプリケーションが実行され、LED4 が点灯します。ビーコンアプリケーションはデフォルトで Eddystone-URL パケットを送信します。スマートフォンを使用することで本パケットの受信が可能です。確認手順は iOS デバイスと Android デバイスで共通です。

1. Eddystone-URL パケットを受信するために、下記のアプリをスマートフォンにインストールします。
   - Android デバイス向け, Physical Web - Google Play
     https://play.google.com/store/apps/details?id=physical_web.org.physicalweb
   - iOS デバイス向け, Physical Web - App Store
     https://itunes.apple.com/jp/app/physical-web/id927653608
2. スマートフォンアプリを起動し、画面のプルダウンで Eddystone ビーコンを探索します。
3. サンプルプログラムから Eddystone-URL パケットを受信すると、下記 URL へのリンクが表示されます。
   - Renesas Electronics
     https://www.renesas.com/

4.4.2　Advertising パケットの更新

サンプルプログラムのコネクトアプリケーションを実行し、スマートフォンを使用してビーコンアプリケーションが送信する Advertising データを更新します。



評価ボードの SW2 を押下すると、サンプルプログラムはビーコンアプリケーションとコネクトアプリケーションを交互に切り替えます。コネクトアプリケーションが実行されると、LED4 に加え LED3 が点灯します。コネクトアプリケーションは対向デバイスと接続するための Connectable Advertising パケットを送信します。スマートフォンと RL78/G1D の確立後、GATT アクセスによりビーコンアプリケーションが送信する Advertising データの更新することが可能です。

(1)　iOS デバイス使用時

1. コネクトアプリケーションの Custom Profile にアクセスするために、GATT Client アプリが必要です。例として下記のアプリを iOS デバイスにインストールして使用します。
   - GATTBrowser - App Store
     https://itunes.apple.com/jp/app/gattbrowser/id1163057977
2. 評価ボードの SW2 を押下し、評価ボードの LED3 が点灯することを確認します。
3. スマートフォンのアプリを起動し、デバイスの探索を開始します。
4. RL78/G1D Beacon Updater を選択し、接続を確立します。
5. Renesas Beacon Updater Service の Advertising Data Characteristic を選択します。
6. [Read]ボタンを押下します。
7. [Read]ボタンの下に表示された値を押下します。
8. Advertising Data ダイアログで新しい Advertising データとして下記の値を入力します。
   - Advertising データ値 (https://www.bluetooth.com/の短縮 URL を含む Eddystone-URL)
     1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C547800000000
9. Advertising Data ダイアログで[OK]ボタンを押下します。
10. [Write]ボタンを押下します。
11. ペアリング確認ダイアログで[ペアリング]ボタンを押下します。
12. Pairing の成功により、評価ボードの LED1 が点灯することを確認します。
13. 書き込みの成功により、値が表示されることを確認します。
14. Disconnect を押下します。
15. 評価ボードの LED1 と LED2 が消灯することを確認します。

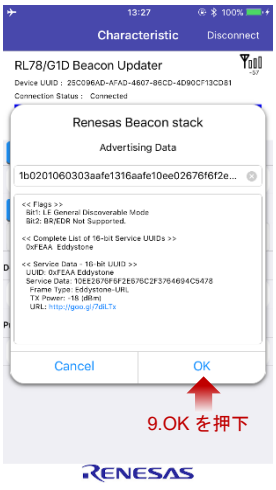

GATT Browser
Menu
RL78/G1D Beacon Updater
4.デバイスを選択

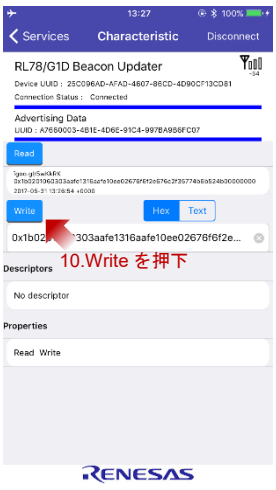

Back
Services
Disconnect
RL78/G1D Beacon Updater
Renesas Beacon Updater Service
Advertising Information
Advertising Data
Scan Response Data
Code Flash Memory Updated Count
Data Flash Memory Updated Count
5.Advertising Data を選択

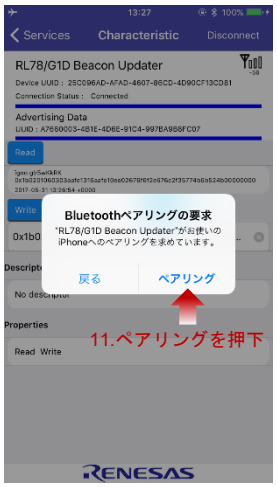

Services
Characteristic
Disconnect
RL78/G1D Beacon Updater
Read
Write
Hex
Text
6.Read を押下
Descriptors
No descriptor
Properties
Read Write

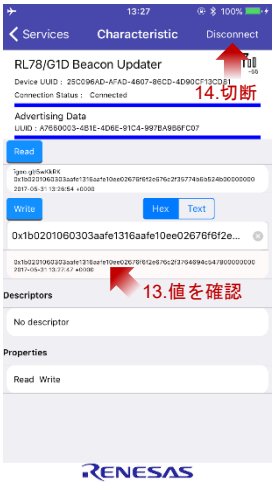

Services
Characteristic
Disconnect
RL78/G1D Beacon Updater
Read
Write
Hex
Text
7.値を押下
Descriptors
No descriptor
Properties
Read Write


Characteristic
Disconnect
RL78/G1D Beacon Updater
Renesas Beacon stack
Advertising Data
1b0201060303aafe1316aafe10ee02676f6f2e...
8.値を入力
Cancel
OK


Characteristic
Disconnect
RL78/G1D Beacon Updater
Renesas Beacon stack
Advertising Data
1b0201060303aafe1316aafe10ee02676f6f2e...
Cancel
OK
9.OK を押下


Services
Characteristic
Disconnect
RL78/G1D Beacon Updater
Read
Write
Hex
Text
10.Write を押下
Descriptors
No descriptor
Properties
Read Write

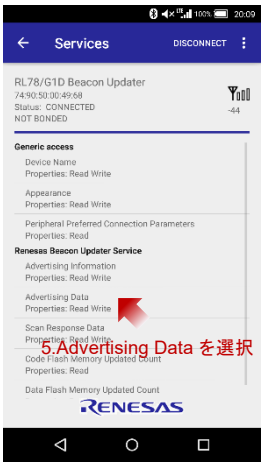

Services
Characteristic
Disconnect
RL78/G1D Beacon Updater
Bluetoothペアリングの要求
"RL78/G1D Beacon Updater"がお使いのiPhoneへのペアリングを求めています。
戻る
ペアリング
11.ペアリングを押下


Services
Characteristic
Disconnect
RL78/G1D Beacon Updater
14.切断
Advertising Data
Read
Write
Hex
Text
0x1b0201060303aafe1316aafe10ee02676f6f2e...
13.値を確認
Descriptors
No descriptor
Properties
Read Write

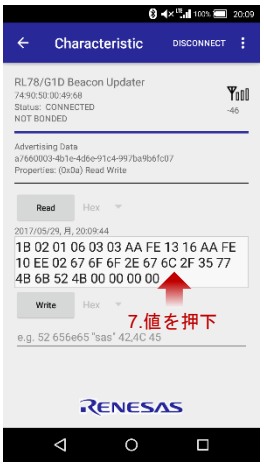
(2) Android デバイス使用時

1. コネクトアプリケーションの Custom Profile にアクセスするために、GATT Client アプリが必要です。例として下記のアプリを Android デバイスにインストールして使用します。
   - GATTBrowser - Google Play
     https://play.google.com/store/apps/details?id=com.renesas.ble.gattbrowser
2. 評価ボードの SW2 を押下し、評価ボード LED3 が点灯することを確認します。
3. スマートフォンのアプリを起動し、デバイスの探索を開始します。
4. RL78/G1D Beacon Updater を選択して、接続を確立します。
5. Renesas Beacon Updater Service の Advertising Data Characteristic を選択します。
6. [Read]ボタンを押下します。
7. [Read]ボタンの下に表示された値を押下します。
8. Advertising Data ダイアログで新しい Advertising データとして下記の値を入力します。
   - Advertising データ値 (https://www.bluetooth.com/の短縮 URL を含む Eddystone-URL)
     0x1B0201060303AAFE1316AAFE10EE02676F6F2E676C2F3764694C5478
9. Advertising Data ダイアログで[OK]ボタンを押下します。
10. [Write]ボタンを押下します。
11. Pairing の成功により、BONDED と表示されることを確認します。
12. 評価ボードの LED1 が点灯することを確認します。
13. [Write]ボタンを再度押下します。
14. 書き込みの成功により、値が表示されることを確認します。
15. DISCONNECT を押下します。
16. 評価ボードの LED1 と LED2 が消灯することを確認します。

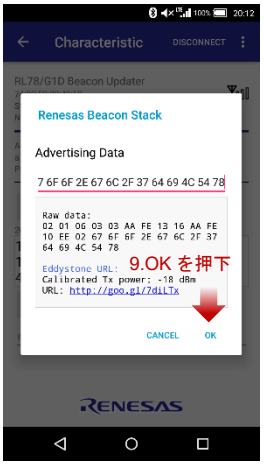

GATTBrowser
SCAN
RL78/G1D Beacon Updater
4.デバイスを選択

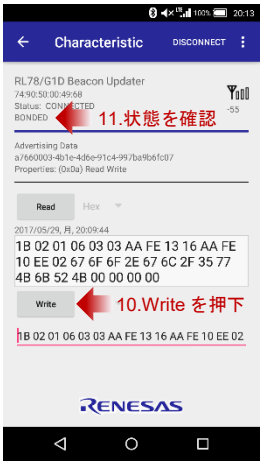

Services
DISCONNECT
RL78/G1D Beacon Updater
5.Advertising Data を選択


Characteristic
DISCONNECT
RL78/G1D Beacon Updater
Read
6.Read を押下
Write


Characteristic
DISCONNECT
RL78/G1D Beacon Updater
1B 02 01 06 03 03 AA FE 13 16 AA FE
10 EE 02 67 6F 6E 67 6C 2F 35 77
4B 6B 52 4B 00 00 00 00
7.値を押下
Write


Characteristic
DISCONNECT
Renesas Beacon Stack
Advertising Data
02 01 06 03 03 AA FE 13 16 AA FE 10
8.値を入力
CANCEL
OK

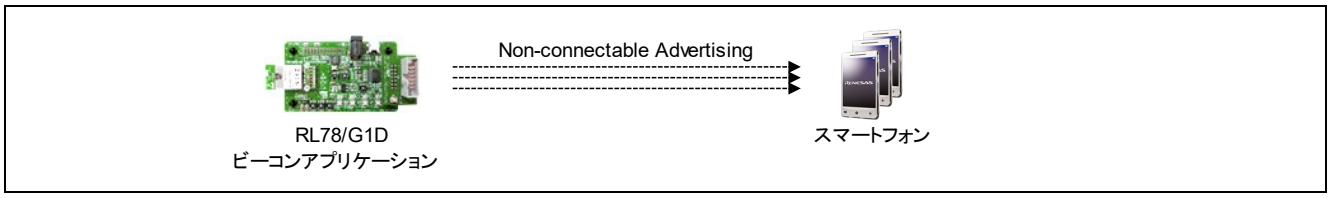

Characteristic
DISCONNECT
Renesas Beacon Stack
Advertising Data
7 6F 6F 2E 67 6C 2F 37 64 69 4C 54 78
9.OK を押下
CANCEL
OK


Characteristic
DISCONNECT
RL78/G1D Beacon Updater
11.状態を確認
Write
10.Write を押下


Characteristic
DISCONNECT
15.接続を切断
RL78/G1D Beacon Updater
Write
13.Write を押下
14.値を確認

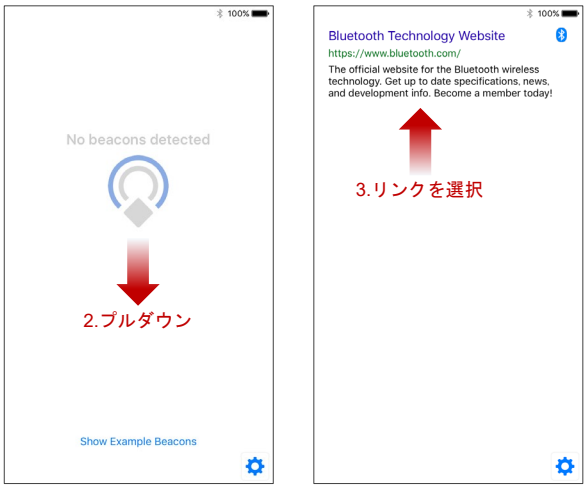
4.4.3　Advertising パケット更新の確認

サンプルプログラムのビーコンアプリケーションを再度実行し、スマートフォンを使用して更新後の Advertising パケットを確認します。



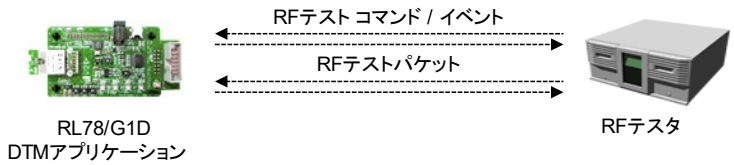
スマートフォンで Custom Profile の Characteristic 値を更新し、接続を切断すると、ビーコンアプリケーションのための Advertising データはコードフラッシュメモリに書き込まれます。更新された Advertising データは、再度スマートフォンで確認が可能です。確認手順は iOS デバイスと Android デバイスで共通です。

1. 評価ボードの SW2 を押下、または MCU をリセット、または評価ボードへの電源供給の停止と再開で、ビーコンアプリケーションを再実行します。
2. 4.4.1 項でインストールしたアプリを起動し、画面をプルダウンしてビーコンの探索を開始します。
3. サンプルプログラムからの新しい Eddystone-URL パケットを受信すると、下記 URL へのリンクが表示されます。
   - Bluetooth Technology Website
     https://www.bluetooth.com/



4.5　RF 特性の評価

サンプルプログラムの DTM アプリケーションを実行し、Direct Test Mode を実行して RL78/G1D の RF 特性を評価します。



評価ボードへの電源供給開始前に DIP スイッチ SW6-1 を ON にすることで、サンプルプログラムは DTM アプリケーションを実行します。RF テストを使用して、RL78/G1D の RF 特性評価が可能です。

1. 評価ボードの DIP スイッチ SW6-1 を ON に切り替えます。
2. 評価ボードの UART TxD0 端子、RxD0 端子、GND 端子を RF テスタに接続します。RL78/G1D と RF テスタのロジックレベルが異なる場合は、ロジックレベル変換器を介して RF テスタに接続します。
3. RF テスタのマニュアルを参照し、**表 4-2** に従って RF テストの UART を設定します。
4. RF テスタのマニュアルを参照し、RF テストから Direct Test Mode を実行します。

表 4-2　UART 設定

設定	値
Baud rate	9600bps
Data bit length	8bit
Parity	None
Stop bit length	1bit
Flow control	None

4.6　消費電流の測定

RL78/G1D 評価ボード（RTK0EN0001D01001BZ）による消費電流の測定方法を以下に示します。RL78/G1D 評価ボードの詳細については『RL78/G1D 評価ボード ユーザーズマニュアル』(R30UZ0048)を参照してください。

4.6.1　測定環境の構築

消費電流の測定に必要な機材を**表 4-3** に示します。

各機材の使用方法の詳細についてはご使用になる各機材のマニュアルを参照してください。

表 4-3　消費電流測定に必要な機材

機材	概要	機材例
電源	RL78/G1D への電力供給	安定化電源、またはバッテリ ※動作電圧の範囲内であること
測定装置	測定結果の確認と出力	オシロスコープ
電圧検出器	RL78/G1D の動作電圧検出	電圧プローブ
電流検出器	RL78/G1D の消費電流検出	クランプ式電流プローブ、または シャント抵抗(推奨値 10Ω)と電圧プローブの組合せ

電流検出器として電流プローブを使用する場合の測定環境を**図 4-3** に示します。評価ボードの端子 TP7-TP8 間の電流を電流プローブで測定した結果が、RL78/G1D の消費電流です。

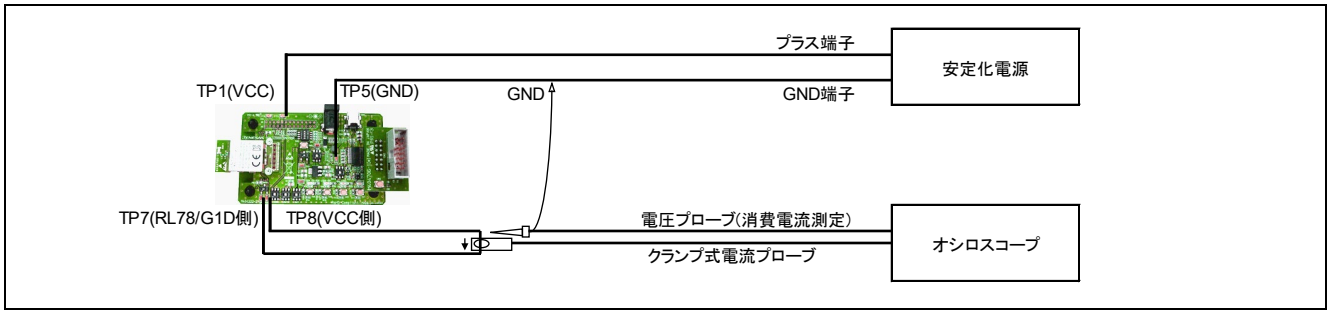


図 4-3　電流プローブを使用した消費電流測定環境

電流検出器としてシャント抵抗と電圧プローブを組み合わせて使用する場合の測定環境を**図 4-4** に示します。評価ボードの端子 TP7-TP8 間に抵抗を挿入し、２つの電圧プローブの測定結果から抵抗両端の電圧を測定します。

抵抗による電圧降下 *dV* は、2 つの電圧プローブの測定結果の差分とします。抵抗による電圧降下 *dV* と抵抗値 *R* による電流 *I=dV/R* の計算結果が、RL78/G1D の消費電流です。

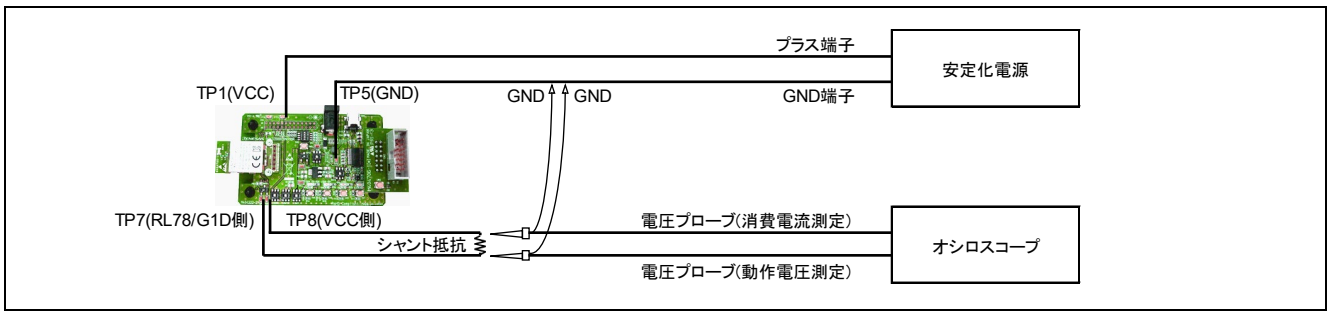


図 4-4　シャント抵抗と電圧プローブを使用した消費電流測定環境

4.6.2　評価ボード設定

表 4-4 を参照して評価ボードの消費電流測定時のスライドスイッチを設定します。

表 4-4　消費電流測定時のスライドスイッチ設定

スイッチ	設定	機能
SW7	1-2 接続(左側)	外部からレギュレータを経由せず電源供給 ※USB から電源供給する場合は 2-3 接続(右側)
SW8	1-2 接続(左側)	TP1,TP5 端子または AC 電源アダプタから電源供給 ※USB から電源供給する場合は 2-3 接続(右側)
SW9	1-2 接続(左側)	外部拡張インタフェースと接続
SW10	2-3 接続(右側)	電源供給ラインをオープン
SW11	2-3 接続(右側) (デフォルト設定)	E1 デバッガ 3.3V 以外から電源供給
SW12	2-3 接続(右側) (デフォルト設定)	(デフォルト固定)
SW13	2-3 接続(右側)	USB 切断

4.6.3　消費電流の測定

消費電流の測定手順を以下に示します。なお、測定手順に示すオシロスコープ設定例は、ビーコンアプリケーション動作時の消費電流を測定するための参考値です。

各機材の測定方法の詳細についてはご使用になる各機材のマニュアルを参照してください。

(1)　周期的な Advertising パケット送信動作の消費電流

1. 電源出力を開始しビーコンアプリケーションを起動します。
2. **図 4-5** を参考にオシロスコープの測定項目を設定します。
   - 測定開始トリガ　　：消費電流プローブの 0.5mA 程度
   - 電流測定レンジ　　：10mA 程度
   - 測定時間　　：測定開始を先頭に 10msec 程度
3. 周期的な Advertising パケット送信動作の電流によりオシロスコープの測定が開始します。

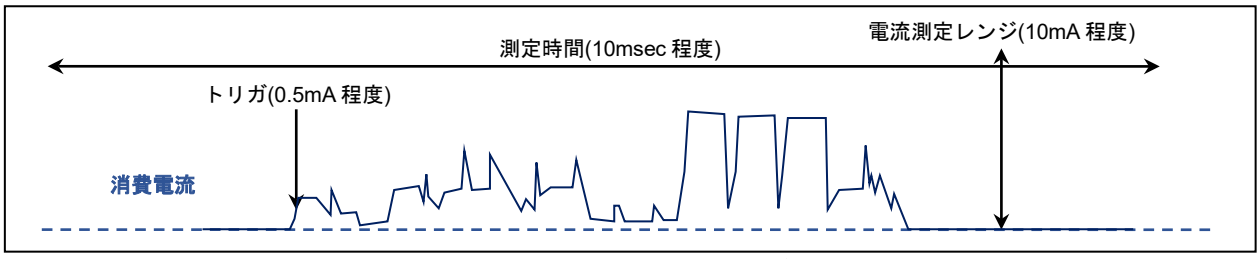


図 4-5　周期的な Advertising パケット送信時の消費電流

5. 仕様

5.1 ビーコンアプリケーション

5.1.1 Non-connectable Advertising

ビーコンアプリケーションはコードフラッシュメモリのシステム動作設定から Advertising 設定と Advertising データを読み込み、情報発信のための Non-connectable Undirected Advertising パケットの送信を開始します。スマートフォンのような対向デバイスは Advertising パケットを受信し、Advertising データに対応したサービスを提供します。ビーコンアプリケーションはアプリケーション終了を要求されると、Advertising を停止し、RF 部の電源供給を停止します。

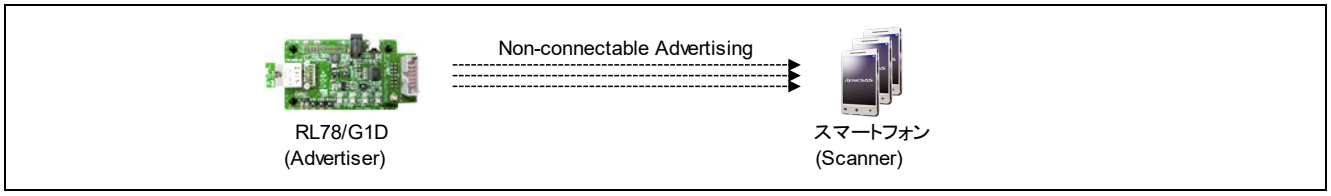


図 5-1　Non-connectable Advertising

ビーコンアプリケーションの状態遷移、Non-connectable Advertising 動作のシーケンスについては 8.1.1 項「ビーコンアプリケーション」、8.2.1(1)項「Initializing & Advertising & RF Powerdown シーケンス」をそれぞれ参照してください。

システム動作設定の詳細については 5.4.1 項「コードフラッシュメモリへのアクセス 」を参照してください。

ビーコンスタックの RF 送信のみ有効化した場合、ビーコンアプリケーションは Non-connectable Undirected Advertising パケットを送信します。

ビーコンアプリケーションのデフォルト Advertising 設定を**表 5-1** に示します。

表 5-1　ビーコンアプリケーションのデフォルト Advertising 設定

Advertiser Address	Public Device Address 12:34:56:78:9A:B0
Advertising Type	Non-connectable Undirected Advertising (ADV_NONCONN_IND)
Advertising Interval	100msec
Advertising Interval Delay	Advertising Interval へのランダムディレイ加算あり
Advertising Channel Map	全チャネル(37,38,39ch)
Advertising Loop Count	回数無制限
Advertising Transmit Power	0dBm ※RL78/G1D の ANT 端子
Advertising Data Count	Advertising データ数 1
Advertising Data [0]～[9]	Advertising データ[0] (ADV_NONCONN_IND ペイロードデータ) Length: 2byte AD Type: <<Flags>> (0x01) AD Data: LE General Discoverable Mode (bit1), BR/EDR Not Supported (bit2) Length: 3byte AD Type: <<Complete List of 16-bit Service Class UUIDs>> (0x03) AD Data: Eddystone (0xFEAA) Length: 19byte AD Type: <<Service Data>> (0x16) AD Data: Eddystone-URL: https://goo.gl/5wKkRK Advertising データ[1]～[9]は未設定
Advertising Event Permission	全イベント通知を許可
Use White List	-

ビーコンスタックの RF 送受信を有効化した場合、ビーコンアプリケーションは Scannable Undirected Advertising パケットを送信し、Scan Request パケットを受信すると、さらに Scan Response パケットを送信します。

RF 送受信の有効化設定については 6.1.2 項「RF 送受信動作」を参照してください。

ビーコンアプリケーションの RF 送受信有効時の Advertising 設定を**表 5-2** に示します。

表 5-2　ビーコンアプリケーションの RF 送受信有効時の Advertising 設定

Advertiser Address	Public Device Address 12:34:56:78:9A:B0
Advertising Type	Scannable Undirected Advertising (ADV_SCAN_IND)
Advertising Interval	100msec
Advertising Interval Delay	Advertising Interval へのランダムディレイ加算あり
Advertising Channel Map	全チャネル(37,38,39ch)
Advertising Loop Count	回数無制限
Advertising Transmit Power	0dBm ※RL78/G1D の ANT 端子
Advertising Data Count	Advertising データ数 2
Advertising Data [0]～[9]	Advertising データ[0] (ADV_SCAN_IND ペイロードデータ) Length: 2byte AD Type: <<Flags>> (0x01) AD Data: LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2) Length: 3byte AD Type: <<Complete List of 16-bit Service Class UUIDs>> (0x03) AD Data: Eddystone (0xFEAA) Length: 19byte AD Type: <<Service Data>> (0x16) AD Data: Eddystone-URL: https://goo.gl/5wKkRK Advertising データ[1] (SCAN_RSP ペイロードデータ) Length: 24byte AD Type: <<Complete Local Name>> (0x09) AD Data: "Renesas RL78/G1D Beacon" Advertising データ[2]～[9]は未設定
Advertising Event Permission	全イベント通知を許可
Use White List	White List を使用しない

Eddystone と Eddystone-URL の仕様については下記の WEB サイトを参照してください。

- Specification for Eddystone, an open beacon format from Google
  https://github.com/google/eddystone
- Specification for Eddystone, an open beacon format from Google - Eddystone-URL
  https://github.com/google/eddystone/tree/master/eddystone-url

5.2　コネクトアプリケーション

5.2.1　Connectable Advertising

コネクトアプリケーションは最初に接続のための Connectable undirected Advertising パケットの送信を開始します。スマートフォンのような対向デバイスは Advertising パケットを受信し、Connection Request を送信することで RL78/G1D と接続を確立します。

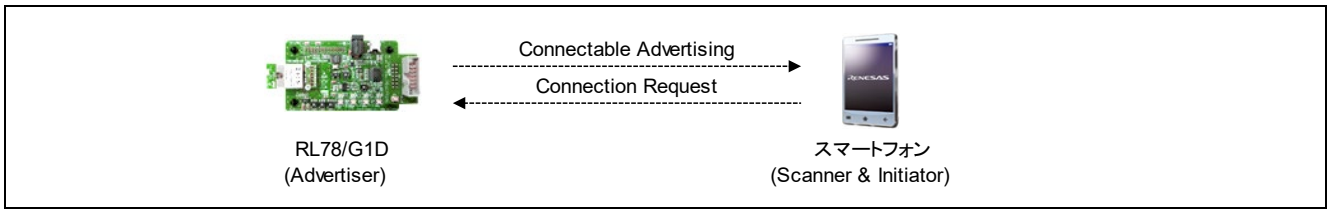


図 5-2　Connectable Advertising

コネクトアプリケーションの状態遷移、Connectable Advertising 動作のシーケンスについては 8.1.2 項「コネクトアプリケーション」、8.2.2(1)項「Initializing & Advertising & Slave Connection (Configurations)をそれぞれ参照してください。

コネクトアプリケーションのデフォルト Advertising 設定を**表 5-3** に示します。

表 5-3　コネクトアプリケーションのデフォルト Advertising 設定

Advertiser Address	Public Device Address 12:34:56:78:9A:B0		
Advertising Type	Connectable undirected Advertising(ADV_IND)		
Advertising Interval Min	30msec		
Advertising Interval Max	30msec		
Advertising Channel Map	全チャネル (37,38,39ch)		
Advertising Data		Length	2byte
		AD Type	<<Flags>> (0x01)
		AD Data	LE General Discoverable Mode (bit1) BR/EDR Not Supported (bit2)
		Length	24byte
		AD Type	<<Complete Local Name>> (0x09)
		AD Data	“RL78/G1D Beacon Updater”
Scan Response Data	未設定		

5.2.2　Pairing / Start Encryption

接続の確立後、Master デバイスからの要求により Pairing シーケンスと Start Encryption シーケンスを実行します。Pairing シーケンスまたは Start Encryption シーケンスの完了後、以降のデータパケットは暗号化されます。

Pairing シーケンスは対向デバイスとの初回接続時に実行されます。シーケンスでは Pairing 情報を交換し、暗号化鍵を生成します。生成した暗号化鍵は、今後の接続での暗号化で必要となります。そのためコネクトアプリケーションはデータフラッシュライブラリを使用して、暗号化鍵をデータフラッシュメモリ書き込みます。

Start Encryption シーケンスは Pairing 済みのデバイスとの接続時に実行されます。コネクトアプリケーションはデータフラッシュから暗号化鍵を読み込み、データパケットの暗号化を開始します。

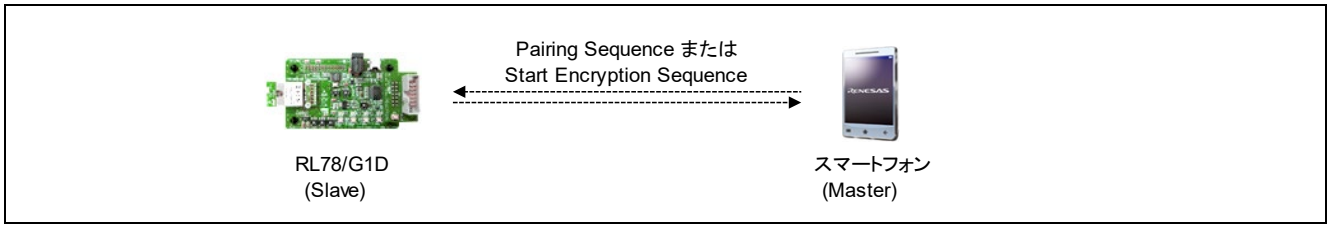


図 5-3　Pairing / Start Encryption

コネクトアプリケーションの状態遷移、Pairing / Start Encryption シーケンスについては 8.1.2 項「コネクトアプリケーション」、8.2.2(2)項「Slave Connection (Pairing) Sequence」と 8.2.2(3)項「Slave Connection (Start Encryption) Sequence」をそれぞれ参照してください。

コネクトアプリケーションのデフォルト Pairing 設定を**表 5-4** に示します。

表 5-4　コネクトアプリケーションのデフォルト Pairing 設定

Bonding	Bondable Mode
Security Mode	Unauthenticated pairing with encryption
Pairing Method	Just Works
IO Capabilities	No Input No Output
OOB Flag	OOB Data not present
Authentication Requirements	No MITM Bonding
Encryption Key Size	128bit
Initiator Key Distribution	None
Responder Key Distribution	Encryption key

5.2.3　Custom Profile Communication

接続中、コネクトアプリケーションは Custom Profile に従ってデータを通信します。まず GATT Client デバイスは Primary Service Discovery、Characteristic Discovery、Characteristic Descriptor Discovery によって GATT Server デバイスの Service と Characteristic の構成を取得します。

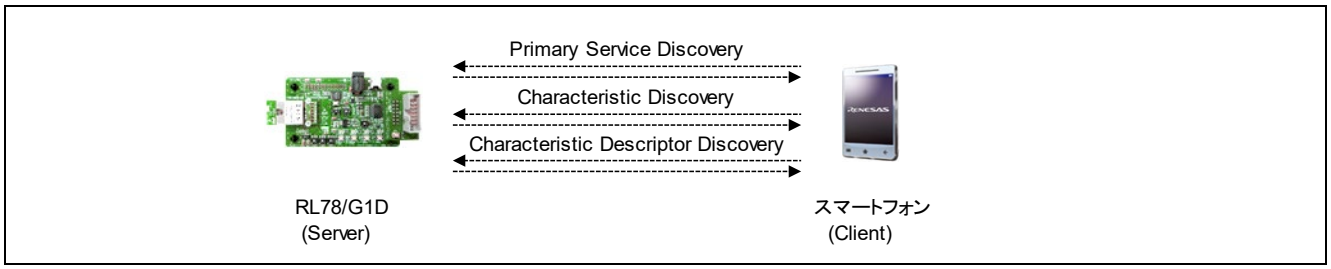


図 5-4　GATT Discovery

その後、GATT Client デバイスは Characteristic Value Read と Characteristic Value Write によって GATT Server デバイスの Characteristic 値の取得と更新を実行します。

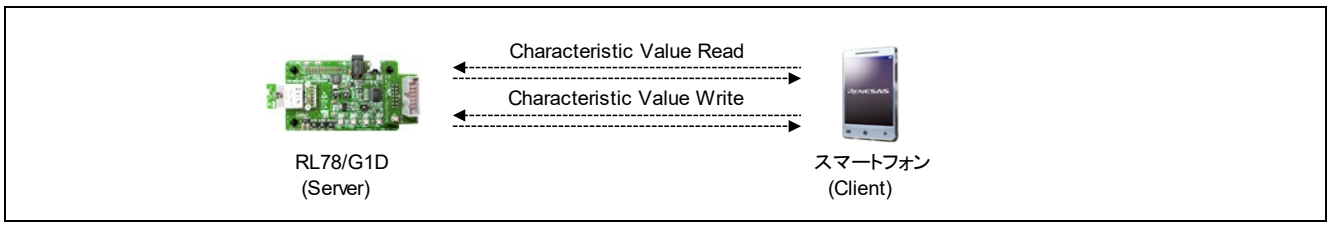


図 5-5　GATT Read / Write

コネクトアプリケーションの状態遷移、GATT アクセスシーケンスについては 8.1.2 項「コネクトアプリケーション」、8.2.2(4)項「Slave Connection (GATT Access) Sequence」をそれぞれ参照してください。

Custom Profile の実装および変更については 6.2.6 項「Custom Profile」を参照してください。

サンプルプログラムに実装するデフォルト Custom Profile 仕様を以下に示します。

➢Custom Profile ロール

- ビーコンデバイスが GATT Server となる
- ビーコンデバイスに接続するデバイスが GATT Client となる。
- GATT Server は Custom Service を保持する。
- GATT Client は Characteristic Value Read で Custom Service の Characteristic 値を取得し、Characteristic Value Write で Custom Service の Characteristic 値を更新する。
- GATT Server は Notification または Indication によるデータの通知は行わない。

➢Custom Profile シナリオ

- ビーコンデバイス上で動作するビーコンスタックの Advertising 情報と Advertising データを、GATT Client デバイスが Custom Service の Characteristic 値として書き込むことで更新する。
- ビーコンデバイスの Advertising 情報と Advertising データはコードフラッシュで保持される。
- コードフラッシュメモリ、データフラッシュメモリの更新回数を、GATT Client デバイスは Custom Service の Characteristic 値として取得する。

コネクトアプリケーションのデフォルト Custom Profile ロールを**図 5-6** に示します。

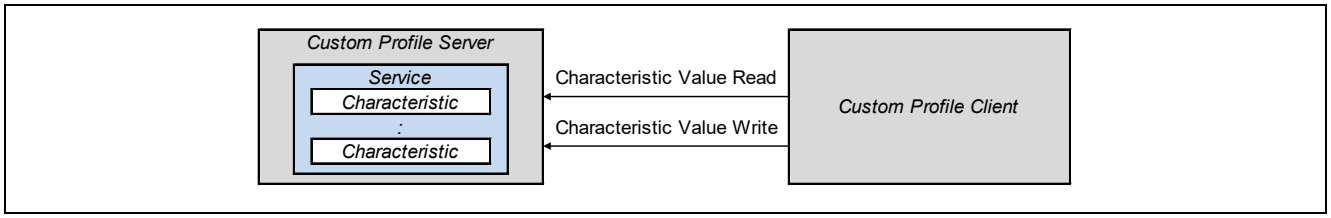


図 5-6　Custom Profile ロール

コネクトアプリケーションのデフォルト Custom Service 仕様を**表 5-5** に示します。

表 5-5　コネクトアプリケーションのデフォルト Custom Service 仕様

	Attribute Handle	Attribute Type	Attribute Value
<<Custom Service : Renesas Beacon Updater Service>>			
	0x000C	Primary Service Declaration (0x2800)	UUID: A7660001-4B1E-4D6E-91C4-997BA9B6FC07
<<Characteristic : Advertising Information>>			
	0x000D	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x000E UUID: A7660002-4B1E-4D6E-91C4-997BA9B6FC07
	0x000E	Advertising Information	ビーコンスタックが定義する Advertising 情報構造体 (18byte)
<<Characteristic : Advertising Data>>			
	0x000F	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x0010 UUID: A7660003-4B1E-4D6E-91C4-997BA9B6FC07
	0x0010	Advertising Data	ビーコンスタックが定義する Advertising データ構造体 (32byte)
<<Characteristic : Scan Response Data>>			
	0x0011	Characteristic Declaration (0x2803)	Properties: Read, Write (0x0A) Value Handle: 0x0012 UUID: A7660006-4B1E-4D6E-91C4-997BA9B6FC07
	0x0012	Scan Response Data	ビーコンスタックが定義する Advertising データ構造体 (32byte)
<<Characteristic : Code Flash Memory Updated Count>>			
	0x0013	Characteristic Declaration (0x2803)	Properties: Read (0x02) Value Handle: 0x0014 UUID: A7660004-4B1E-4D6E-91C4-997BA9B6FC07
	0x0014	Code Flash Memory Updated Count	コードフラッシュメモリ更新回数 (2byte) バイト順序：Least Significant Byte First
<<Characteristic : Data Flash Memory Updated Count>>			
	0x0015	Characteristic Declaration (0x2803)	Properties: Read (0x02) Value Handle: 0x0016 UUID: A7660005-4B1E-4D6E-91C4-997BA9B6FC07
	0x0016	Data Flash Memory Updated Count	データフラッシュメモリ更新回数 (2byte) バイト順序：Least Significant Byte First

Advertising 情報構造体、Advertising データ構造体の仕様については『RL78/G1D ビーコンスタック ユーザーズマニュアル』(R01UW0171)の 4 章「API」を参照してください。

5.3　DTM アプリケーション

5.3.1　Direct Test Mode

DTM アプリケーションは RF テストコマンドと RF テストイベントを通信するために UART を有効化します。テスタからの RF テストコマンドを UART 受信すると、アプリケーションは RF 送信テストまたは RF 受信テストを実行し、RF テストイベントをテスタに UART 送信します。

図 5-7　Direct Test Mode

DTM アプリケーションの状態遷移、Direct Test Mode シーケンスについては 8.1.3 項「DTM アプリケーション」、8.2.3(1)項「Initializing & Transmitter Test & Receiver Test シーケンス」をそれぞれ参照してください。

Direct Test Mode を実行するための RF テストコマンドを**表 5-6** に示します。

表 5-6　RF テストコマンド

RF テストコマンド	パラメータ
LE_RESET	Control (ignored)
LE_RECEIVER_TEST	Frequency, Length, Packet Type
LE_TRANSMITTER_TEST	Frequency, Length, Packet Type
LE_TEST_END	None

Direct Test Mode の実行結果を応答するための RF テストイベントを**表 5-7** に示します。

表 5-7　RF テストイベント

RF テストイベント	パラメータ
LE_TEST_STATUS	Status(Success / Error)
LE_TEST_PACKET_REPORT	Packet Count

RF テスト/イベントの仕様については『Bluetooth Core Specification v4.2』の [Vol. 6, Part F] Section 3.3 を参照してください。

5.4　フラッシュメモリへのアクセス

5.4.1　コードフラッシュメモリへのアクセス

ビーコンアプリケーションとコネクトアプリケーションは、コードフラッシュメモリのファームウェア外部の一部を、システム動作設定を配置するための領域として使用します。システム動作設定には、デバイスごとに異なる必要のあるパラメータを格納します。

ビーコンアプリケーションはデバイスアドレス、デバイスアドレスタイプ、Advertising 情報をシステム動作設定から読み込み、Advertising を開始します。

コネクトアプリケーションはデバイスアドレス、デバイスアドレスタイプ、デバイスネームをシステム動作設定から取得し、BLE プロトコルスタックに設定します。コネクトアプリケーションが接続デバイスから新しい Advertising 情報または Advertising データを受信した場合、アプリケーションはコードフラッシュのシステム動作設定を更新します。

システム動作設定の仕様を**表 5-8** に示します。システム動作設定の配置については 5.10 節「アドレスマップ」を参照してください。

表 5-8　コードフラッシュメモリのシステム動作設定

オフセット	データ		サイズ	読み込み (○:読み込む、×:読み込まない)	書き込み (○:書き込む、×:書き込まない)
0x00	デバイスアドレス (RBLE_BD_ADDR 構造体)		6 byte	○	×
0x06	デバイスアドレスタイプ 0x00: public, 0x01: random (uint8_t 型)		1 byte	○	×
0x07	(reserved)		1 byte	×	×
0x08	デバイスネーム (device_name 構造体)		66 byte		
		namelen	1 byte	○	×
		name	65 byte	○	×
0x4A	Advertising 情報 (RBLE_ADV_INFO 構造体)		18 byte		
		interval	2 byte	○	○
		delay	1 byte	○	○
		ch_map	1 byte	○	○
		loop_cnt	1 byte	○	○
		tx_pwr	1 byte	○	○
		own_addr	6 byte	×	×
		own_addr_type	1 byte	×	×
		data_cnt	1 byte	×	×
		data	2 byte	×	×
		evt_permit	1 byte	×	×
		use_wl	1 byte	×	×
0x5C	Non-connectable Undirected Advertising パケットデータ (RBLE_ADV_DATA 構造体)		32 byte		
		len	1 byte	○	○
		data	31 byte	○	○
0x7C	Scannable Undirected Advertising パケットデータ (RBLE_ADV_DATA 構造体)		32 byte		
		len	1 byte	○	○
		data	31 byte	○	○
0x9C	Scan Response パケットデータ (RBLE_ADV_DATA 構造体)		32 byte		

		len		1 byte	○	○
		data		31 byte	○	○
0xAC	-		-		-	

コードフラッシュメモリの更新にはコードフラッシュライブラリを使用します。コードフラッシュライブラリの詳細については『RL78 ファミリ フラッシュ・セルフ・プログラミング・ライブラリ Type01 ユーザーズマニュアル』(R01US0050)を参照してください。

5.4.2　データフラッシュメモリへのアクセス

コネクトアプリケーションは、電源供給の停止後も保持する必要があるパラメータを、データフラッシュメモリに保存します。コネクトアプリケーションは、データフラッシュライブラリを使用して、データフラッシュメモリの読み込みと書き込みを実行します。

データフラッシュメモリに保持するデータを**表 5-9** に示します。

表 5-9　データフラッシュメモリの格納データ

データ ID	データ		サイズ	読み込み	書き込み
0x02	Paring 情報 (con_pairing_t 構造体)			・コネクトアプリケーション開始時、RAM の変数にコピー	・Pairing 情報が更新されたならば、コネクトアプリケーション切断後に新しい Pairing 情報を書き込み
		対向デバイスアドレス	6 byte		
		対向デバイスアドレスタイプ	1 byte		
		セキュリティ状態	1 byte		
		暗号化鍵情報			
		EDIV (Encrypted Diversifier)	2 byte		
		Random Number	8 byte		
		LTK(Long Term Key)	16 byte		
0x03	フラッシュメモリ更新情報 (con_flashcnt_t 構造体)				・Pairing 情報、Advertising 情報、Advertising データのいずれかが更新されたならば、コネクトアプリケーション切断後に更新回数を書き込み
		コードフラッシュメモリ更新回数	2 byte		
		データフラッシュメモリ更新回数	2 byte		

データフラッシュメモリの読み込みと書き込みにはデータフラッシュライブラリを使用します。データフラッシュライブラリの詳細については『RL78 ファミリ EEPROM エミュレーション・ライブラリ Pack02 ユーザーズマニュアル』(R01US0068)を参照してください。

※データフラッシュの格納データ仕様が Rev.1.00 から変更されました。本サンプルプログラム Rev.1.00 の評価で使用した RL78/G1D 評価ボードにて、引き続き本サンプルプログラム Rev.1.10 を評価する場合、あらかじめ Renesas Flash Programmer(RFP)等を使用してデータフラッシュメモリを消去する必要があります。

RFP による消去手順については『Renesas Flash Programmer V3.02 フラッシュ書き込みソフトウェア ユーザーズマニュアル』(R20UT3841)の 2.3.2 項「[操作設定]タブ」を参照してください。

5.5　BLE プロトコルスタック機能の対応

BLE プロトコルスタックに実装された機能に対するサンプルプログラムの対応を示します。

表 5-10　ソフトウェア構成

ソフトウェア構成	対応/未対応	備考
Embedded 構成	対応	-
Modem 構成	未対応	RSCIP を実行するアプリケーションは実装しない

表 5-11　GAP ロール

GAP ロール	対応/未対応	備考
Broadcaster	対応	-
Observer	未対応	Observer ロールを実行するアプリケーションは実装しない
Central	未対応	Central ロールを実行するアプリケーションは実装しない
Peripheral	対応	-

表 5-12　BLE プロトコルスタック・レイヤ

BLE プロトコルスタック・レイヤ	対応/未対応	備考
LL	対応	-
GAP	対応	-
SM	対応	-
GATT	対応	-
VS	対応	-
Adopted Profile (※1)	未対応	Adopted Profile を実行するアプリケーションは実装しない
Custom Profile (※2)	対応	-

※1: Adopted Profile:
　　Bluetooth SIG によって採択された GATT ベースの Profile です。
※2: Custom Profile:
　　ユーザが独自に定義する GATT ベースの Profile です。

GATT(Generic Attributes)の詳細は下記の WEB サイトを参照してください。
　　https://www.bluetooth.com/specifications/gatt

表 5-13　追加機能

オプション機能	対応/未対応	備考
RWKE	対応	-
SLEEP	対応	-
RSCIP	未対応	RSCIP を実行するアプリケーションは実装しない
DTM 2Wire-UART	対応	-
アダプタブル	未対応	Adaptable を実行するアプリケーションは実装しない
ピーク電流通知	未対応	ピーク電流通知を実行するアプリケーションは実装しない
FW アップデート	未対応	FW アップデートを実行するアプリケーションは実装しない
HCI モニタリング	対応	-
DataFlash リード/ ライト	対応	-
CodeFlash ライト	対応	-

表 5-14　ハードウェア設定

HW 設定	対応/未対応	備考
RF 高速クロック出力	未対応	RF 高速クロックを出力するアプリケーションは実装しない
MCU 外部クロック入力動作	未対応	本機能はビーコンスタックが未対応
RF 外部パワーアンプ	未対応	本機能はビーコンスタックが未対応

5.6　使用ハードウェアリソース

サンプルプログラムのデフォルトでの使用ハードウェアリソースを**表 5-15** に示します。

表 5-15　使用ハードウェアリソース

RL78/G1D MCU 部		
	クロック発生回路	共通 • MCU 部メイン・システム・クロックとして、高速オンチップ・オシレータの 8MHz クロックを使用
		共通 • XT1 発振は使用しない（RF 部スロー・クロックの生成には RF 部内蔵オシレータを使用）
	クロック出力／ブザー出力	共通 • PCLBUZ0 端子から XT1 発振クロックを出力しない
	タイマ･アレイ･ユニット	ビーコンスタック • TM00 使用、動作クロック CK00 を 1MHz に設定
	シリアル･アレイ･ユニット	ビーコンスタック、BLE プロトコルスタック • CSI21 を使用 DTM アプリケーション • UART0 を使用
	DMA コントローラ	ビーコンスタック、BLE プロトコルスタック • DMA2、DMA3 を使用 DTM アプリケーション • DMA0、DMA1 を使用
	割り込み	ビーコンスタック • INTRF、INTDMA2、INTDMA3、INTTM00 を使用 ビーコンアプリケーション、コネクトアプリケーション • INTP5 を使用 DTM アプリケーション • INTDMA0、INTDMA1、INTSR0、INTSRE0、INTST0 を使用
	ポート	共通 • 評価ボードの DIP スイッチ SW6-1 入力に P10 を使用 • 評価ボードのスイッチ SW2 入力に P16 を使用 • 評価ボードの LED4 制御に P60 を使用 • 評価ボードの LED1,2,3,4 制御にそれぞれ P120,P147,P03,P60 を使用
RL78/G1D RF 部		
	DC-DC コンバータ	RF 部内蔵 DC-DC コンバータを使用
	スロー・クロック用 オシレータ	RF 部内蔵オシレータを使用
	GPIO0	入力モード ※変更不可
	GPIO1	入力モード※変更不可
	GPIO2	入力モード、RF 高速クロック出力なし※変更不可
	GPIO3	スロー・クロック用オシレータ使用時：出力 Low モード スロー・クロック用オシレータ未使用時：入力モード(スロー・クロック入力)
RL78/G1D 評価ボード		
	入力機能	共通 • アプリケーションの選択に DIP スイッチ SW6-1 を使用 ビーコンアプリケーション、コネクトアプリケーション • アプリケーションの切り換えにプッシュスイッチ SW2 を使用
	表示機能	ビーコンアプリケーション、DTM アプリケーション • サンプルプログラム開始表示に LED4 を使用 コネクトアプリケーション • データ暗号化開始表示に LED1 を使用 • 接続確立表示に LED2 を使用 • コネクトアプリケーション開始表示に LED3 を使用 • サンプルプログラム開始表示に LED4 を使用

5.7　コンパイラ

ビーコンスタックは下記のコンパイラで生成されています。ビーコンスタックを使用するアプリケーションの開発は CC-RL コンパイラを使用してください。

コンパイラ　　：Renesas CC-RL V1.04.00

5.8　メモリモデル

ビーコンスタックのメモリモデルは、ミディアムモデルです。ビーコンスタックを使用するアプリケーションのコンパイルオプションでは下記を設定してください。

メモリモデル　　：-memory_model=medium

5.9　プログラムサイズ

サンプルプログラムの合計プログラムサイズを**表 5-16** に示します。

ターゲットデバイス　　：R5F11AGJ

コンパイラ　　：Renesas CC-RL V1.04.00

コンパイル設定　　：サンプルプログラムのデフォルト設定

表 5-16　サンプルプログラムの合計プログラムサイズ

ROM サイズ	119,998 byte PROGRAM SECTION + ROMDATA SECTION
RAM サイズ	10,439 byte RAMDATA SECTION (プログラムが関数コールやオート変数確保のために使用するスタックメモリは含みません)

セクション仕様については『CC-RL コンパイラ ユーザーズマニュアル』(R20UT3123)の 6 章「セクション仕様」を参照してください。

5.10　アドレスマップ

サンプルプログラムの RL78/G1D(R5F11AGG)でのアドレスマップを**図 5-8** に示します。

なお下線部は R5F11AGH、R5F11AGJ の値と異なることを示します。

アドレス	領域サイズ	セクション	セクション名
	131,072byte	コードフラッシュメモリ	
0x00000	128byte	ベクタテーブル領域	.vect
0x00080	64byte	CALLT テーブル領域	.callt0
0x000C0	4byte	オプション・バイト領域	.option_byte
0x000C4	10byte	セキュリティ ID 設定領域	.security_id
0x000CE	129,842byte	プログラム領域 (セクション配置は順不同)	
		OCD モニタ	.monitor1, .monitor2
		スタートアップ	BOOT0_TEXT
		ランタイムライブラリ	.RLIB
		標準ライブラリ	.SLIB
		コードフラッシュライブラリ	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
		データフラッシュライブラリ	EEL_CODE, FDL_CODE
		ビーコンスタック	BCN_CONST, BCN_TEXT
		BLE プロトコルスタック	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
		アプリケーション	.const, .constf, .data, .text, .textf
		未使用	-
0x1FC00	156byte	システム動作設定	
0x1FC9C		未使用	
0x20000		使用不可	
0xF0000	2048byte	特殊機能レジスタ(2nd SFR)	
0xF0800		使用不可	
0xF1000	8192byte	データフラッシュメモリ	
0xF3000	40,704byte	Mirror 領域	
0xFCF00	12,064byte	RAM 領域	
		プログラムリソース領域(セクション配置は順不同)	
		アプリケーション	.bss, .dataR
		BLE プロトコルスタック	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
		ビーコンスタック	BCN_BSS
		データフラッシュメモリ	EEL_SDAT , FDL_SDAT
		未使用領域	-
		スタックメモリ	-
0xFFEE0	32byte	汎用レジスタ	
0xFFF00	256byte	特殊機能レジスタ(SFR)	
0xFFFFF			

図 5-8　アドレスマップ (R5F11AGG)

サンプルプログラムの RL78/G1D(R5F11AGH)でのアドレスマップを**図 5-9** に示します。

なお下線部は R5F11AGG、R5F11AGJ の値と異なることを示します。

アドレス	領域サイズ	セクション	セクション名
	196,608byte	コードフラッシュメモリ	
0x00000	128byte	ベクタテーブル領域	.vect
0x00080	64byte	CALLT テーブル領域	.callt0
0x000C0	4byte	オプション・バイト領域	.option_byte
0x000C4	10byte	セキュリティ ID 設定領域	.security_id
0x000CE	195,378byte	プログラム領域 (セクション配置は順不同)	
		OCD モニタ	.monitor1, .monitor2
		スタートアップ	BOOT0_TEXT
		ランタイムライブラリ	.RLIB
		標準ライブラリ	.SLIB
		コードフラッシュライブラリ	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
		データフラッシュライブラリ	EEL_CODE, FDL_CODE
		ビーコンスタック	BCN_CONST, BCN_TEXT
		BLE プロトコルスタック	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
		アプリケーション	.const, .constf, .data, .text, .textf
		未使用	-
0x2FC00	156byte	システム動作設定	
0x2FC9C		未使用	
0x30000		使用不可	
0xF0000	2048byte	特殊機能レジスタ(2nd SFR)	
0xF0800		使用不可	
0xF1000	8192byte	データフラッシュメモリ	
0xF3000	36,608byte	Mirror 領域	
0xFBF00	16,160byte	RAM 領域	
		プログラムリソース領域(セクション配置は順不同)	
		アプリケーション	.bss, .dataR
		BLE プロトコルスタック	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
		ビーコンスタック	BCN_BSS
		データフラッシュメモリ	EEL_SDAT , FDL_SDAT
		未使用領域	-
		スタックメモリ	-
0xFFEE0	32byte	汎用レジスタ	
0xFFF00	256byte	特殊機能レジスタ(SFR)	
0xFFFFF			

図 5-9　アドレスマップ (R5F11AGH)

サンプルプログラムの RL78/G1D (R5F11AGJ)でのアドレスマップを**図 5-10** に示します。

なお下線部は R5F11AGG、R5F11AGH の値と異なることを示します。

アドレス	領域サイズ	セクション	セクション名
	262,144byte	コードフラッシュメモリ	
0x00000	128byte	ベクタテーブル領域	.vect
0x00080	64byte	CALLT テーブル領域	.callt0
0x000C0	4byte	オプション・バイト領域	.option_byte
0x000C4	10byte	セキュリティ ID 設定領域	.security_id
0x000CE	258,866byte	プログラム領域 (セクション配置は順不同)	
		OCD モニタ	.monitor1, .monitor2
		スタートアップ	BOOT0_TEXT
		ランタイムライブラリ	.RLIB
		標準ライブラリ	.SLIB
		コードフラッシュライブラリ	FSL_FCD, FSL_RCD, FSL_BCD, FSL_BECD
		データフラッシュライブラリ	EEL_CODE, FDL_CODE
		ビーコンスタック	BCN_CONST, BCN_TEXT
		BLE プロトコルスタック	RBL_CNST_n, RBL_CODE_n, RBL_CODE_f, HST_CNST_n, HST_CODE_n, HST_CODE_f, CNT_CNST_n, CNT_CODE_n, CNT_CODE_f
		アプリケーション	.const, .constf, .data, .text, .textf
		未使用	-
0x3F400	156byte	システム動作設定	
0x3F49C		未使用	
0x3F800	512byte	予約領域 (RL78/G1D モジュールのみ)	
0x3FC00	6byte	顧客固有情報	
0x3FC06		未使用	
0x40000		使用不可	
0xF0000	2048byte	特殊機能レジスタ(2nd SFR)	
0xF0800		使用不可	
0xF1000	8192byte	データフラッシュメモリ	
0xF3000	32,512byte	Mirror 領域	
0xFAF00	1024byte	セルフ RAM 領域(R5F11AGJ only)	
0xFB300	20,447byte	RAM 領域	
		プログラムリソース領域(セクション配置は順不同)	
		アプリケーション	.bss, .dataR
		BLE プロトコルスタック	RBL_DATA_n, HST_DATA_n , CNT_DATA_n
		ビーコンスタック	BCN_BSS
		データフラッシュメモリ	EEL_SDAT , FDL_SDAT
		未使用領域	-
		スタックメモリ	-
0xFFEE0	32byte	汎用レジスタ	
0xFFF00	256byte	特殊機能レジスタ(SFR)	
0xFFFFF			

図 5-10　アドレスマップ (R5F11AGJ)

6. 設定

サンプルプログラムのハードウェア設定とアプリケーション設定について示します。

6.1 ハードウェア設定

BLE プロトコルスタックとビーコンスタックの主要なハードウェア設定は、r_config.h でマクロとして定義されています。マクロ定義の詳細については以降の記載を参照してください。

Project_Source¥application¥src¥r_config.h, line 34-86

```
34:    /*
35:     * CONFIGURATIONS (NEED TO CHANGE BELOW DEFINES AS NECESSARY)
36:     *********************************************************************************
37:     */
38:    /* MCU Main System Clock (either clock frequency of 4MHz,8MHz,16MHz,32MHz)              */
39:    /*  Note: It is necessary to set Option Bytes Value at Device Setting of Linker Option  */
40:    #define MCU_HOCO_CLK            (8)
41:
42:    /* RF Operation (0:enable both Tx and Rx, 1:enable Tx only) */
43:    /*   Note: This configuration is only for Beacon Stack      */
44:    #define RF_TX_ONLY              (0)
45:
46:    /* RF DC-DC Converter (0:disable DC-DC, 1:enable DC-DC) */
47:    #define RF_DCDC_EN              (1)
48:
49:    /* RF Slow Clock Source (0:RF On-Chip Oscillator, 1:MCU XT1 Oscillator) */
50:    #define RF_SLK_XT1              (0)
51:
52:    /* RF Slow Clock Calibration (0:not execute, 1:execute)                    */
53:    /*   Note: This configuration is only for Beacon Stack                     */
54:    /*       : RF Slow Clock Calibration is only for RF-On_Chip_Oscillator     */
55:    /*       : Protocol Stack always execute RF on chip oscillator calibration */
56:    #define RF_SLK_CAL              (1)
57:
58:    /* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec)     */
59:    /*   Note: This configuration is only for Beacon Stack                  */
60:    /*       : Stabilization Time needs to be optimized for 32MHz resonator */
61:    #define RF_32MHZ_WAIT           (1000)
62:
63:    /* Maximum number of Simultaneous Connections (fixed 1)               */
64:    /*   Note: This configuration is only for BLE Protocol Stack          */
65:    /*       : fixed 1, Connect Application behaves as peripheral device  */
66:    #define MAX_CONNECTION          (1)
67:
68:    /* Packet Monitoring (0:disable Packet Monitor, 1:enable Packet Monitor) */
69:    /*   Note: This configuration is only for BLE Protocol Stack             */
70:    /*       : Packet Monitoring uses UART1 for using output HCI log         */
71:    #define PKTMON_EN               (0)
72:
73:    /* System Configuration Address in CodeFlash memory */
74:    #if   defined(_USE_R5F11AGG)
75:        /* System Configuration is located the last block */
76:        #define SYSCFG_ADDR         (0x1FC00)
77:    #elif defined(_USE_R5F11AGH)
78:        /* System Configuration is located the last block */
79:        #define SYSCFG_ADDR         (0x2FC00)
80:    #elif defined(_USE_R5F11AGJ)
81:        /* System Configuration is located the third last block            */
82:        /* by taking into account the location of RL78/G1D module (RY7011) */
83:        #define SYSCFG_ADDR         (0x3F400)
84:        /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
85:        #define MODCFG_ADDR         (0x3FC00)
86:    #endif
```

6.1.1　MCU メイン・システム・クロック周波数

高速オンチップ・オシレータが生成するクロックは MCU メイン・システム・クロックとして使用され、選択可能な動作周波数は 4、8、16、32MHz のいずれかです。サンプルプログラムでは、MCU メイン・システム・クロックの動作周波数を MCU_HOCO_CLK マクロとオプション・バイトで定義します。デフォルトの動作周波数は 8 (MHz)です。

MCU 動作周波数を変更するには、MCU_HOCO_CLK マクロ値を 4 (MHz)または 8(MHz) または 16 (MHz) または 32 (MHz)に設定します。

Project_Source¥application¥src¥r_config.h, line 38-40

```
38:    /* MCU Main System Clock (either clock frequency of 4MHz,8MHz,16MHz,32MHz)            */
39:    /*  Note: It is necessary to set Option Bytes Value at Device Setting of Linker Option  */
40:    #define MCU_HOCO_CLK            (8)
```

オプション・バイトはリンク・オプション"-user_opt_byte"で設定します。オプション・バイト値については**表 6-1** を参照 してください。

表 6-1　オプション・バイト設定

オプション・バイト設定			クロック周波数	動作モード
000C0	000C1	000C2		
(任意)	(任意)	2B	4MHz	低電圧メインモード
		AA	8MHz	低速メインモード
		E9	16MHz	高速メインモード
		E8	32MHz	

オプション・バイトの詳細については『RL78/G1D ユーザーズマニュアル ハードウェア編』(R01UH0515) の 25 章「オプション・バイト」を参照してください。

MCU 動作周波数によって動作電圧範囲が異なります。動作電圧については『RL78/G1D ユーザーズマニュアル ハードウェア編』(R01UH0515)の 30.2 節「動作電圧」を参照してください。

(1)　CS+ for CC 使用時

CS+ for CC では、下記の手順でオプション・バイト値を変更します。

1　R5F11AGJ_BcnCmb サブプロジェクトの[CC-RL]の右クリック

2　右クリックメニューで[プロパティ]を選択

3　[リンク・オプション]タブの[デバイス]→[ユーザ・オプション・バイト値]でオプション・バイト値を変更

(2)　e^2 studio 使用時

e^2 studio では、下記の手順でオプション・バイト値を変更します。

1　R5F11AGJ_BcnCmb プロジェクトを右クリック

2　右クリックメニューで[Reneas Tool Settings]を選択

3　[ツール設定]タブの[Linker]→[デバイス]→[ユーザ・オプション・バイト値]でオプション・バイト値を変更

6.1.2　RF 送受信動作

ビーコンスタックは RF 動作として送受信ともに有効化、または送信のみ有効化を選択可能です。送信のみ有効化を選択することで RF 初期化時間が短縮されます。サンプルプログラムでは、RF 送受信動作設定を RF_TX_ONLY マクロで設定します。デフォルト設定は 0(送受信とも有効化)です。

ビーコンスタックの送信のみ有効化するには、RF_TX_ONLY マクロ値を 1 に設定します。なお BLE プロトコルスタックは常に送受信が有効となります。

Project_Source¥application¥src¥r_config.h, line 42-44

```
42:    /* RF Operation (0:enable both Tx and Rx, 1:enable Tx only) */
43:    /*   Note: This configuration is only for Beacon Stack      */
44:    #define RF_TX_ONLY              (0)
```

なお RF 送受信動作を変更すると、コネクトアプリケーションの一部が変更となります。変更箇所については 6.2.7 項「RF 送受信動作」を参照してください。

6.1.3　RF 内蔵 DC-DC コンバータ

サンプルプログラムでは、RF 内蔵 DC-DC コンバータの使用設定を RF_DCDC_EN マクロで定義します。RF 内蔵 DC-DC コンバータを使用する、または使用しないを選択可能です。デフォルト設定は 1(RF 内蔵 DC-DC コンバータを使用する)です。

RF 内蔵 DC-DC コンバータの使用を選択するには、RF_DCDC_EN マクロ値を 0 に設定します。

Project_Source¥application¥src¥r_config.h, line 46-47

```
46:    /* RF DC-DC Converter (0:disable DC-DC, 1:enable DC-DC) */
47:    #define RF_DCDC_EN              (1)
```

6.1.4　RF スロー・クロック供給源

RF スロー・クロックは RF 部の時間計測に使用され、RF スロー・クロックの供給源は RF 内蔵オシレータまたは MXU 部の XT1 オシレータのいずれかを選択可能です。サンプルプログラムでは、RF スロー・クロック供給源の選択を RF_SLK_XT1 マクロで定義します。デフォルト設定は 0(RF 内蔵オシレータ)です。

RF スロー・クロック供給源を MCU 部 XT1 オシレータに設定するには、RF_SLK_XT1 マクロ値を 1 に設定します。マクロ値を 1 に設定することで、MCU 部 XT1 オシレータで生成したクロックを EXSLK_RF 端子経由で RF 部に供給します。

Project_Source¥application¥src¥r_config.h, line 49-50

```
49:    /* RF Slow Clock Source (0:RF On-Chip Oscillator, 1:MCU XT1 Oscillator) */
50:    #define RF_SLK_XT1              (0)
```

6.1.5　RF 内蔵オシレータのキャリブレーション

RF 内蔵オシレータを RF スロー・クロック供給源に選択すると、BLE プロトコルスタックは RF 内蔵オシレータが生成するクロック精度のキャリブレーションを常に実行します。一方、ビーコンスタックはキャリブレーションを実行する、または実行しないを選択可能です。サンプルプログラムでは、ビーコンスタック動作中のキャリブレーション実行設定を RF_SLK_CAL マクロで定義します。デフォルト設定は 1(キャリブレーションを実行する)です。

ビーコンスタックは RF 初期化後の初回の Advertising パケット送信後にキャリブレーションを実行します。キャリブレーションの実行により、Advertising インターバルの精度が向上します。

キャリブレーションは実行しないを選択するには、RF_SLK_CAL マクロ値を 0 に設定します。

Project_Source¥application¥src¥r_config.h, line 52-56

```
52:    /* RF Slow Clock Calibration (0:not execute, 1:execute)                    */
53:    /*   Note: This configuration is only for Beacon Stack                     */
54:    /*       : RF Slow Clock Calibration is only for RF-On_Chip_Oscillator     */
55:    /*       : Protocol Stack always execute RF on chip oscillator calibration */
56:    #define RF_SLK_CAL              (1)
```

6.1.6　RF 基準クロックの発振安定時間

サンプルプログラムでは、発振安定時間を RF_32MHZ_WAIT マクロで定義します。RF 基準クロックとして使用する XTAL_RF 回路の発振安定時間は、XTAL1_RF 端子と XTAL2_RF 端子に接続した 32MHz 共振子に依存し、最適な値を設定する必要があります。デフォルト設定は RL78/G1D 評価ボードに適した 1000(usec)です。

発振安定時間を設定するには、RF_32MHZ_WAIT マクロ値を 550 (usec)以上の値に設定します。

Project_Source¥application¥src¥r_config.h, line 58-61

```
58:    /* RF 32MHz Oscillation Stabilization Time (usec, at least 550usec)     */
59:    /*   Note: This configuration is only for Beacon Stack                  */
60:    /*       : Stabilization Time needs to be optimized for 32MHz resonator */
61:    #define RF_32MHZ_WAIT           (1000)
```

RF 基準クロック発生回路の詳細については『RL78/G1D ユーザーズマニュアル ハードウェア編』(R01UH0515)の 15.3.9 項「RF クロック発生回路・ブロック」を参照してください。

6.1.7　最大同時接続数

サンプルプログラムは Peripheral ロールとして動作するため、最大同時接続数は 1 固定です。サンプルプログラムでは、最大同時接続数を MAX_CONNECTION マクロで定義します。デフォルト設定は 1 で、対向の Central ロールデバイス 1 つのみと接続します。

Project_Source¥application¥src¥r_config.h, line 63-66

```
63:    /* Maximum number of Simultaneous Connections (fixed 1)               */
64:    /*   Note: This configuration is only for BLE Protocol Stack          */
65:    /*       : fixed 1, Connect Application behaves as peripheral device  */
66:    #define MAX_CONNECTION          (1)
```

6.1.8　HCI モニタリング

BLE プロトコルスタックはデバッグ用途として、HCI シーケンスをモニタリングする機能を提供します。HCI モニタリングを有効化すると、HCI ログパケットが UART1 から出力され、BLE プロトコルスタック動作のモニタリングが可能です。サンプルプログラムでは、HCI モニタリングの有効化設定を PKTMON_EN マクロで定義します。デフォルト設定は 0(HCI モニタリングを無効)です。

HCI モニタリングを有効化するには、PKTMON_EN マクロ値を 1 に設定します。

Project_Source¥application¥src¥r_config.h, line 68-71

```
68:    /* Packet Monitoring (0:disable Packet Monitor, 1:enable Packet Monitor) */
69:    /*   Note: This configuration is only for BLE Protocol Stack             */
70:    /*       : Packet Monitoring uses UART1 for using output HCI log         */
71:    #define PKTMON_EN               (0)
```

HCI ログパケットを確認するには、PC とソフトウェアが必要になります。

HCI モニタリングの詳細と使用方法については『Bluetooth Low Energy プロトコルスタック ユーザーズマニュアル』(R01UW0095)の 12 章「HCI パケットモニタ機能」を参照してください。

6.1.9　システム動作設定アドレス

コードフラッシュメモリのファームウェア外部に、システム動作設定の配置が可能です。デバイスごとに異なるシステム動作設定を配置することで、デバイスアドレス、Advertising データなどの情報を、ファームウェアをリビルドすることなく設定することが可能です。サンプルプログラムでは、システム動作設定の配置アドレスを SYSCFG_ADDR マクロで定義します。

アドレスマップでの配置アドレスを変更するには、SYSCFG_ADDR マクロ値に新しいアドレスを設定します。

Project_Source¥application¥src¥r_config.h, line 73-86

```
73:    /* System Configuration Address in CodeFlash memory */
74:    #if   defined(_USE_R5F11AGG)
75:        /* System Configuration is located the last block */
76:        #define SYSCFG_ADDR         (0x1FC00)
77:    #elif defined(_USE_R5F11AGH)
78:        /* System Configuration is located the last block */
79:        #define SYSCFG_ADDR         (0x2FC00)
80:    #elif defined(_USE_R5F11AGJ)
81:        /* System Configuration is located the third last block           */
82:        /* by taking into account the location of RL78/G1D module (RY7011) */
83:        #define SYSCFG_ADDR         (0x3F400)
84:        /* In the case of RL78/G1D Module (RY7011), Device Address is located the last block */
85:        #define MODCFG_ADDR         (0x3FC00)
86:    #endif
```

システム動作設定の詳細については 5.4.1 項「コードフラッシュメモリへのアクセス」を参照してください。

6.1.10　評価ボードのスイッチ

サンプルプログラムは、アプリケーションの切り換えのために評価ボードのスイッチを使用します。DIP スイッチ SW6-1 は、サンプルプログラムのビーコン動作と RF 評価動作を切り換えます。スイッチ SW2 は、ビーコン動作時のビーコンアプリケーションとコネクトアプリケーションを切り替えます。サンプルプログラムでは、スイッチの使用設定を EVB_SW マクロで定義します。スイッチを使用する、または使用しないを選択可能です。デフォルト設定は 1(スイッチを使用する)です。

評価ボードのスイッチは使用しないを選択するには、EVB_SW マクロ値を 0 に設定します。

Project_Source¥application¥src¥r_main.c, line 51-61

```
51:    /* Switches on RL78/G1D Evaluation Board (0:not to use, 1:use)                          */
52:    /*   Operation:                                                                         */
53:    /*     When use Switches:                                                               */
54:    /*       - After power up, Beacon Application starts running at first                   */
55:    /*       - It is possible to switch Beacon Application and Connect Application alternately */
56:    /*     When use no Switches:                                                            */
57:    /*       - After power up, Connect Application starts running at first                  */
58:    /*       - If connection is not established within 30sec, Connect Application stops and */
59:    /*         Beacon Application starts running                                            */
60:    /*       - It is not possible to switch from Beacon Application to Connect Application  */
61:    #define EVABOARD_SWITCH_EN      (1)
```

評価ボードのスイッチを使用する場合のアプリケーション切り替え動作を**図 6-1** に示します。

サンプルプログラムは、評価ボードの SW2 と SW6-1 をアプリケーションの切り替えに使用します。SW6-1 が ON ならば、DTM アプリケーションを実行します。SW6-1 が OFF ならば、ビーコンアプリケーションとコネクトアプリケーションを実行します。起動後、まずはビーコンアプリケーションを実行します。SW2 を押下すると、ビーコンアプリケーションとコネクトアプリケーションを交互に切り替えます。

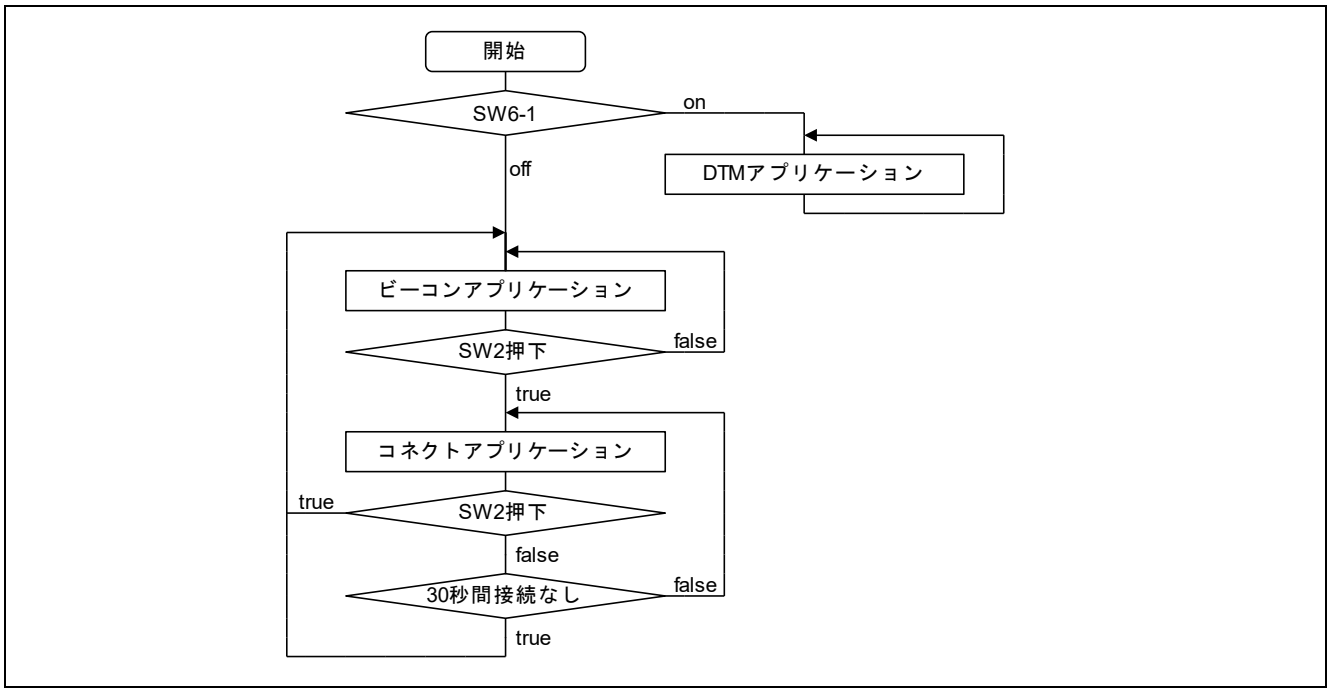


図 6-1　評価ボード使用時のアプリケーション切り換え動作

評価ボードのスイッチを使用しない場合のアプリケーション切り替え動作を**図 6-2** に示します。

サンプルプログラムはビーコンアプリケーションとコネクトアプリケーションを実行します。起動後、まずはコネクトアプリケーションを実行します。30 秒間接続がないならば、ビーコンアプリケーションを実行します。再度コネクトアプリケーションを実行するには、MCU をリセットする必要があります。

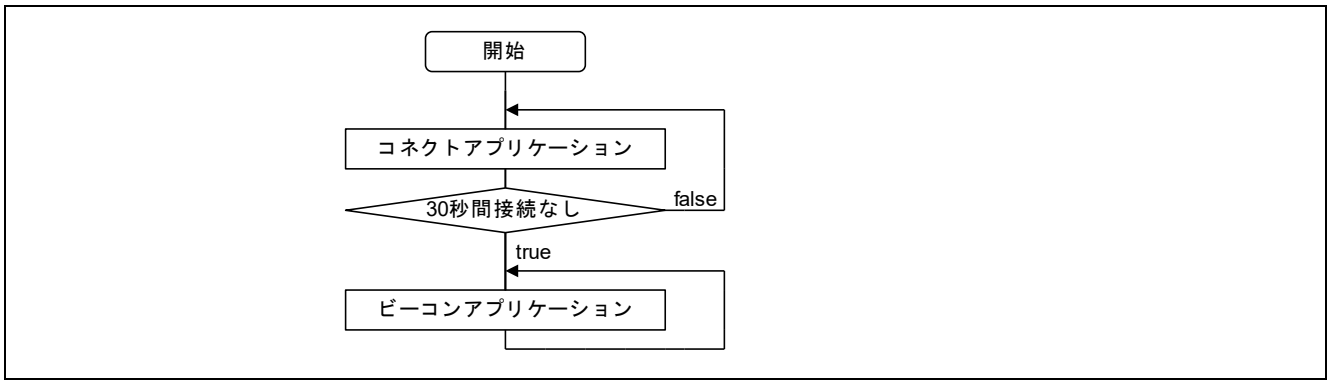


図 6-2　評価ボード未使用時のアプリケーション切り替え動作

6.2　アプリケーション設定

6.2.1　システム動作設定

システム動作設定は、コードフラッシュメモリのファームウェア外部に配置されるデータです。本データは Renesas Flash Programmer のユニークコード埋め込み機能を使用することで、ファームウェアと同時にシステム動作設定を書き込むことが可能です。

リリースパッケージはシステム動作設定のユニークコードファイルをサンプルとして含みます。システム動作設定の詳細については 5.4.1 項「コードフラッシュメモリへのアクセス」を参照してください。

RUC_File¥r5f11agj_syscfg.ruc, line 1-10

```
 1:    // ------------------------------------------------------------------------------------------------
 2:    // -- System Configuration for RL78/G1D BLE Protocol / Beacon Stack Combination Sample Program --
 3:    // --   Device Part Number : R5F11AGJ                                                            --
 4:    // ------------------------------------------------------------------------------------------------
 5:    format hex
 6:    area user flash
 7:    address 0x3f400
 8:    size 156
 9:    index data
          (a)  (b)   (c)                                  (d)
10:    000001 B19A7856341200FF12524C37382F4731442042656 1636F6E2030310000000000000000000000000000000000000000
                                  (d)                                  (e)          (f)
       000000000000000000000000000000000000000000000000000000000000000A00001070009FFFFFFFFFFFFFFFFFFFFFFFF1B02010
                                  (f)                                  (g)
       60303AAFE1316AAFE10EE02676F6F2E676C2F3764694C5478000000001B180952656E6573617320524C37382F4731442042
                 (g)
       6561636F6E303100000000
```

R5F11AGJ ユニークコードファイルには下記の内容が記述されています。

line 1-4 : //から始まる行はコメント行
line 5　: 16 進数フォーマットを指定
line 6　: 書き込みエリアにユーザエリアを指定
line 7　: 書き込みアドレスを 0x3F400 (block 253)に指定
line 8　: 書き込みサイズを 124 byte に指定
line 9　: 次行からのユニークコードデータ開始を宣言
line10　: インデックスとユニークコードを指定
(a): ユニークコードデータのインデックス
(b): デバイスアドレス (6byte)
(c): デバイスアドレスタイプ(1byte), パディング(1byte)
(d): デバイスネーム (66byte)
(e): Advertising 情報 (18byte)
(f): Advertising データ (32byte)
(g): Scan Response データ (32byte)

6.2.2　カーネルヒープメモリ設定

BLE プロトコルスタックはカーネル(RWKE)を含み、カーネルは下記の機能を提供します。BLE プロトコルスタックとコネクトアプリケーションは、カーネル機能を利用して動作します。

- イベント管理機能
- メッセージ通信機能
- タスク状態遷移機能
- タイマ管理機能
- メモリ管理機能

各機能の実行時、カーネルはヒープメモリとしてあらかじめ宣言された領域の一部を動的に確保します。カーネルへの負荷が大きくなった場合、ヒープメモリが枯渇する可能性があります。コネクトアプリケーションの変更や動作拡張などによりヒープメモリが不足する場合は、ヒープメモリサイズを定義する APP_HEAP_SIZE 値を変更してください。

なお RAM の未使用領域はスタックメモリとして使用されます。ヒープメモリサイズを過度に大きくすると、スタックメモリのオーバーフローが発生する可能性があります。

Project_Source¥application¥src¥connect¥resource¥r_kernel.c, line 56-65

```
56:    #define APP_HEAP_SIZE           (0)
57:    /* Note: When Kernel Heap size is not enough, it is necessary to incease APP_HEAP_SIZE */
58:
59:    #define BLE_HEAP_SIZE           ((MAX_CONNECTION * 256) + 512           ¥
60:                                      + BLE_HEAP_CONT                       ¥
61:                                      + (BLE_HEAP_HOST * MAX_CONNECTION)    ¥
62:                                      + BLE_DB_SIZE                         ¥
63:                                      + RBLE_TABLE_SIZE                     ¥
64:                                      + APP_HEAP_SIZE                       ¥
65:                                    )
```

カーネルが提供する機能の詳細については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)の 9 章「RWKE」を参照してください。

6.2.3　Advertising 設定

ビーコンアプリケーションのデフォルト Advertising 設定は r_beacon.c ファイルで定義します。

デフォルトの Advertising 情報や Advertising データを変更するには、RBLE_ADV_INFO 構造体や RBLE_ADV_DATA 構造体の値を変更します。構造体の仕様については『RL78/G1D ビーコンスタック ユーザーズマニュアル』(R01UW0171)の 4 章「API」を参照してください。

Project_Source¥application¥src¥beacon¥r_beacon.c, line 42-92

```
42:    /* Advertising Data Array */
43:    static RBLE_ADV_DATA adv_data[] =
44:    {
45:        /* Advertising Data[0] */
46:        /* Eddystone-URL: https://goo.gl/5wKkRK -> https://www.renesas.com/ */
47:        {
48:            /* Advertising data length */
49:            27,
50:            /* Advertising data <<Flags>> */
51:            0x02, 0x01, 0x06,
52:            /* Advertising data <<Complete List of 16-bit Service Class UUIDs>> */
53:            0x03, 0x03, 0xAA, 0xFE,
54:            /* Advertising data <<Service Data>> */
55:            0x13, 0x16, 0xAA, 0xFE, 0x10, 0xEE, 0x02,
56:            'g', 'o', 'o', '.', 'g', 'l', '/', '5', 'w', 'K', 'k', 'R', 'K'
57:        },
58:        #if !RF_TX_ONLY
59:        /* Scan Response Data[0] */
60:        {
61:            /* Scan Response data length */
62:            25,
63:            /* Scan Response data <<Complete local name>> */
64:            0x18, 0x09,
65:            'R','e','n','e','s','a','s',' ','R','L','7','8','/','G','1','D',
66:            ' ','B','e','a','c','o','n'
67:        },
68:        #endif
69:    };
70:
71:    /* Advertising packet type */
72:    #if RF_TX_ONLY
73:    static const uint8_t adv_type = RBLE_PDU_ADV_NONCONN_IND;
74:    #else
75:    static const uint8_t adv_type = RBLE_PDU_ADV_SCAN_IND;
76:    #endif
77:
78:    /* Advertising Information */
79:    static RBLE_ADV_INFO adv_info =
80:    {
81:        0x00A0,                                     /* Advertising Interval         */
82:        true,                                       /* Advertising Interval Delay   */
83:        RBLE_ADV_ALL_CHANNELS,                      /* Advertising Channel Map      */
84:        0x00,                                       /* Advertising Transfer Count   */
85:        RBLE_TXPW_LV9,                              /* Advertising Transfer Power   */
86:        { 0xB0, 0x9A, 0x78, 0x56, 0x34, 0x12 },     /* Own Device Address           */
87:        RBLE_ADDR_PUBLIC,                           /* Own Device Address Type      */
88:        sizeof(adv_data) / sizeof(RBLE_ADV_DATA),   /* Advertising Data Count       */
89:        &adv_data[0],                               /* Advertising Data             */
90:        RBLE_EVT_PERMIT_ADV_ALL,                    /* Advertising Event Permission */
91:        false                                       /* Use White List               */
92:    };
```

複数 Advertising データを繰り返し送信するには、RBLE_ADV_DATA 構造体配列を追加します。

複数 Advertising データ送信のコード例

```
/* Advertising Data Array */
static RBLE_ADV_DATA adv_data[] =
{
    /* Advertising data No.1 */
    {
        /* Advertising data length */
        … ,
        /* Advertising data */
        …
    },
    /* Advertising data No.2 */
    {
        /* Advertising data length */
        … ,
        /* Advertising data */
        …
    }
    :
};
```

コネクトアプリケーションのデフォルト Advertising 設定は r_connect.c ファイルで定義します。

デフォルト Advertising 設定を変更するには、RBLE_ADV_INFO 構造体のデフォルト Advertising 情報、デフォルト Advertising データ、デフォルト Scan Response データの値を変更します。 構造体の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)の 5 章「Generic Access Profile」を参照してください。

Project_Source¥application¥src¥connect¥r_connect.c, line 148-181

```
148:    /* Advertising Information for connection as a slave role */
149:    /*  Note : it is necessary to change configuration corresponds to each use case */
150:    static RBLE_ADV_INFO  broadcast_info =
151:    {
152:        /* Advertising Parameter structure */
153:        {
154:            0x0030,                                     /* Advertising Interval Min        */
155:            0x0030,                                     /* Advertising Interval Max        */
156:            RBLE_GAP_ADV_CONN_UNDIR,                    /* Advertising Type                */
157:            RBLE_ADDR_PUBLIC,                           /* Own Address Type                */
158:            0x00,                                       /* Direct Advertising Address Type */
159:            { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },     /* Direct Advertising Address      */
160:            RBLE_ADV_ALL_CHANNELS,                      /* Advertising Channel Map         */
161:            RBLE_ADV_ALLOW_SCAN_ANY_CON_ANY,            /* Advertising Filter Policy       */
162:            0x00,                                       /* (reserved)                      */
163:        },
164:        /* Advertising Data structure */
165:        {
166:            /* Advertising data length (max 31byte)  */
167:            3+25,
168:            /* Advertising data <<Flags>> */
169:            2,  0x01, 0x06,
170:            /* Advertising data <<Complete Local Name>> */
171:            24, 0x09,
172:            'R','L','7','8','/','G','1','D',' ','B','e','a','c','o','n',' ','U','p','d','a','t','e','r'
173:        },
174:        /* Scan Response Data structure */
175:        {
176:            /* Scan Response data length (max 31byte) */
177:            0,
178:            /* Scan Response data */
179:            0x00
180:        }
181:    };
```

特定の対向デバイスと接続するには、White List を有効化するために RBLE_ADV_INFO 構造体の Advertising フィルタポリシーに RBLE_ADV_ALLOW_SCAN_ANY_CON_WLST を設定します。

また Advertising の開始前に、RBLE_GAP_Add_To_White_List をコールして White List にデバイスアドレスを追加します。複数のデバイスアドレスを追加するには、デバイスアドレスごとに RBLE_GAP_Add_To_White_List をコールします。

White List へのデバイスアドレス追加のコード例

```
uint9_t wl_cnt;
/* device address list for white list */
static RBLE_DEV_ADDR_INFO wl_info[] =
{
    {RBLE_ADDR_PUBLIC, { 0x01, 0x90, 0x78, 0x56, 0x34, 0x12 }},
    :
};

/* set device address to white list */
/* it is necessary to call repeatedly until all device address of list is added */
if(wl_cnt < (sizeof(wl_info) / sizeof(RBLE_DEV_ADDR_INFO)))
{
    RBLE_GAP_Add_To_White_List(&wl_info[wl_cnt++]);
}
```

※Resolvable Private address は Identity Resolving Key により定期的に変更されるため、Resolvable Private address は White List に設定すべきではありません。

6.2.4　未接続タイムアウト時間設定

コネクトアプリケーションはアプリケーション起動または直前の接続切断から、未接続タイムアウト時間として定義された時間内に接続がないならば、アプリケーションを終了します。未接続タイムアウト時間は CON_EXIT_TIME マクロで定義され、タイムアウト時間は 10～299,990msec の範囲を 10msec 単位で設定可能です。未接続タイムアウト時間のデフォルト設定は 30 秒です。

未接続タイムアウト時間を変更するには、CON_EXIT_TIME マクロ値を 1～29,999 の範囲で変更します。

Project_Source¥application¥src¥connect¥r_connect.c, line 48-50

```
48:    /* Connect Application Exit Timeout Time (unit: 10msec) */
49:    /* When connection is not established within this time, Connect Application exits. */
50:    #define CON_EXIT_TIME       (3000)
```

タイムアウトの監視には BLE プロトコルスタックのカーネルタイマを使用します。カーネルタイマの仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)の 9.5 節「タイマ管理機能」を参照してください。

6.2.5　Pairing 設定

デフォルトの Pairing 設定は r_connect.c ファイルで定義します。サンプルプログラムは Pairing 方法として Just Works を実行します。

Project_Source¥application¥src¥connect¥r_connect.c, line 183-196

```
183:    /* Pairing Information for secure connection */
184:    /*  Note : it is necessary to change configuration corresponds to each secure level requested */
185:    static RBLE_BOND_RESP_PARAM bond_info =
186:    {
187:        0x0000,                                     /* Connection handle           */
188:        RBLE_OK,                                    /* accept or reject bonding    */
189:        RBLE_IO_CAP_NO_INPUT_NO_OUTPUT,             /* IO capabilities             */
190:        RBLE_OOB_AUTH_DATA_NOT_PRESENT,             /* OOB flag                    */
191:        RBLE_AUTH_REQ_NO_MITM_BOND,                 /* Authentication Requirements */
192:        RBLE_SMP_MAX_ENC_SIZE_LEN,                  /* Encryption key size         */
193:        RBLE_KEY_DIST_NONE,                         /* Initiator key distribution  */
194:        RBLE_KEY_DIST_ENCKEY,                       /* Responder key distribution  */
195:        0x00                                        /* Reserved                    */
196:    };
```

Pairing 方法を Just Works から Passkey Entry に変更し、アプリケーションから Passkey を提供するには、RBLE_BOND_RESP_PARAM 構造体の IO Capabilities に RBLE_IO_CAP_DISPLAY_ONLY を設定し、Authentication Requirements に RBLE_AUTH_REQ_MITM_BOND を設定します。

Passkey Entry による Pairing シーケンスの実行時、RBLE_SM_TK_REQ_IND イベントが通知されます。アプリケーションは Passkey を Temporary Key として応答するために、RBLE_SM_Tk_Req_Resp をコールする必要があります。下記のサンプルコードは、標準ライブラリの rand 関数で Passkey を生成します。Passkey の生成後、LCD などの表示によりユーザに Passkey を表示する必要があります。

Temporary Key 応答のコード例

```
    uint32_t passkey;
    uint8_t* byteptr = (uint8_t*)&passkey;
    uint8_t  idx;

    /* TK(Temporary Key) buffer */
    RBLE_KEY_VALUE tk =
    {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    /* generate Passkey (range:000,000 - 999,999) */
    passkey  = (uint32_t)rand();
    passkey |= (uint32_t)rand() << 16;
    passkey %= 1000000;

    /* copy Passkey to TK(Temporary Key) buffer */
    for(idx = 0; idx < sizeof(uint32_t); idx++)
    {
        tk.key[RBLE_KEY_LEN - 1 - idx] = byteptr[idx];
    }
    RBLE_SM_Tk_Req_Resp(con_idx, RBLE_OK, &tk);
```

6.2.6 Custom Profile

Custom Profile を実装および変更するには、下記の定義、リソース、処理を実装する必要があります。

- 定義:
  - Custom Profile の Service, Characteristic を識別するための UUID
  - Custom Profile の Service, Characteristic を BLE プロトコルスタックが操作するための Attribute index
  - Custom Profile の Service, Characteristic を Client デバイスに通知するための Attribute handle

- リソース:
  - Custom Profile の Characteristic 値を格納するための変数
  - Custom Profile の Service, Characteristic の特徴を Attribute データベースに設定するための記述子
  - Custom Profile の Characteristic 変数を Attribute データベースに設定するための記述子
  - BLE プロトコルスタックがアクセスするための Attribute データベース

- 処理:
  - 接続確立後の GATT の有効化とイベントコールバック関数の登録
  - Client デバイスからの Write Request による Characteristic 値の更新と、Write Response の送信

※1: Client デバイスからの Read Request に対する Characteristic 値の Read Response の送信を実装する必要はありません。Read Response の送信は、BLE プロトコルスタックが自動で実行します。

※2: Server デバイスが能動的に Characteristic 値を通知する必要があるならば、Notification または Indication を送信する処理が必要となります。Notification または Indication を送信するには、BLE プロトコルスタックの RBLE_GATT_Notify_Request または RBLE_GATT_Indicate_Request を使用します。

上記関数の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)の 7.2.9 項「RBLE_GATT_Notify_Request」と 7.2.10 項「RBLE_GATT_Indicate_Request」を参照してください。

(1) 定義

UUID は r_profile.h ファイルで定義します。

乱数による UUID は、UUIDGEN Linux コマンドで生成が可能です。UUID の生成後、UUID 値は LSB 順に設定する必要があります。UUID 値は各 Service、各 Characteristic の記述子に設定されます。

Project_Source¥application¥src¥connect¥r_profile.h, line 45-53

```
45:    /* Custom Profile 128bit UUID: A766xxxx-4B1E-4d6e-91C4-997BA9B6FC07 */
46:    /*  Note: randomly numbers UUID can be generated by UUIDGEN linux command */
47:    /*        regarding the specification of UUID, refer to ITU-T X.667       */
48:    #define PRF_UUID_SERVICE
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x01,0x00,0x66,0xA7}
49:    #define PRF_UUID_CHAR_BCNINFO
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x02,0x00,0x66,0xA7}
50:    #define PRF_UUID_CHAR_BCNDATA
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x03,0x00,0x66,0xA7}
51:    #define PRF_UUID_CHAR_CFLCNT
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x04,0x00,0x66,0xA7}
52:    #define PRF_UUID_CHAR_DFLCNT
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x05,0x00,0x66,0xA7}
53:    #define PRF_UUID_CHAR_RSPDATA
                        {0x07,0xFC,0xB6,0xA9,0x7B,0x99,0xC4,0x91,0x6e,0x4d,0x1E,0x4B,0x06,0x00,0x66,0xA7}
```

Attribute indexes と Attribute handles は r_gatt.h ファイルで定義します。Custom Profile の Service、Characteristic は追加や削除が可能です。

Attribute index 値は Attribute データベースに直接設定され、Attribute handle 値は各 Service、各 Characteristic の記述子に設定されます。

Project_Source¥application¥src¥connect¥resource¥r_gatt.h, line 37-113

```
 37:    /* Attribute index */
 38:    enum
 39:    {
 60:        /* offset index for Custom Profiles */
 61:        ATT_IDX_CUSTOM = 0x0200,
 :
 65:        /* Custom Profile Service */
 66:        PRF_IDX_SVC,
 67:        PRF_IDX_BCNINFO_CHAR,
 68:        PRF_IDX_BCNINFO_VAL,
 69:        PRF_IDX_BCNDATA_CHAR,
 70:        PRF_IDX_BCNDATA_VAL,
 71:        PRF_IDX_RSPDATA_CHAR,
 72:        PRF_IDX_RSPDATA_VAL,
 73:        PRF_IDX_CFLCNT_CHAR,
 74:        PRF_IDX_CFLCNT_VAL,
 75:        PRF_IDX_DFLCNT_CHAR,
 76:        PRF_IDX_DFLCNT_VAL,
 79:    };
 80:
 81:    /* Attribute handles */
 82:    enum
 83:    {
 99:        /* Custom Profile Service */
100:        PRF_HDL_SVC                   = 0x000C,
101:        PRF_HDL_BCNINFO_CHAR          = 0x000D,
102:        PRF_HDL_BCNINFO_VAL           = 0x000E,
103:        PRF_HDL_BCNDATA_CHAR          = 0x000F,
104:        PRF_HDL_BCNDATA_VAL           = 0x0010,
105:        PRF_HDL_RSPDATA_CHAR          = 0x0011,
106:        PRF_HDL_RSPDATA_VAL           = 0x0012,
107:        PRF_HDL_CFLCNT_CHAR           = 0x0013,
108:        PRF_HDL_CFLCNT_VAL            = 0x0014,
109:        PRF_HDL_DFLCNT_CHAR           = 0x0015,
110:        PRF_HDL_DFLCNT_VAL            = 0x0016,
113:    };
```

(2)　リソース

Characteristic 値を格納するための変数は r_profile.c ファイルで定義します。

Characteristic の初期値はアプリケーションによって設定されます。その後、書き込み許可を持つ Characteristic は Client デバイスにより更新されます。Characteristic 値変数は各 Characteristic の記述子に設定されます。

Project_Source¥application¥src¥connect¥r_profile.c, line 64-69

```
64:    /* Custom Profile characteristic variables */
65:    PRF_ADV_INFO    prf_bcninfo_val;                    /* Advertising   Information */
66:    PRF_ADV_DATA    prf_bcndata_val;                    /* Advertising   Data        */
67:    PRF_ADV_DATA    prf_rspdata_val;                    /* Scan Response Data        */
68:    uint16_t        prf_cflcnt_val;                     /* Code Flash Updated Count  */
69:    uint16_t        prf_dflcnt_val;                     /* Data Flash Updated Count  */
```

Custom Profile の Service、Characteristic の記述子は r_gatt.c ファイルで定義します。

UUID、Attribute Handle、Attribute Permissions、Characteristic 値変数のような特徴を Attribute データベースに設定するために、記述子を定義する必要があります。

Project_Source¥application¥src¥connect¥resource¥r_gatt.c, line 103-179

```
103:    /* Custom Service */
104:    static const uint8_t custom_svc[RBLE_GATT_128BIT_UUID_OCTET] = PRF_UUID_SERVICE;
105:
106:    /* Advertising Information */
107:    static const struct atts_char128_desc prf_bcninfo_char =
108:    {
 :          …
114:    };
115:
116:    struct atts_elmt_128 prf_bcninfo_elmt =
117:    {
 :          …
119:    };
120:
121:    /* Advertising Data */
122:    static const struct atts_char128_desc prf_bcndata_char =
123:    {
 :          …
129:    };
130:
131:    struct atts_elmt_128 prf_bcndata_elmt =
132:    {
 :          …
134:    };
135:
136:    /* Scan Response Data */
137:    static const struct atts_char128_desc prf_rspdata_char =
138:    {
 :          …
144:    };
145:
146:    struct atts_elmt_128 prf_rspdata_elmt =
147:    {
 :          …
149:    };
150:
151:    /* Code Flash Updated Count */
152:    static const struct atts_char128_desc prf_cflcnt_char =
153:    {
 :          …
159:    };
160:
161:    struct atts_elmt_128 prf_cflcnt_elmt =
162:    {
 :          …
164:    };
165:
166:    /* Data Flash Updated Count */
167:    static const struct atts_char128_desc prf_dflcnt_char =
168:    {
 :          …
174:    };
175:
176:    struct atts_elmt_128 prf_dflcnt_elmt =
177:    {
 :          …
179:    };
```

Attribute データベースは r_gatt.c ファイルで定義します。

Attribute データベースは Service、Characteristic を BLE プロトコルスタックに設定するために必要となります。Attribute データベースは Service 記述子と Characteristic 記述子で構成されます。

Project_Source¥application¥src¥connect¥resource¥r_gatt.c, line 213-243

```
213:    /* Attribute Database */
214:    const struct atts_desc atts_desc_list_prf[] =
215:    {
216:        /**********************/
217:        /* Custom Service      */
218:        /**********************/
219:        {RBLE_DECL_PRIMARY_SERVICE,  sizeof(custom_svc),       sizeof(custom_svc),              … },
220:        /* Advertising Information */
221:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_bcninfo_char), sizeof(prf_bcninfo_char),        … },
222:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_INFO),     sizeof(PRF_ADV_INFO),            … },
223:        /* Advertising Data */
224:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_bcndata_char), sizeof(prf_bcndata_char),        … },
225:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),            … },
226:        /* Scan Response Data */
227:        #if RF_TX_ONLY
228:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),        … },
229:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),            … },
230:        #else
231:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_rspdata_char), sizeof(prf_rspdata_char),        … },
232:        { DB_TYPE_128BIT_UUID,       sizeof(PRF_ADV_DATA),     sizeof(PRF_ADV_DATA),            … },
233:        #endif
234:        /* Code Flash Updated Count */
235:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_cflcnt_char),  sizeof(prf_cflcnt_char),         … },
236:        { DB_TYPE_128BIT_UUID,       sizeof(uint16_t),         sizeof(uint16_t),                … },
237:        /* Data Flash Updated Count */
238:        { RBLE_DECL_CHARACTERISTIC,  sizeof(prf_dflcnt_char),  sizeof(prf_dflcnt_char),         … },
239:        { DB_TYPE_128BIT_UUID,       sizeof(uint16_t),         sizeof(uint16_t),                … },
240:
241:        /* zero terminator */
242:        {0,0,0,0,0,0}
243:    };
```

(3) Processing

GATT の有効化処理と Characteristic 値の更新処理は r_profile.c ファイルで定義します。

GATT の有効化と GATT イベントコールバック関数を登録するために、RBLE_GATT_Enable をコールします。Client デバイスからの Write Request により Characteristic 値の更新が要求されると、RBLE_GATT_EVENT_WRITE_CMD_IND イベントが通知されます。アプリケーションは Characteristic 値を更新し、Write Response を送信するために RBLE_GATT_Write_Response をコールします。

Project_Source¥application¥src¥connect¥r_profile.c, line 92-341

```
 92:    RBLE_STATUS PRF_Server_Enable(uint16_t conhdl, PRF_EVT_HANDLER callback)
 93:    {
101:            result = RBLE_GATT_Enable(prf_gatt_callback);
113:    }
 :
215:    static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216:    {
224:        switch(evt->type)
225:        {
226:            case RBLE_GATT_EVENT_WRITE_CMD_IND:
227:                /* reach here when client device requests to write characteristic */
235:                switch(att_hdl)
236:                {
237:                    /* Advertising information (18byte fixed) is written */
238:                    case PRF_HDL_BCNINFO_VAL:
 :
243:                                /* update characteristic value */
 :
256:                        break;
257:
258:                    /* Advertising data (2byte - 32byte variable) is written */
259:                    /* Note: when requested size is over than the size of single write request, */
260:                    /*       characteristic value is transferred separately per 18byte          */
261:                    case PRF_HDL_BCNDATA_VAL:
 :
268:                                    /* update characteristic value */
 :
277:                        break;
304:                }
305:
306:                /* send the write response to client device */
307:                if(evt->param.write_cmd_ind.resp)
308:                {
309:                    prf_send_wr_resp(att_hdl, result);
310:                }
311:                break;
315:        }
316:    }
317:
 :
325:    static void prf_send_wr_resp(uint16_t att_hdl, RBLE_STATUS result)
326:    {
340:        RBLE_GATT_Write_Response(&wr_resp);
341:    }
```

6.2.7　RF 送受信動作

ビーコンスタックの RF 送受信動作を変更すると、アプリケーションの動作が変更となります。ビーコンアプリケーションとコネクトアプリケーションの変更点を**表 6-2** に示します。

RF 送受信動作の詳細については 6.1.2 項「RF 送受信動作 」を参照してください。

表 6-2　RF 送受信動作設定のアプリケーション差分

	RF 送受信動作時(RF_TX_ONLY=0)	RF 送信動作時(RF_TX_ONLY=1)
ビーコンアプリケーション		
Advertising Type (r_beacon.c)	Scannable Undirected Advertising	Non-connectable Undirected Advertising
コネクトアプリケーション		
Custom Profile (r_gatt.c) (r_profile.c) (r_connect.c)	Characteristic 構成 − Advertising Information − Advertising Data − Scan Response Data − Code Flash Memory Updated Count − Data Flash Memory Updated Count	Characteristic 構成 − Advertising Information − Advertising Data − Code Flash Memory Updated Count − Data Flash Memory Updated Count ※Scan Response Data Characteristic へのアクセス不可

iOS デバイスは、接続した Bluetooth low energy デバイスの Service 構成と Characteristic 構成を OS がキャッシュします。RF 送受信動作を変更すると、コネクトアプリケーションが保持する Resasas Bluetooth Updater Service の Characteristic 構成と、キャッシュされた Characteristic 構成に差異が発生します。

RF 送受信動作を変更した場合、iOS デバイスがキャッシュした Custom Profile 構成をクリアするため、iOS デバイスの設定画面で Bluetooth の無効化と再有効化を行ってください。

7. 関数

サンプルプログラムに実装された主要な関数を示します。

7.1 関数一覧

7.1.1 アプリケーション切り換え処理

表 7-1 にアプリケーション切り換え処理の関数を示します。

表 7-1 アプリケーション切り換え処理関数

ファイル	関数	説明
r_main.c	main	MCU の初期化とアプリケーションの実行
	input_callback	アプリケーション終了関数の実行
r_input.c	R_INPUT_Init	外部入力割り込みの初期化
	intp5_interrupt	外部入力割り込みハンドラ
r_plf.c	R_PLF_Init	MCU (ポート、クロック)の初期化

7.1.2 ビーコンアプリケーション

表 7-2 にビーコンアプリケーションの関数を示します。

表 7-2 ビーコンアプリケーション関数

ファイル	関数	説明
r_beacon_main.c	R_BEACON_Main	ビーコンアプリケーションのメインループ
r_beacon.c	R_BEACON_Start	ビーコンアプリケーションの開始
	R_BEACON_Exit	ビーコンアプリケーションの終了
	R_BEACON_EventHandler	イベントハンドラ

7.1.3 コネクトアプリケーション

表 7-3 にコネクトアプリケーションの関数を示します。

表 7-3 コネクトアプリケーション関数

ファイル	関数	説明
r_connect_main.c	R_CONNECT_Main	コネクトアプリケーションのメインループ
r_connect.c	R_CONNECT_Start	コネクトアプリケーションの開始
	R_CONNECT_Exit	コネクトアプリケーションの終了
	con_rble_callback	イベントコールバック関数：RBLE
	con_gap_callback	イベントコールバック関数：Generic Access Profile
	con_sm_callback	イベントコールバック関数：Security Manager
	con_profile_callback	イベントコールバック関数：Custom Profile
	con_vs_callback	イベントコールバック関数：Vendor Specific
	con_exit_timer_task	30 秒間接続がない場合のアプリケーション終了

7.1.4 DTM アプリケーション

表 7-4 に DTM アプリケーションの関数を示します。

表 7-4 DTM アプリケーション関数

ファイル	関数	説明
r_dtm_main.c	R_DTM_Main	DTM アプリケーションのメインループ
	R_DTM_Start	DTM アプリケーションの開始
r_dtm.c	dtm_rble_callback	イベントコールバック関数：RBLE
	dtm_gap_callback	イベントコールバック関数：Generic Access Profile
	dtm_sm_callback	イベントコールバック関数：Security Manager
	dtm_vs_callback	イベントコールバック関数：Vendor Specific

7.2　関数コール

7.2.1　ビーコン動作時の関数コール

ビーコン動作時の関数コールグラフを**図 7-1** に示します。評価ボードの DIP スイッチ SW6-1 が OFF ならば、main 関数は R_BEACON_Main 関数と R_CONNECT_Main 関数を交互にコールします。

R_BEACON_Main 関数は、R_BEACON_Start 関数をコールしてビーコンアプリケーション を開始し、メインループを実行します。その後、メインループは R_BEACON_Break 関数をコールすることで終了します。

R_CONNECT_Main 関数は、R_CONNECT_Start 関数をコールしてコネクトアプリケーションを開始し、メインループを実行します。その後、メインループは R_CONNECT_Break 関数をコールすることで終了します。

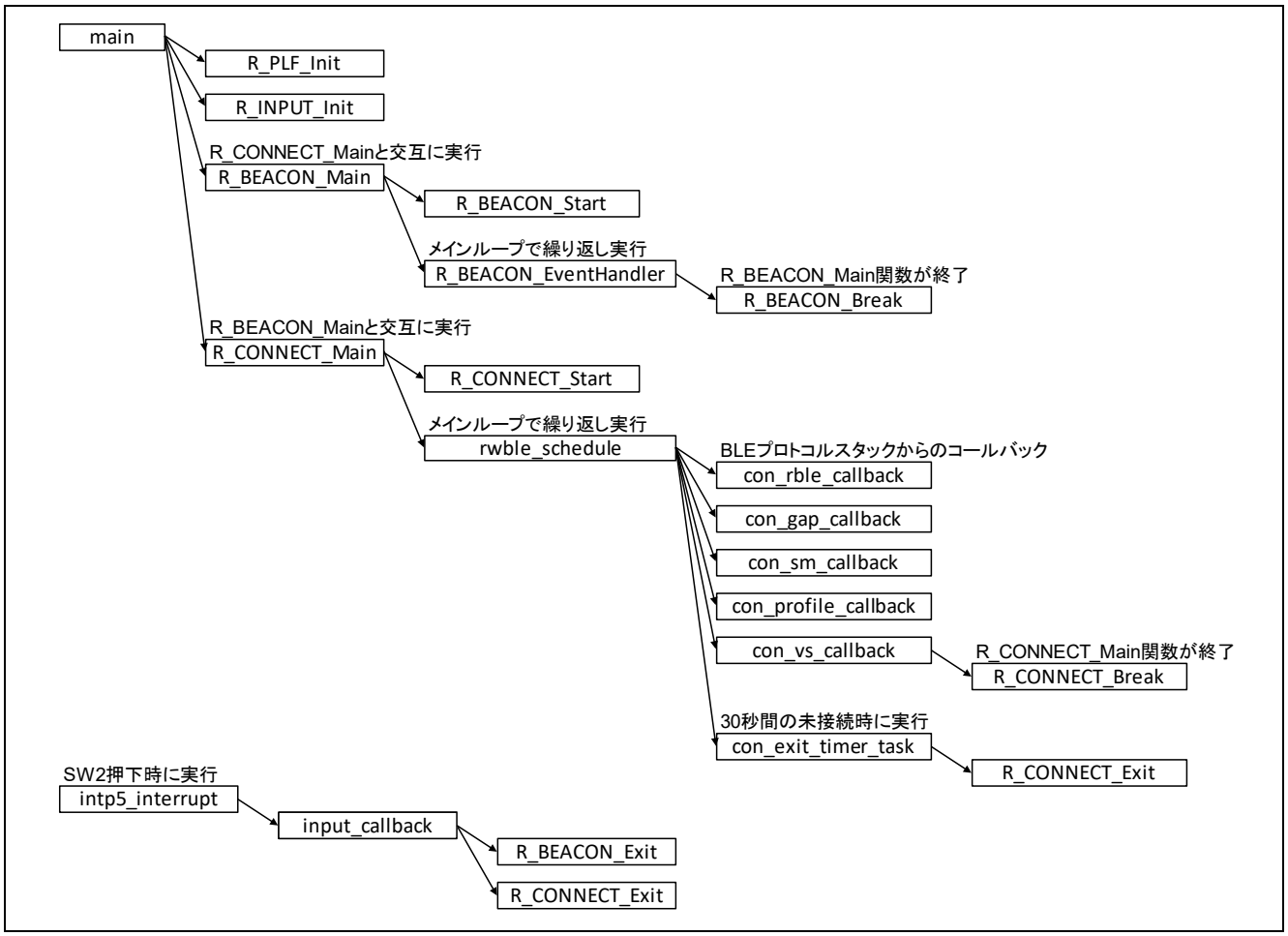


図 7-1　ビーコン動作時の関数コールグラフ

7.2.2　RF 評価動作時の関数コール

RF 評価動作時の関数コールグラフを**図 7-2** に示します。評価ボードの DIP スイッチ SW6-1 が ON ならば、main 関数は R_DTM_Main 関数をコールします。R_DTM_Main 関数は R_DTM_Start 関数をコールして DTM アプリケーションを開始し、メインループを実行します。DTM アプリケーションのメインループは終了しません。

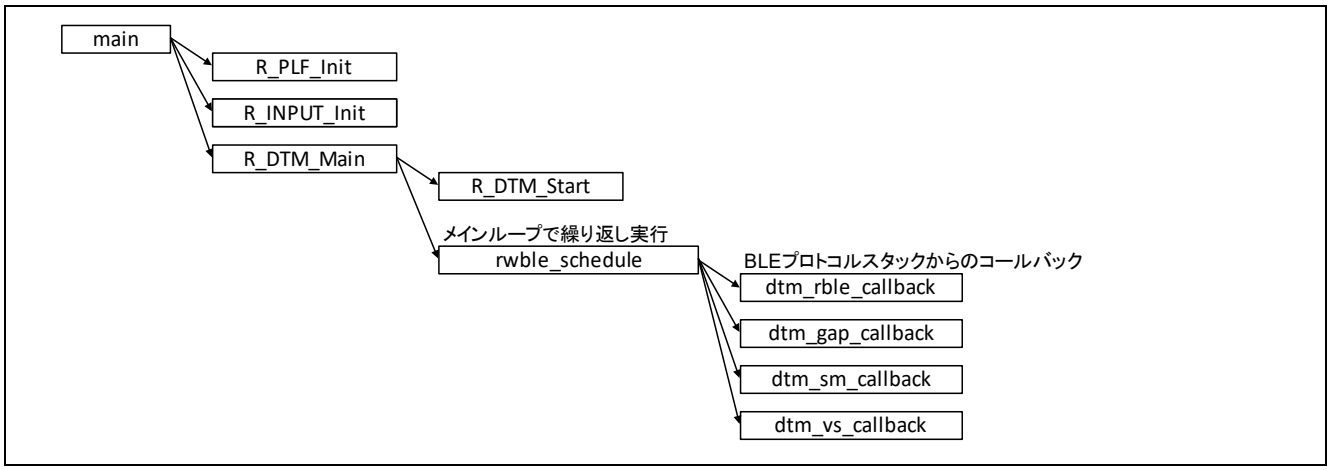


図 7-2　RF 評価動作時の関数コールグラフ

8. 動作

8.1 状態遷移

サンプルプログラムは、3 つのアプリケーション（ビーコンアプリケーション、コネクトアプリケーション、DTM アプリケーション）で構成されます。本節では各アプリケーションの状態遷移を示します。

8.1.1 ビーコンアプリケーション

ビーコンアプリケーションの状態遷移を**図 8-1** に示します。

ビーコンアプリケーションは Initializing 状態で動作を開始し、その後 Advertising 状態に遷移します。Advertising 状態では、ビーコンアプリケーションは Advertising を実行します。アプリケーションの終了要求があると、RF Powerdown 状態に遷移し、その後ビーコンアプリケーションは終了します。

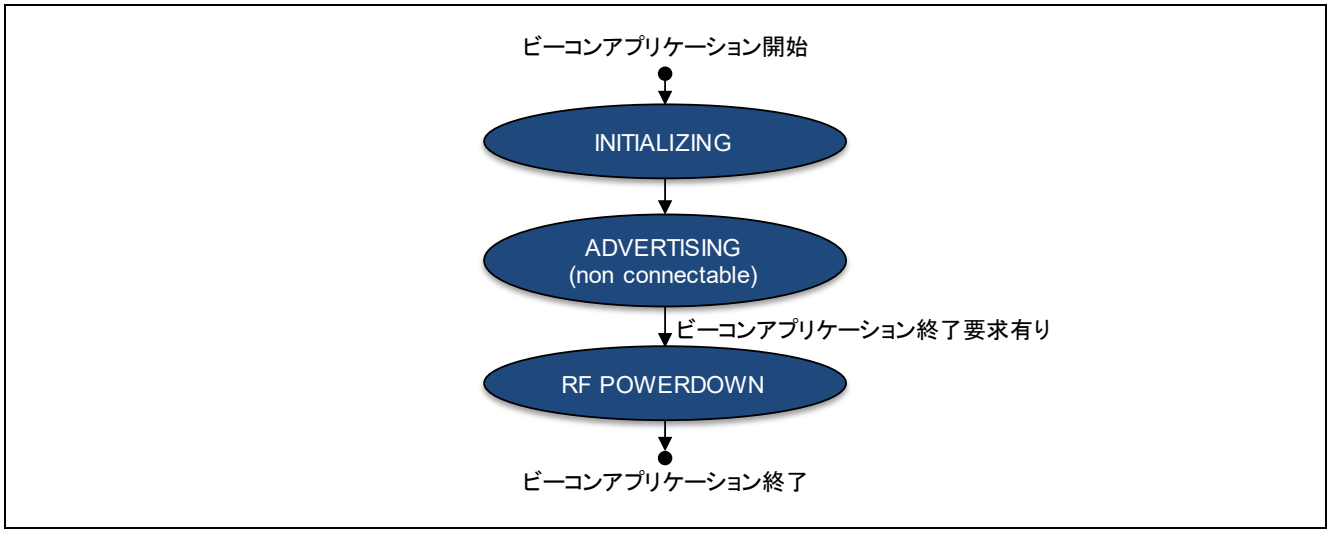


図 8-1　ビーコンアプリケーションの状態遷移

8.1.2　コネクトアプリケーション

コネクトアプリケーションの状態遷移を**図 8-2** に示します。

コネクトアプリケーションは Initializing 状態で動作を開始し、その後 Advertising 状態に遷移します。

対向デバイスからの Connection Request により、Slave Connection 状態に遷移します。Slave Connection 状態では、コネクトアプリケーションは GATT の有効化と Security 状態を確認します。接続済み対向デバイスからの Pairing Request または Start Encryption Request により、コネクトアプリケーションは Pairing または Start Encryption を実行します。また本状態では、GATT アクセスを実行します。対向デバイスとの接続が切断されると、Advertising 状態に戻ります。

Advertising 状態では、アプリケーションの終了要求または 30 秒間接続がないならば、コネクトアプリケーションは Advertising を停止後、RF Powerdown 状態に遷移し、その後コネクトアプリケーションは終了します。

Slave Connection 状態では、アプリケーションの終了要求があると、接続済み対向デバイスに接続の切断を要求後、RF Powerdown 状態に遷移し、その後コネクトアプリケーションは終了します。

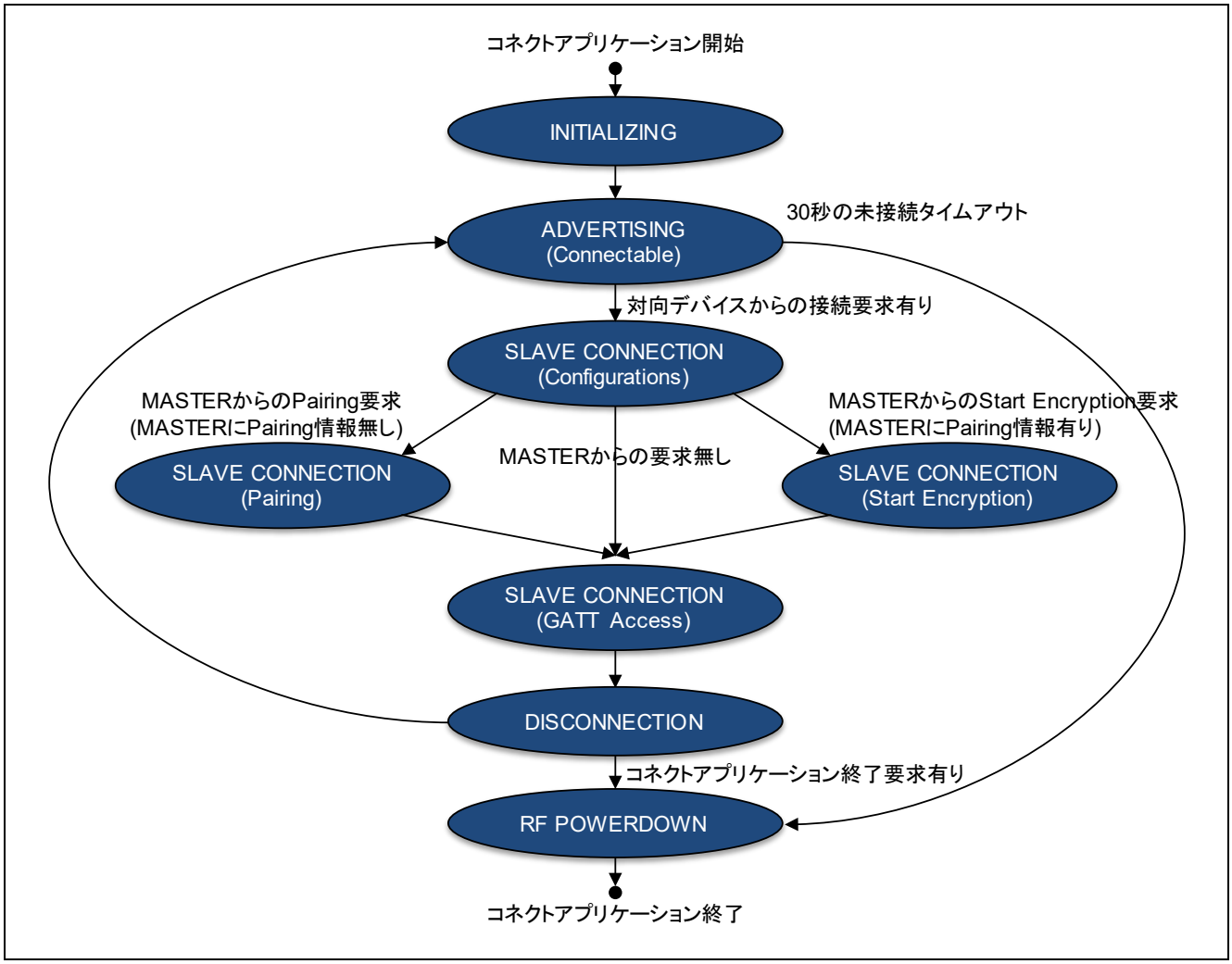


図 8-2　コネクトアプリケーションの状態遷移

8.1.3　DTM アプリケーション

DTM アプリケーションの状態遷移を**図 8-3** に示します。

DTM アプリケーションは Initializing 状態で動作を開始し、その後 Idling 状態に遷移します。RF テスタからの要求により RF Transmitter Test または RF Receiver Test を実行し、テスト完了後は Idling 状態に戻ります。

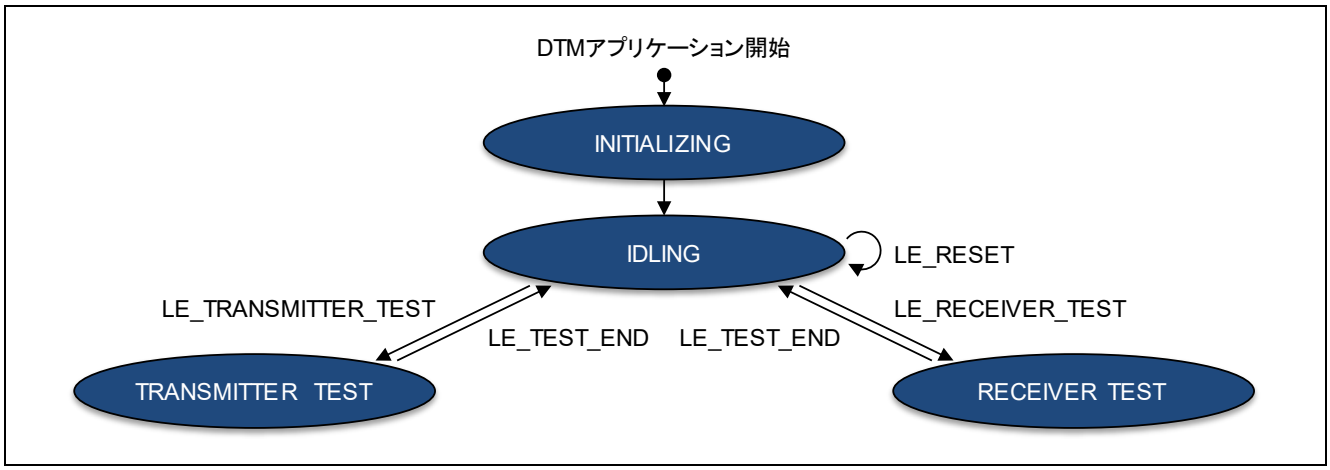


図 8-3　DTM アプリケーションの状態遷移

8.2　シーケンス

8.2.1　ビーコンアプリケーション

(1)　Initializing & Advertising & RF Powerdown シーケンス

ビーコンアプリケーションの Initializing 状態、Advertising 状態、RF Powerdown 状態のシーケンスを**図 8-4** に示します。ビーコンアプリケーションとビーコンスタックでは、ビーコンスタック API が使用されます。

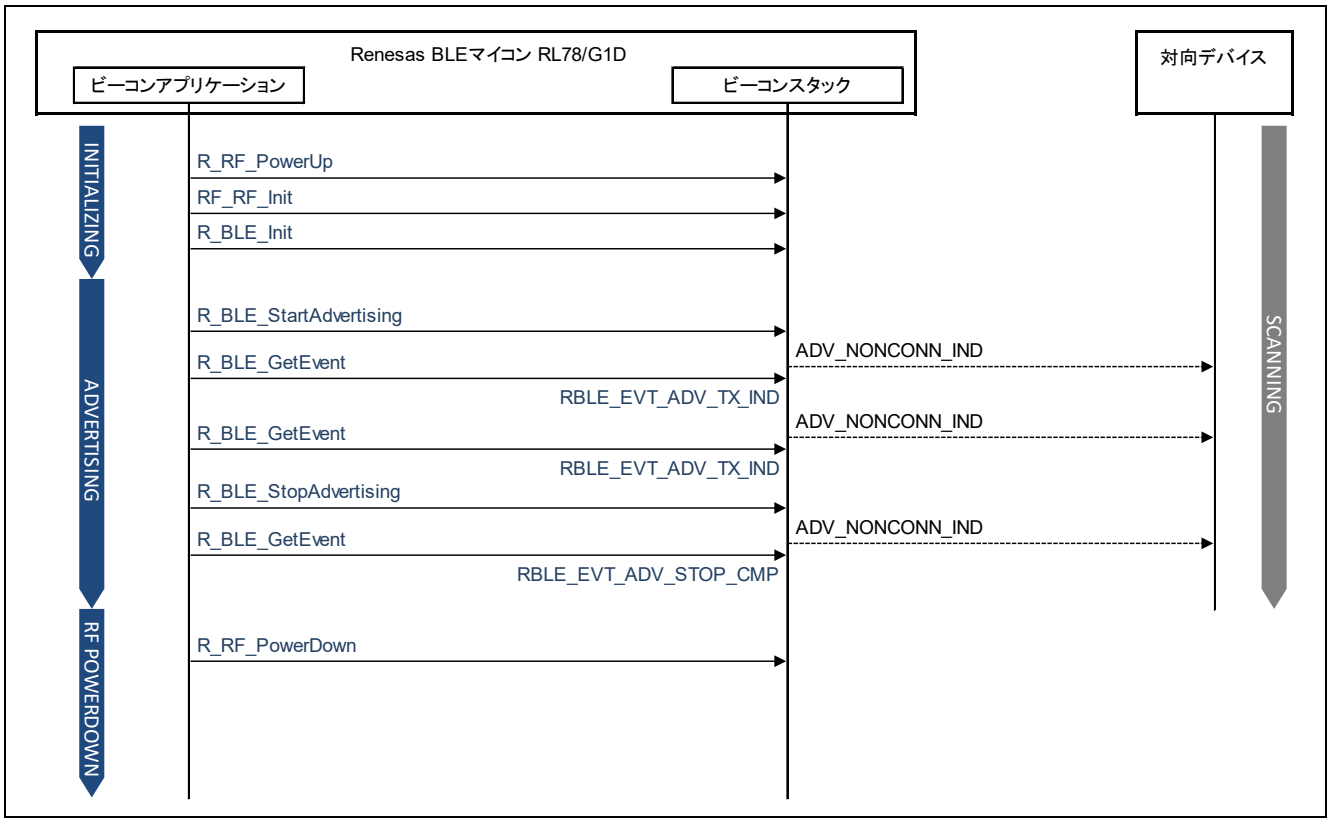


図 8-4　ビーコンアプリケーション Initializing & Advertising & RF Powerdown シーケンス

ビーコンスタック API の仕様については『RL78/G1D ビーコンスタック ユーザーズマニュアル』(R01UW0171)の 4 章「API」を参照してください。

アプリケーションは RF 部を有効化するために、R_RF_PowerUp、R_RF_Init をコールします。その後、ビーコンスタックを初期化するために、R_BLE_Init をコールします。

Project_Source¥application¥src¥beacon¥r_beacon_main.c, line 63-129

```
 63:    void R_BEACON_Main(void)
 64:    {
 72:        /*
 73:         ***************************************************************************************
 74:         * Beacon Stack Initialization
 75:         ***************************************************************************************
 76:        */
 77:        if (RBLE_OK != R_RF_PowerUp(BCN_RF_CFG, RF_32MHZ_WAIT))
 78:        {
 81:            mcu_reset();
 82:        }
 83:
 84:        if (RBLE_OK != R_RF_Init())
 85:        {
 88:            mcu_reset();
 89:        }
 90:
 91:        /* Initialize Beacon Stack */
 92:        interrupt_init();
 93:        R_BLE_Init();
129:    }
```

情報発信のための Advertising を開始するために、R_BLE_StartAdvertising をコールします。

Advertising パケットの送信完了時は RBLE_EVT_ADV_TX_IND イベントが毎回通知されます。

Advertising を停止するために、R_BLE_StopAdvertising をコールします。 Advertising の停止完了後、RBLE_EVT_ADV_STOP_CMP イベントが通知されます。

Project_Source¥application¥src¥beacon¥r_beacon.c, line 132-202

```
132:    bool R_BEACON_Start(void)
133:    {
149:        if(RBLE_OK != R_BLE_StartAdvertising(adv_type, &adv_info))
150:        {
151:            return false;
152:        }
155:    }
 :
163:    void R_BEACON_Exit(void)
164:    {
165:        R_BLE_StopAdvertising();
166:    }
 :
174:    void R_BEACON_EventHandler(void)
175:    {
176:        RBLE_EVT* evt = R_BLE_GetEvent();
177:
178:        while (evt != NULL)
179:        {
180:            switch (evt->type)
181:            {
182:                case RBLE_EVT_ADV_TX_IND:
183:                    /* reach here after transmitting Advertising packet */
184:                    bcn_adv_tx_eventhandler(evt);
185:                    break;
186:
187:                case RBLE_EVT_SCANREQ_RX_IND:
188:                    /* reach here after receiving scan request packet */
189:                    bcn_scanreq_rx_eventhandler(evt);
190:                    break;
191:
192:                case RBLE_EVT_ADV_STOP_CMP:
193:                    /* reach here when advertising is stopped */
194:                    bcn_adv_stop_eventhandler(evt);
195:                    break;
196:
197:                default:
198:                    break;
199:            }
200:            evt = R_BLE_GetEvent();
201:        }
202:    }
```

8.2.2　コネクトアプリケーション

(1)　Initializing & Advertising & Slave Connection (Configurations)シーケンス

コネクトアプリケーションの Initialization 状態、Advertising 状態、Slave Connection (Configurations) 状態のシーケンスを**図 8-5** に示します。コネクトアプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

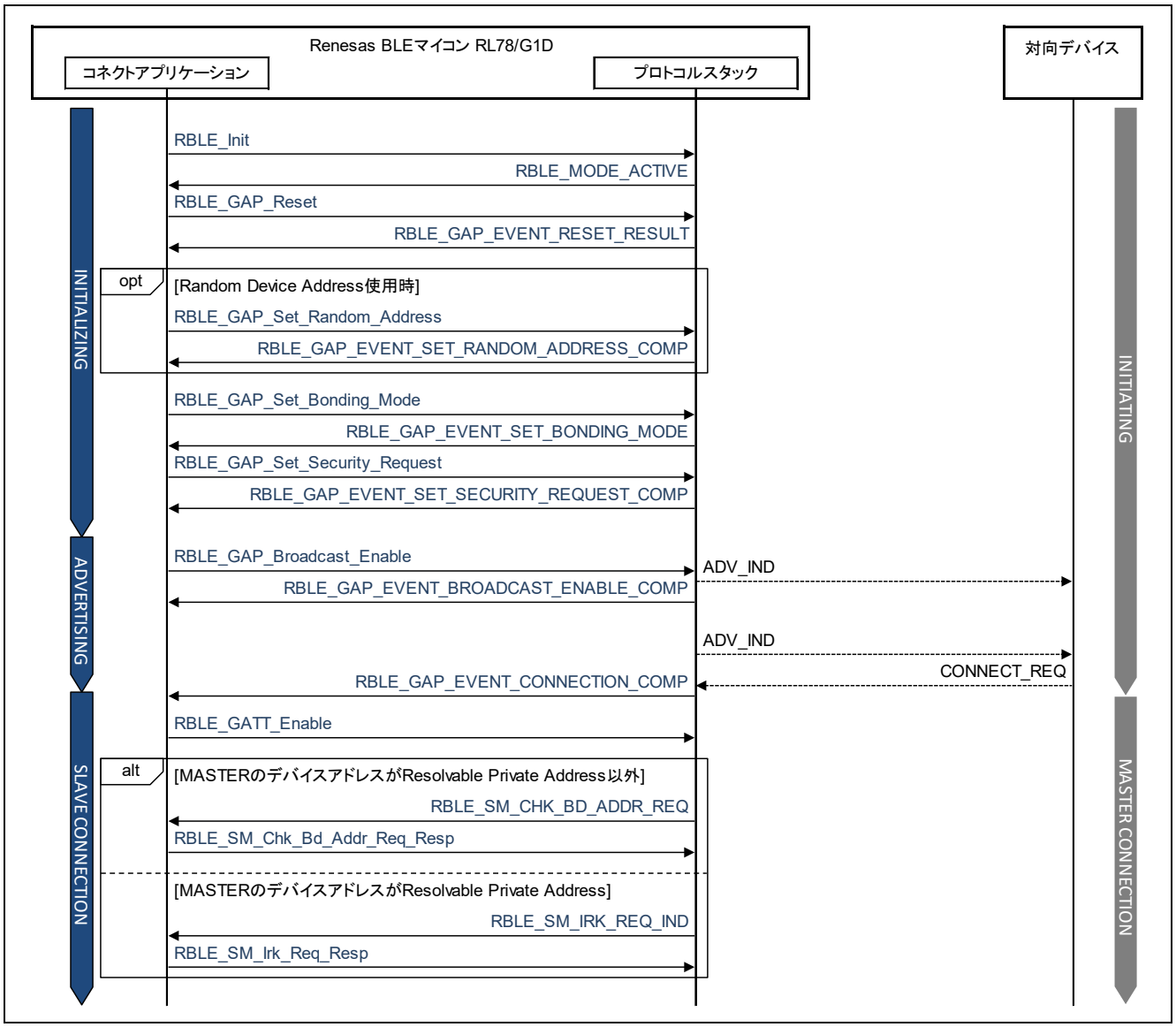


図 8-5　コネクトアプリケーション Initializing & Advertising & Slave Connection (Configurations)シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

アプリケーションは BLE プロトコルスタックの rBLE_Core 初期化のため、RBLE_Init をコールします。

BLE プロトコルスタックの GAP 初期化と GAP イベントコールバック関数、SM イベントコールバック関数を登録するため、RBLE_GAP_Reset をコールします。

Project_Source¥application¥src¥connect¥r_connect.c, line 268-490

```
268:    bool R_CONNECT_Start(void)
269:    {
270:        /* initialize rBLE */
271:        if(RBLE_OK != RBLE_Init(&con_rble_callback))
272:        {
273:            return false;
274:        }
293:    }
 :
374:    static void con_rble_callback(RBLE_MODE mode)
375:    {
383:        switch(mode)
384:        {
385:            case RBLE_MODE_ACTIVE:
386:                /* reach here when activating rBLE is completed after calling RBLE_Init */
387:                con_rble_active_eventhandler();
388:                break;
391:        }
392:    }
 :
400:    static void con_rble_active_eventhandler(void)
401:    {
402:        /* reach here when activating rBLE is completed after calling RBLE_Init */
405:        RBLE_GAP_Reset(&con_gap_callback, &con_sm_callback);
406:    }
 :
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
442:            case RBLE_GAP_EVENT_RESET_RESULT:
443:                /* reach here after RBLE_GAP_Reset is called */
444:                con_gap_reset_eventandler(evt);
445:                break;
446:            case RBLE_GAP_EVENT_SET_RANDOM_ADDRESS_COMP:
447:                /* reach here after RBLE_GAP_Set_Random_Address is called */
448:                con_gap_set_random_address_eventhandler(evt);
449:                break;
450:            case RBLE_GAP_EVENT_SET_BONDING_MODE_COMP:
451:                /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
452:                con_gap_set_bonding_mode_eventhandler(evt);
453:                break;
454:            case RBLE_GAP_EVENT_SET_SECURITY_REQUEST_COMP:
455:                /* reach here after RBLE_GAP_Set_Security_Request is called */
456:                con_gap_set_security_request_eventhandler(evt);
457:                break;
458:            case RBLE_GAP_EVENT_BROADCAST_ENABLE_COMP:
459:                /* reach here after RBLE_GAP_Broadcast_Enable is called */
460:                con_gap_broadcast_enable_eventhandler(evt);
461:                break;
466:            case RBLE_GAP_EVENT_CONNECTION_COMP:
467:                /* reach here when connection occurred */
468:                con_gap_connection_eventhandler(evt);
469:                break;
489:        }
490:    }
```

Random デバイスアドレスを使用する場合、BLE プロトコルスタックに Random デバイスアドレスを設定するために、RBLE_GAP_Set_Random_Address をコールします。

Pairing シーケンスを実行するため、RBLE_GAP_Set_Bonding_Mode と RBLE_GAP_Set_Security_Request をコールします。

上記の初期化完了後、RBLE_GAP_Broadcast_Enable をコールして、Slave として接続を確立するための Advertising を開始します。

Project_Source¥application¥src¥connect¥r_connect.c, line 498-609

```
498:    static void con_gap_reset_eventandler(RBLE_GAP_EVENT* evt)
499:    {
500:        /* reach here after RBLE_GAP_Reset is called */
501:
541:            if(own_type == RBLE_ADDR_RAND)
542:            {
546:                /* Set Random Device Address */
547:                RBLE_GAP_Set_Random_Address(&own_addr);
551:            }
552:            else
553:            {
554:                /* Set Bonding Mode */
555:                RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
556:            }
558:    }
 :
566:    static void con_gap_set_random_address_eventhandler(RBLE_GAP_EVENT* evt)
567:    {
568:        /* reach here after RBLE_GAP_Set_Random_Address is called */
569:
572:            /* Set Bonding Mode */
573:            RBLE_GAP_Set_Bonding_Mode(RBLE_GAP_BONDABLE);
575:    }
 :
583:    static void con_gap_set_bonding_mode_eventhandler(RBLE_GAP_EVENT* evt)
584:    {
585:        /* reach here after RBLE_GAP_Set_Bonding_Mode is called */
586:
589:            /* Set Security Request */
590:            RBLE_GAP_Set_Security_Request(RBLE_GAP_SEC1_NOAUTH_PAIR_ENC);
592:    }
 :
600:    static void con_gap_set_security_request_eventhandler(RBLE_GAP_EVENT* evt)
601:    {
602:        /* reach here after RBLE_GAP_Set_Security_Request is called */
603:
606:            /* Start Broadcast for the First Connection */
607:            RBLE_GAP_Broadcast_Enable(RBLE_GAP_GEN_DISCOVERABLE, RBLE_GAP_UND_CONNECTABLE,       … );
609:    }
```

接続の確立後、RBLE_GAP_EVENT_CONNECTION_COMP イベントが通知されます。

対向デバイスのデバイスアドレスタイプが Resolvable Private Address 以外ならば、RBLE_SM_CHK_BD_ADDR_REQ イベントが通知されます。アプリケーションは以前の接続でのセキュリティ状態を応答するため、RBLE_SM_Chk_Bd_Addr_Req_Resp をコールする必要があります。ただし対向デバイスが Pairing 情報を保持しない場合にペアリングを再実行するため、アプリケーションは常にセキュリティ情報なしと応答します。

対向デバイスのデバイスアドレスが Resolvable Private Address ならば、RBLE_SM_IRK_REQ_IND イベントが通知されます。アプリケーションは以前の接続でのセキュリティ状態と、アドレス解決に必要な Identity Resolving Key (IRK)を応答するため、RBLE_SM_Irk_Req_Resp をコールする必要があります。ただし対向デバイスが Pairing 情報を保持しない場合にペアリングを再実行するため、アプリケーションはセキュリティ状態なし、IRK なしと応答します。

Project_Source¥application¥src¥connect¥r_connect.c, line 815-906

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
826:            case RBLE_SM_CHK_BD_ADDR_REQ:
827:                /* reach here when connection is established to peer device that address is
828:                   public address or random address except resolvable private address       */
829:                con_sm_bdaddr_check_request_eventhandler(evt);
830:                break;
831:            case RBLE_SM_IRK_REQ_IND:
832:                /* reach here when connection is established to peer device that address is
833:                   resolvable private address                                               */
834:                /* IRK is requested for resolving peer's resolvable private address         */
835:                con_sm_irk_request_eventhandler(evt);
836:                break;
858:        }
859:    }
 :
867:    static void con_sm_bdaddr_check_request_eventhandler(RBLE_SM_EVENT* evt)
868:    {
869:        /* reach here when connection is established to peer device that address is
870:           public address or random address except resolvable private address       */
871:
876:        /* Reply BD Address Check Result */
877:        RBLE_SM_Chk_Bd_Addr_Req_Resp(evt->param.chk_bdaddr.idx,
878:                                     0,
879:                                     false,
880:                                     RBLE_SMP_SEC_NONE,
881:                                     NULL);
882:    }
 :
890:    static void con_sm_irk_request_eventhandler(RBLE_SM_EVENT* evt)
891:    {
892:        /* reach here when connection is established to peer device that address is
893:           resolvable private address                                               */
894:        /* IRK is requested for resolving peer's resolvable private address         */
895:
900:        /* Reply IRK(Identity Resolving Key) */
901:        RBLE_SM_Irk_Req_Resp(evt->param.irk_req.idx,
902:                             RBLE_ERR,
903:                            &con_env.con_addr,
904:                             NULL,
905:                             RBLE_SMP_SEC_NONE);
906:    }
```

(2)　Slave Connection (Pairing) Sequence

コネクトアプリケーションの Slave Connection (Pairing)状態のシーケンスを**図 8-6** に示します。コネクトアプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

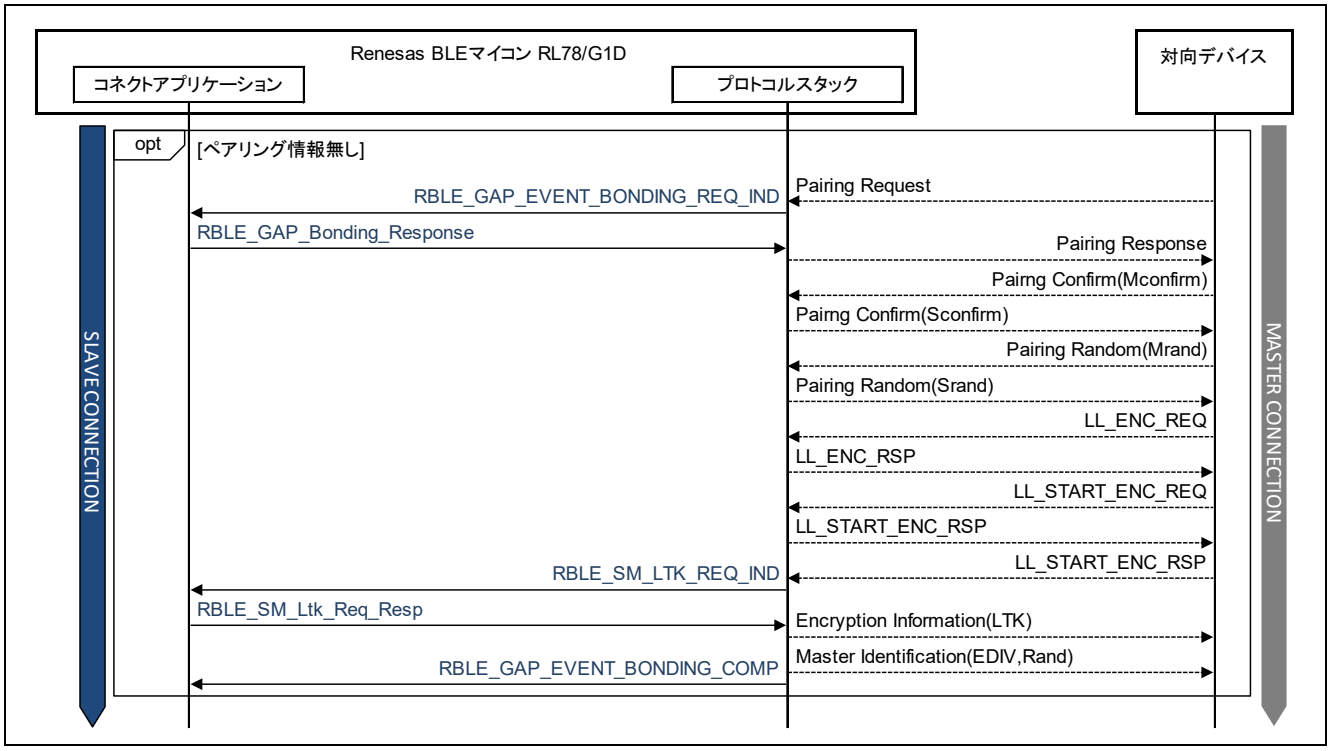


図 8-6　コネクトアプリケーション Slave Connection (Pairing)シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

Pairing していないデバイスとの接続完了後に Pairing Request が送信されると、RBLE_GAP_BONDING_REQ_IND イベントが通知されます。アプリケーションは Pairing Features を応答するために、RBLE_GAP_Bonding_Response をコールします。Pairing Features の交換により、Pairing 方法が決定されます。サンプルプログラムでは、Pairing 方法として Just Works を実行します。Just Works の場合、BLE プロトコルスタックに Temporary Key を設定する必要はありません。以降の Pairing 処理は、BLE プロトコルスタックが生成した Short Term Key によって暗号化されます。

Project_Source¥application¥src¥connect¥r_connect.c, line 414-784

```
414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
415:    {
440:        switch(evt->type)
441:        {
477:            case RBLE_GAP_EVENT_BONDING_REQ_IND:
478:                /* reach here when bonding is requested */
479:                /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
480:                con_gap_bonding_request_eventhandler(evt);
481:                break;
482:            case RBLE_GAP_EVENT_BONDING_COMP:
483:                /* reach here bonding is completed */
484:                /* at the end of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
485:                con_gap_bonding_eventhandler(evt);
486:                break;
489:        }
490:    }
 :
776:    static void con_gap_bonding_request_eventhandler(RBLE_GAP_EVENT* evt)
777:    {
778:        /* reach here when bonding is requested */
779:        /* in the middle of PHASE1: PAIRING FEATURE EXCHANGE in pairing sequence */
780:
781:        /* Reply Bonding Response */
783:        RBLE_GAP_Bonding_Response(&bond_info);
784:    }
```

Short Term Key による暗号化の開始後、RBLE_SM_LTK_REQ_IND イベントが通知されます。アプリケーションは Long Term Key (LTK)を生成し、LTK を応答するため、RBLE_SM_Req_Resp をコールします。LTK は次回以降の接続で、暗号化鍵として使用されます。

LTK の Master デバイスへの提供後、RBLE_SM_KEY_IND イベントが通知され、Bonding Response で指定した暗号化鍵が Master デバイスから提供されます。

Pairing シーケンスが完了すると、RBLE_GAP_EVENT_BONDING_COMP イベントが通知されます。

Project_Source¥application¥src¥connect¥r_connect.c, line 815-940

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
837:            case RBLE_SM_LTK_REQ_IND:
838:                /* reach here when LTK is requested */
839:                /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
840:                con_sm_ltk_request_eventhandler(evt);
841:                break;
842:            case RBLE_SM_KEY_IND:
843:                /* reach here when peer device's encryption information are provided */
844:                /* in the middle of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
845:                con_sm_key_eventhandler(evt);
846:                break;
858:        }
859:    }
 :
914:    static void con_sm_ltk_request_eventhandler(RBLE_SM_EVENT* evt)
915:    {
916:        /* reach here when LTK is requested */
917:        /* in the first of PHASE3: TRANSPORT SPECIFIC KEY DISTRIBUTION in pairing sequence */
918:
933:        /* Reply LTK(Long Term Key) */
934:        RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req.idx,
935:                             RBLE_OK,
936:                             RBLE_SMP_KSEC_NONE,
937:                             pair_info.enc_key.ediv,
938:                            &pair_info.enc_key.nb  ,
939:                            &pair_info.enc_key.ltk );
940:    }
```

(3)　Slave Connection (Start Encryption) Sequence

コネクトアプリケーションの Slave Connection (Start Encryption)状態のシーケンスを**図 8-7** に示します。コネクトアプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

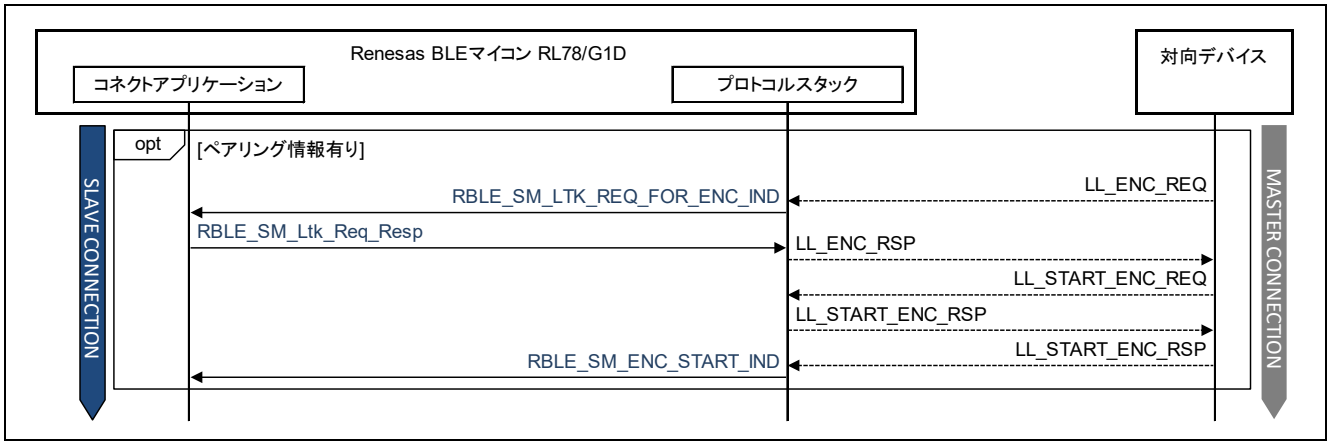


図 8-7　コネクトアプリケーション Slave Connection (Start Encryption)シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

Pairing 済みのデバイスとの接続完了後に Encryption Request が送信されると、RBLE_SM_LTK_REQ_FOR_ENC_IND イベントが通知されます。アプリケーションは Long Term Key (LTK)を応答するため、RBLE_SM_Ltk_Req_Resp をコールします。LTK は以前の接続で生成し交換済みであり、今回の接続での暗号化鍵として使用されます。

Start Encryption シーケンスが完了すると、RBLE_SM_ENC_START_IND イベントが通知されます。

Project_Source¥application¥src¥connect¥r_connect.c, line 815-984

```
815:    static void con_sm_callback(RBLE_SM_EVENT* evt)
816:    {
824:        switch(evt->type)
825:        {
847:            case RBLE_SM_LTK_REQ_FOR_ENC_IND:
848:                /* reach here when LTK is requested */
849:                /* in the first of start encryption sequence */
850:                con_sm_ltk_request_for_enc_eventhandler(evt);
851:                break;
852:            case RBLE_SM_ENC_START_IND:
853:                /* reach here when start encryption sequence is executed */
854:                con_sm_enc_start_eventhandler(evt);
855:                break;
858:        }
859:    }
 :
960:    static void con_sm_ltk_request_for_enc_eventhandler(RBLE_SM_EVENT* evt)
961:    {
962:        /* reach here when LTK is requested */
963:        /* in the first of Start Encryption sequence */
964:
977:        /* Reply LTK(Long Term Key) */
978:        RBLE_SM_Ltk_Req_Resp(evt->param.ltk_req_for_enc.idx,
979:                             status,
980:                             pair_info.sec_prop,
981:                             pair_info.enc_key.ediv,
982:                            &pair_info.enc_key.nb  ,
983:                            &pair_info.enc_key.ltk );
984:    }
```

(4)　Slave Connection (GATT Access) Sequence

コネクトアプリケーションの Slave Connection (GATT Access)状態でのシーケンスを**図 8-8** に示します。コネクトアプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

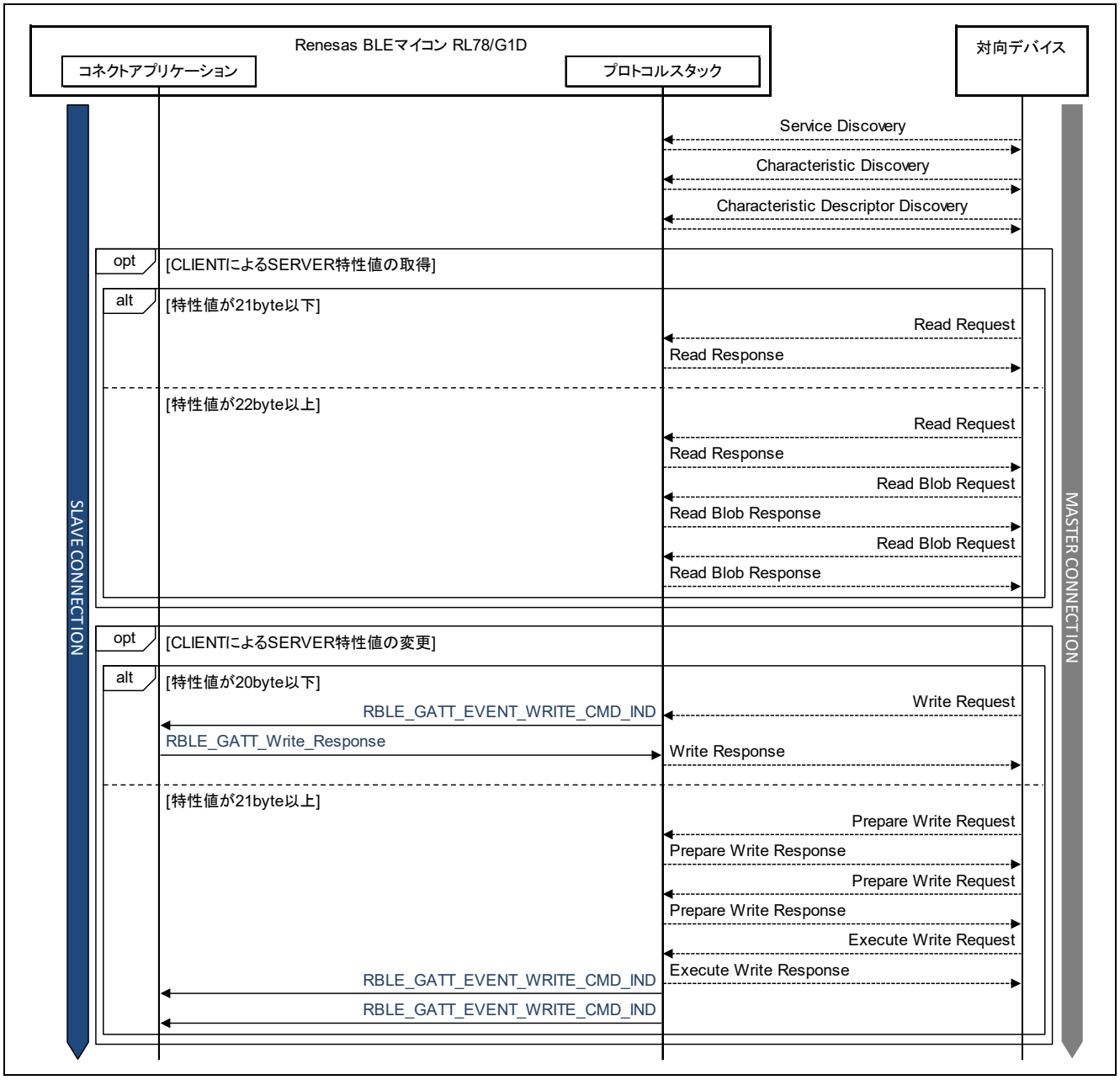


図 8-8　コネクトアプリケーション Slave Connection (GATT Access)シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

Client デバイスから Service Discovery や Characteristic Discovery、Characteristic Descriptor Discovery が要求されると、BLE プロトコルスタックは自動的に応答します。

Client デバイスから Read Request が送信されると、BLE プロトコルスタックは自動的に Read Response を送信します。

Client デバイスから Write Request による Characteristic 値の更新が要求されると、RBLE_GATT_EVENT_WRITE_CMD_IND イベントが通知されます。アプリケーションは Characteristic 値を更新し、Write Response を送信するため、RBLE_GATT_Write_Response をコールします。

Project_Source¥application¥src¥connect¥r_profile.c, line 215-341

```
215:    static void prf_gatt_callback(RBLE_GATT_EVENT* evt)
216:    {
224:        switch(evt->type)
225:        {
226:            case RBLE_GATT_EVENT_WRITE_CMD_IND:
227:                /* reach here when client device requests to write characteristic */
228:
235:                switch(att_hdl)
236:                {
237:                    /* Advertising information (18byte fixed) is written */
238:                    case PRF_HDL_BCNINFO_VAL:
243:                                /* update characteristic value */
256:                        break;
257:
258:                    /* Advertising data (2byte - 32byte variable) is written */
259:                    /* Note: when requested size is over than the size of single write request, */
260:                    /*       characteristic value is transferred separately per 18byte          */
261:                    case PRF_HDL_BCNDATA_VAL:
268:                                /* update characteristic value */
277:                        break;
278:
279:                    /* Scan Response data (2byte - 32byte variable) is written */
280:                    /* Note: when requested size is over than the size of single write request, */
281:                    /*       characteristic value is transferred separately per 18byte          */
282:                    #if !RF_TX_ONLY
283:                    case PRF_HDL_RSPDATA_VAL:
290:                                /* update characteristic value */
299:                        break;
300:                    #endif
304:                }
305:
306:                /* send the write response to client device */
307:                if(evt->param.write_cmd_ind.resp)
308:                {
309:                    prf_send_wr_resp(att_hdl, result);
310:                }
311:                break;
315:        }
316:    }
 :
325:    static void prf_send_wr_resp(uint16_t att_hdl, RBLE_STATUS result)
326:    {
340:        RBLE_GATT_Write_Response(&wr_resp);
341:    }
```

(5)　Disconnection & RF Powerdown Sequence

コネクトアプリケーションの Disconnection 状態と RF Powerdown 状態のシーケンスを**図 8-9** に示します。コネクトアプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

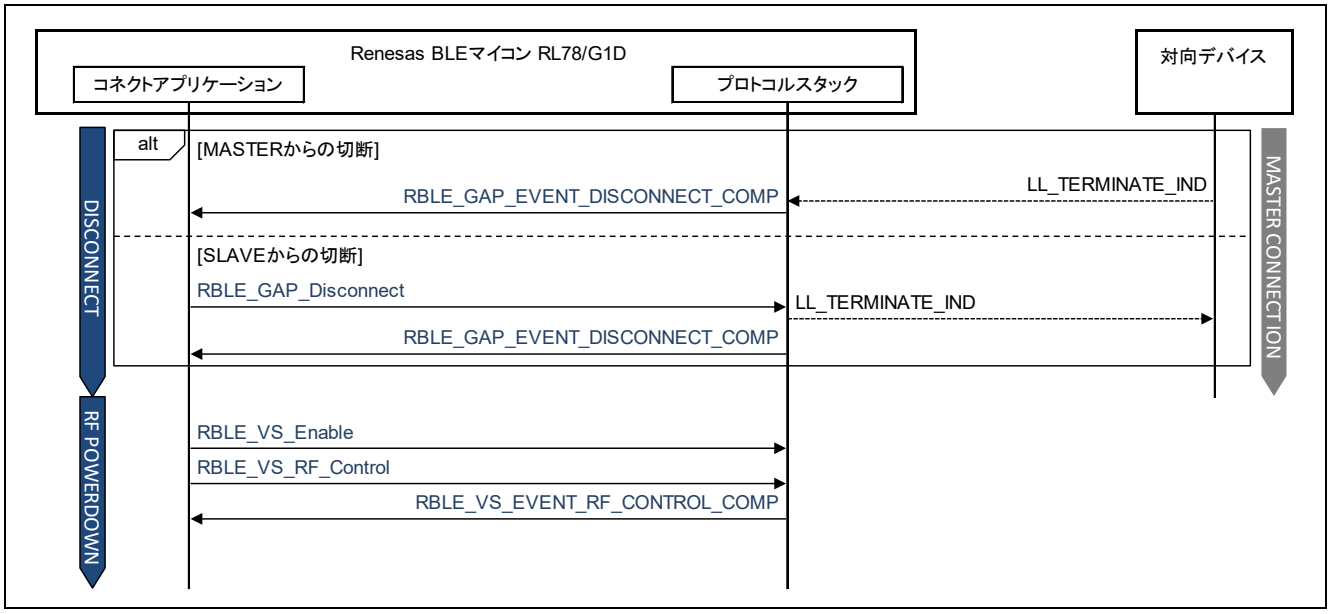


図 8-9　コネクトアプリケーション Disconnection & RF Powerdown シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

Advertising 実行中にアプリケーション終了を要求されたならば、Advertising を停止するため、RBLE_GAP_Broadcast_Disable をコールします。

接続中にアプリケーション終了を要求されたならば、切断するため、RBLE_GAP_Disconnect をコールします。

RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP イベントまたは RBLE_GAP_EVENT_DISCONNECT_COMP イベントが通知されると、RF 部の電源供給を停止するため、RBLE_VS_Enable と RBLE_VS_RF_Control をコールします。

Project_Source¥application¥src¥connect¥r_connect.c, line 414-1085

```
 414:    static void con_gap_callback(RBLE_GAP_EVENT* evt)
 415:    {
 440:        switch(evt->type)
 441:        {
 462:            case RBLE_GAP_EVENT_BROADCAST_DISABLE_COMP:
 463:                /* reach here after RBLE_GAP_Broadcast_Disable is called */
 464:                con_gap_broadcast_disable_eventhandler(evt);
 465:                break;
 470:            case RBLE_GAP_EVENT_DISCONNECT_COMP:
 471:                /* reach here when disconnection occurred */
 472:                con_gap_disconnection_eventhandler(evt);
 473:                break;
 489:        }
 490:    }
  :
 641:    static void con_gap_broadcast_disable_eventhandler(RBLE_GAP_EVENT* evt)
 642:    {
 643:        /* reach here after RBLE_GAP_Broadcast_Disable is called */
 644:
 647:            RBLE_VS_Enable(con_vs_callback);
 648:            RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
 650:    }
  :
 685:    static void con_gap_disconnection_eventhandler(RBLE_GAP_EVENT* evt)
 686:    {
 687:        /* reach here when disconnection occurred */
 688:
 760:            RBLE_VS_Enable(con_vs_callback);
 761:            RBLE_VS_RF_Control(RBLE_VS_RFCNTL_CMD_POWDOWN);
 768:    }
  :
1053:    static void con_vs_callback(RBLE_VS_EVENT* evt)
1054:    {
1076:        switch(evt->type)
1077:        {
1078:            case RBLE_VS_EVENT_RF_CONTROL_COMP:
1079:                /* reach here after RBLE_VS_RF_Control is called */
1080:                con_vs_rf_control_eventhandler(evt);
1081:                break;
1084:        }
1085:    }
```

8.2.3　DTM アプリケーション

(1)　Initializing & Transmitter Test & Receiver Test シーケンス

DTM アプリケーションの Initializing 状態、Transmitter Test 状態、Receiver Test 状態のシーケンスを**図 8-10** に示します。DTM アプリケーションと BLE プロトコルスタックでは、rBLE API が使用されます。

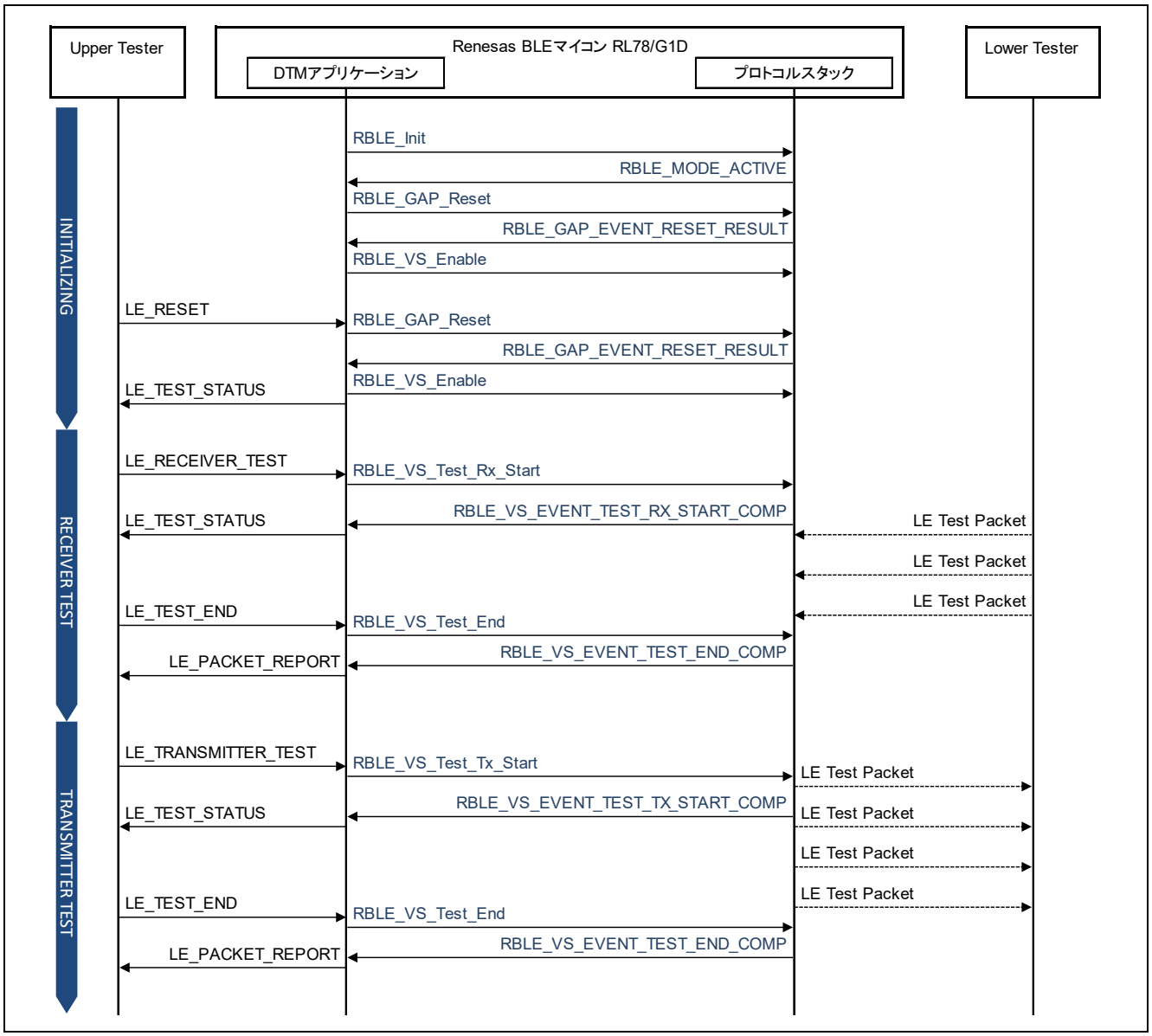


図 8-10　DTM アプリケーション Initializing & Transmitter Test & Receiver Test シーケンス

rBLE API の仕様については『Bluetooth Low Energy プロトコルスタック API リファレンスマニュアル 基本編』(R01UW0088)を参照してください。

DTM アプリケーションは UART 経由で RF テストコマンドを受信することで、RF テストを実行します。また UART 経由で RF テストイベントを送信することで、実行結果を通知します。

LE_RECEIVER_TEST コマンドを受信すると、アプリケーションは RF テスタからの RF テストパケットを受信するため、RBLE_VS_Test_Rx_Start をコールします。

LE_TRANSMITTER_TEST コマンドを受信すると、アプリケーションは RF に RF テストパケットを送信するため、RBLE_VS_Test_Tx_Start をコールします。

LE_TEST_END コマンドを受信すると、RF テストを停止するため、RBLE_VS_Test_End をコールします。

Project_Source¥application¥src¥connect¥r_dtm.c, line 193-384

```
193:    static void dtm_cmdhandler(uint16_t cmd)
194:    {
201:        switch(val16 & DTM_CMD_MASK)
202:        {
203:            case DTM_CMD_RESET:
204:                if(RBLE_OK != RBLE_GAP_Reset(&dtm_gap_callback, &dtm_sm_callback))
205:                {
206:                    dtm_send_error();
207:                }
208:                break;
209:            case DTM_CMD_RX_START:
210:                if(RBLE_OK != RBLE_VS_Test_Rx_Start(DTM_GET_FREQ(val16)))
211:                {
212:                    dtm_send_error();
213:                }
214:                break;
215:            case DTM_CMD_TX_START:
216:                if(RBLE_OK != RBLE_VS_Test_Tx_Start(DTM_GET_FREQ(val16), DTM_GET_LENGTH(val16), … )
217:                {
218:                    dtm_send_error();
219:                }
220:                break;
221:            case DTM_CMD_END:
222:                if(RBLE_OK != RBLE_VS_Test_End())
223:                {
224:                    dtm_send_error();
225:                }
226:                break;
227:        }
228:    }
 :
329:    static void dtm_vs_callback(RBLE_VS_EVENT* evt)
330:    {
333:        switch(evt->type)
334:        {
335:            case RBLE_VS_EVENT_TEST_RX_START_COMP:
336:                /* reach here when RBLE_VS_Test_Rx_Start is called */
348:                break;
349:
350:            case RBLE_VS_EVENT_TEST_TX_START_COMP:
351:                /* reach here when RBLE_VS_Test_Tx_Start is called */
363:                break;
364:
365:            case RBLE_VS_EVENT_TEST_END_COMP:
366:                /* reach here when RBLE_VS_Test_End is called */
379:                break;
383:        }
384:    }
```

9. Appendix

9.1 デバイスアドレス

デバイスアドレスはデバイスを識別するための 48bit の値です。Bluetooth Core Specification で定義されるデバイスアドレスを以下に示します。

- Public Device Address
  - Public Device Address は IEEE 802-2001 のセクション"48-bit universal LAN MAC addresses"に従って生成すべきデバイスアドレスで、IEEE Registration Authority から取得した Organizationally Unique Identifier (OUI)を使用します。
- Random Device Address
  - Static Device Address
    - Static Device Address は 48-bit 乱数値で生成します。デバイスは起動後、新しい乱数値による Static Device Address の生成と使用が可能です。Static Device Address の使用後は、次の起動時まで Static Device Address の値を変更すべきではありません。
  - Private Device Address
    - Non-resolvable Private Address
      - Non-resolvable Private Address は 48-bit 乱数値で生成します。Non-resolvable Private Address は他デバイスからの追跡の可能性を低減するために使用され、一定時間（Bluetooth Core Specification による推奨は 15 分ごと）で変更すべきものです。
    - Resolvable Private Address
      - Resolvable Private Address は 24-bit 乱数値と、乱数値と Identity Resolving Key (IRK)による 24-bit ハッシュ値で構成されます。Resolvable Private Address は他デバイスからの追跡の可能性を低減するために使用され、一定時間（Bluetooth Core Specification による推奨は 15 分ごと）で変更すべきものです。

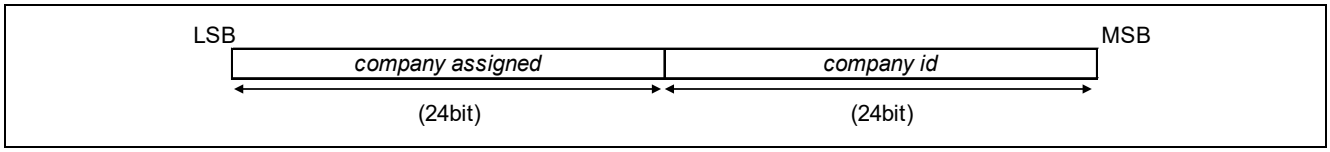


図 9-1　Public Device Address フォーマット

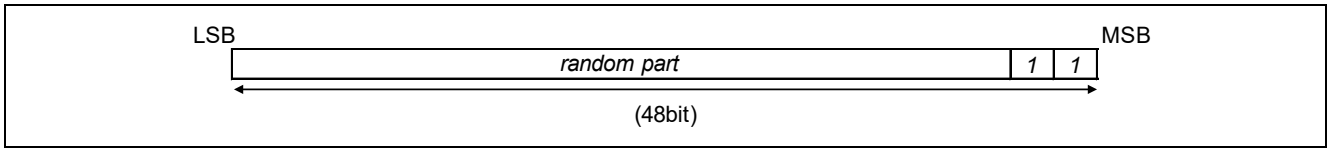


図 9-2　Static Device Address フォーマット

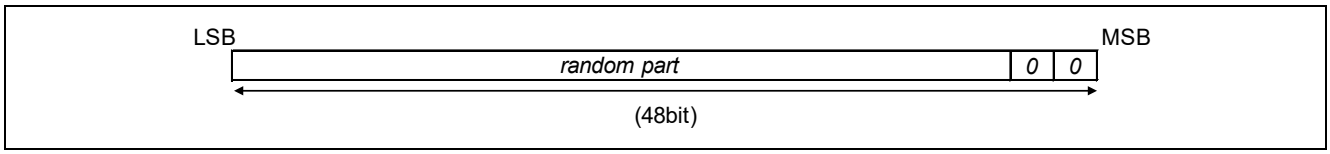


図 9-3　Non-resolvable Private Address フォーマット

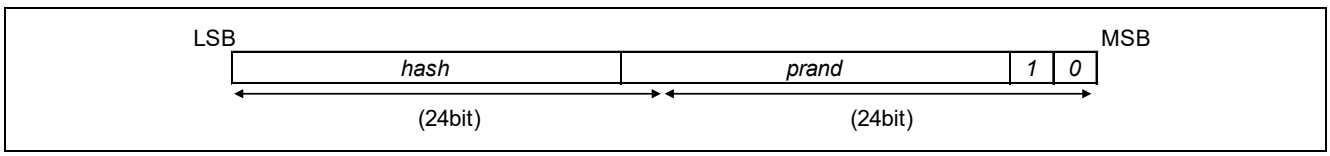


図 9-4　Resolvable Private Address フォーマット

デバイスアドレスの仕様については『Bluetooth Core Specification v4.2』の [Vol. 6, Part B] Section 1.3 を参照してください。

9.2　Advertising パケットフォーマット

ビーコンアプリケーションは Non-connectable Undirected Advertising パケットを送信し、コネクトアプリケーションは Connectable Undirected Advertising パケットを送信します。パケットフォーマットは共通であり、**図 9-5** に示します。

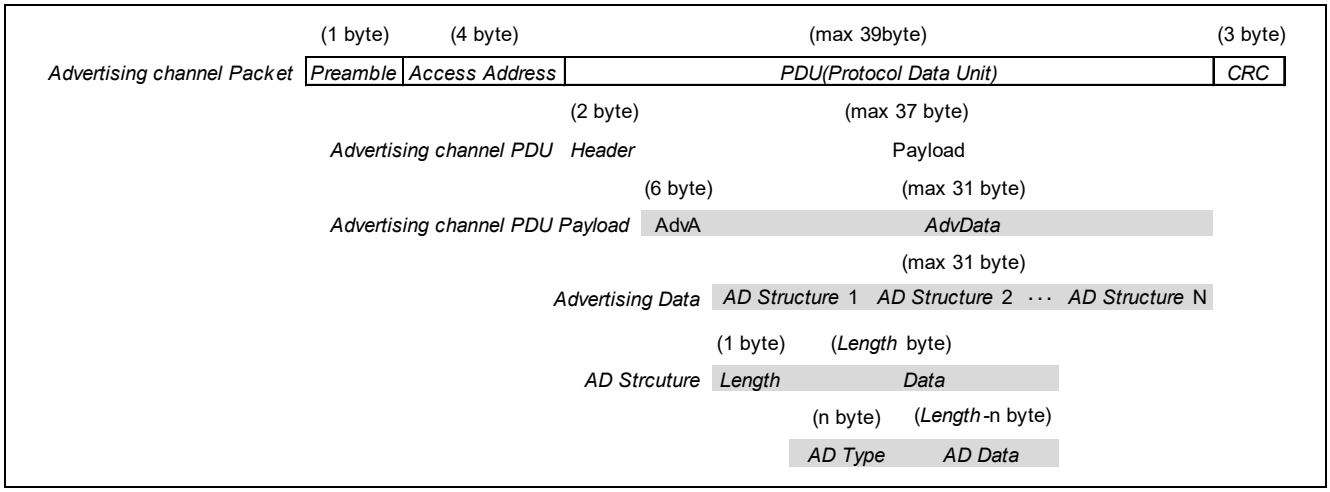


図 9-5　Advertising パケットフォーマット

Advertising パケットは以下のフィールドで構成されます。

- Advertising channel packet
  - Preamble　　: 10101010b 固定
  - Access Address　　: 0x8E89BED6 固定
  - Advertising channel PDU　　: Header と Payload
  - CRC　　: 24bits

下記フィールドはアプリケーションによって設定されます。

- Advertising channel PDU Payload
  - AdvA　　: Advertiser's Address を格納
  - AdvData (Advertising data)　　: 複数の AD structure で構成され、最大 31 bytes
    - AD structure　　: 1 byte の Length 情報と Length byte の Data
      - Data　　: n byte の AD Type と(Length-n) byte の AD Data

詳細についてはそれぞれ下記の仕様を参照してください。

- Advertising packet format　　:『Bluetooth Core Specification v4.2』[Vol. 6, Part B] Section 2.1
- Advertising channel PDU format　　:『Bluetooth Core Specification v4.2』[Vol. 6, Part B] Section 2.3
- Advertising Data format　　:『Bluetooth Core Specification v4.2』[Vol. 3, Part C] Section 11
- AD Type　　:『Supplement to the Bluetooth Core Specification v6』Part A

AD Type の定義については下記の WEB サイトを参照してください。

- Bluetooth SIG Home > Specification > Assigned Numbers > Generic Access Profile
  https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

9.3　Attribute パケットフォーマット

コネクトアプリケーションは接続中に Attribute パケットを送信します。Characteristic Value Read と Characteristic Value Write で使用する Attribute パケットフォーマットを**図 9-6** に示します。

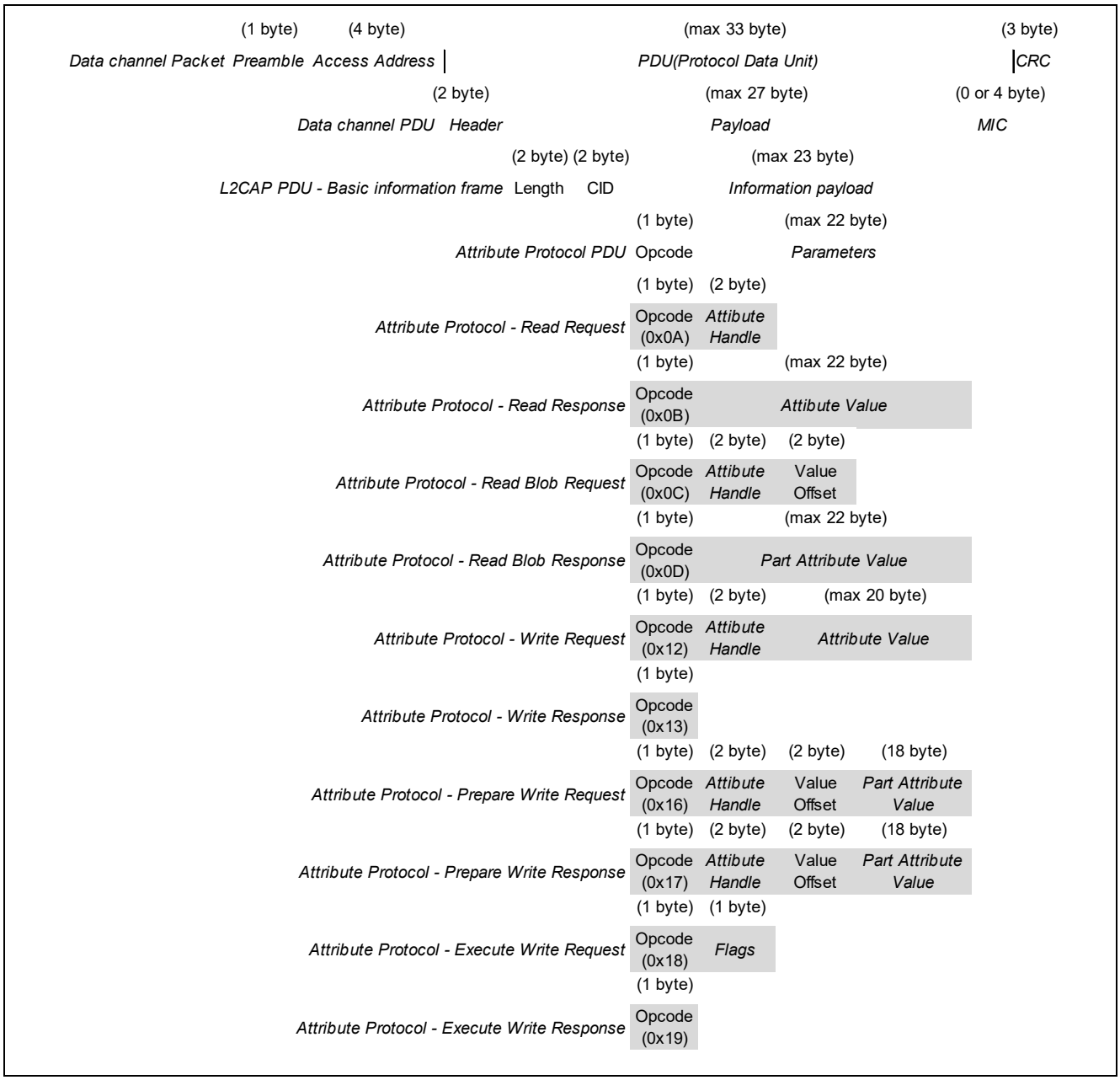


図 9-6　Attribute パケットフォーマット

Attribute パケットは以下のフィールドで構成されます。

- Data channel packet
  - Preamble　　: 10101010b または 01010101b
  - Access Address　　: 接続確立時に Master デバイスが決定
  - Data channel PDU　　: Header と Payload と Message integrity Check (MIC)
    データが暗号化されている場合に MIC が付加される
  - CRC　　: 24bits

- L2CAP PDU
  - Length と Channel ID (CID)と Information payload
  - CID は 0x0004 (Attribute Protocol)
- Attribute Protocol PDU
  - Attribute Opcode と Attribute Parameters
  - Attribute Opcode は Attribute 動作を指定
  - Attribute Parameters は Attribute operation によって異なる

詳細についてはそれぞれ下記の仕様を参照してください。

- Data channel PDU　　: 『Bluetooth Core Specification v4.2』の[Vol. 6, Part B] Section 2.4
- L2CAP PDU　　: 『Bluetooth Core Specification v4.2』の[Vol. 3, Part A] Chapter 3
- Attribute Protocol PDU　　: 『Bluetooth Core Specification v4.2』の[Vol. 3, Part F] Section 3.4
- GATT Features　　: 『Bluetooth Core Specification v4.2』の[Vol. 3, Part G] Chapter 4

9.4　仕様変更点

主な仕様変更点を**表 9-1** に示します。

表 9-1　Rev.1.00→Rev.1.10 の仕様変更点

項目		Rev.1.00	Rev.1.10	記載
動作環境バージョン				
	CC-RL	V1.02.00	V1.04.00	2 章
	CS+ for CC	V3.03.00	V5.00.00	2 章
	e^2 studio	Version 4.3.0.008	Version 5.2.0.023	2 章
	BLE プロトコルスタック	V1.11	V1.20	2 章
	ビーコンスタック	V1.00	V2.10	2 章
ビーコンアプリケーション				
	RF 動作	RF 送信のみ	RF 送信のみ RF 送受信	6.1.2 項
	Advertising Type	Non-connectable Undirected Advertising	Non-connectable Undirected Advertising Scannable Undirected Advertising	5.1.1 項
コネクトアプリケーション				
	Pairing Initiator Key Distribution	ID Resolving key	None	5.2.2 項
	Custom Service Characteristic	Advertising Information Advertising Data Code Flash Memory Updated Count Data Flash Memory Updated Count	Advertising Information Advertising Data Scan Response data Code Flash Memory Updated Count Data Flash Memory Updated Count	5.2.3 項
アプリケーションシーケンス				
	アプリケーション切り替え	ビーコンアプリケーションを起動後、ビーコンアプリケーションとコネクトアプリケーションをスイッチ押下で交互に切替	ビーコンアプリケーションを起動後、ビーコンアプリケーションとコネクトアプリケーションをスイッチ押下で交互に切替 コネクトアプリケーションを起動後、30 秒経過でビーコンアプリケーションに切替	6.1.10 項
フラッシュメモリへの格納データ				
	コードフラッシュ	デバイスアドレス デバイスアドレスタイプ デバイスネーム Advertising 情報 Non-connectable Undirected Advertising データ	デバイスアドレス デバイスアドレスタイプ デバイスネーム Advertising 情報 Non-connectable Undirected Advertising データ Scanable Undirected Advertising データ Scan Response データ	5.4.1 項
	データフラッシュ	Pairing 情報 　対向デバイスアドレス 　対向デバイスアドレスタイプ 　セキュリティ状態 　ローカル暗号化鍵情報 　リモート暗号化鍵情報 乱数シード値 データフラッシュメモリ更新回数 コードフラッシュメモリ更新回数	Pairing 情報 　対向デバイスアドレス 　対向デバイスアドレスタイプ 　セキュリティ状態 　ローカル暗号化鍵情報 データフラッシュメモリ更新回数 コードフラッシュメモリ更新回数	5.4.2 項

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/

改訂記録

Rev.	発行日	改訂内容	
1.00	2016.07.14	初版発行	
1.10	2017.06.30	1 章 概要	
		P.5	ビーコンシステム構成図、RF 評価システム構成図を図 1-1 に統合
		P.6	図 1-2、図 1-3 の図を変更
		-	評価ボード操作の記載を 4 章 動作確認に移動
		2 章 動作環境	
		P.7	ソフトウェア環境、ソフトウェアライブラリのバージョンを更新
		3 章 ファイル構成	
		P.8	ファイル構成に"R5F11AGJ_BcnCmb_no_sw.hex"を追加
		4 章 動作確認	
		P.10	動作確認手順に消費電流の測定を追加
		P.11	ライブラリのダウンロード URL を変更
		P.13	評価ボード操作の記載、図 4-1 を追加
		P.15	評価ボード操作の記載、図 4-2 を追加
		P.15	スマートフォンによる動作確認フローを P.10 動作確認手順に統合
		P.16	Advertising パケット送信の確認の図を追加
		P.17	Advertising パケットの更新の図を追加
		P.17	iOS アプリケーションの GATTBrowser を使用した手順に記載を変更
		P.19	Android アプリケーションの GATTBrowser を使用した手順に記載を変更
		P.21	Advertising パケット更新の確認の図を追加
		P.22	RF 特性の評価の図を追加
		P.23	4.6 節を新規追加
		5 章 仕様	
		P.26	RF 送受信有効時の記載を追加
		P.26	表 5-2 を新規追加
		P.28	表 5-4 の Initiator Key Distribution を None に変更
		P.30	表 5-5 に Scan Response Data Characteristic を追加
		P.30	表 5-5 の Attribute Handle を変更
		P.31	RF テストコマンド/イベント表を分離し表 5-6、表 5-7 を追加
		P.31	RF テストコマンド/イベントの組み合わせ表を削除
		P.32	表 5-8 の表フォーマットを変更
		P.32	表 5-8 に Scannable Advertising データと Scan Response データを追加
		P.33	表 5-9 の Pairing 情報とフラッシュメモリ更新情報のデータ構造を変更
		P.33	表 5-9 から乱数シード値を削除
		P.35	5.6 節 を新規追加
		P.36	コンパイラのバージョンを更新
		P.37	図 5-8、図 5-9、図 5-10 のシステム動作設定領域のサイズを変更
		6 章 設定	
		P.45	6.1.10 項を新規追加
		P.47	ユニークコードファイルに Scan Response データを追加
		P.48	6.2.2 項を新規追加
		P.52	6.2.4 項を新規追加
		P.59	6.2.7 項を新規追加
		9 章 Appendix	
		-	Android アプリのビルドの記載を削除
		P.87	9.4 節を新規追加
		全体	
		-	"ビーコンシステム"を"ビーコン動作"に表記を変更
		-	"RF 評価システム"を"RF 評価動作"に表記を変更

		-	参照すべき関連資料の記載を追加
		-	用語、シンボル名、その他の誤字を修正
1.11	2018.03.30	5 章 仕様	
		P. 34	表 5-12 の補足 URL を変更
		9 章 Appendix	
		P.84	9.2 節の記述を修正
		P.85	9.3 節の記述を修正

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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
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。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.4.0-1　2017.11)

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：https://www.renesas.com/contact/