RX23W グループ

Bluetooth Low Energy と USB の同時利用ガイド

要旨

本書は、RX23W MCU に Bluetooth Low Energy 通信と USB 2.0 通信を同時利用するアプリケーションを実装する場合に留意すべき内容を示します。また Bluetooth Low Energy 通信と USB 2.0 通信を同時利用するサンプルプログラムを紹介します。

実装時に留意すべき内容として次の内容を解説します。

- Bluetooth Low Energy と USB の機能性を維持するためのアプリケーションの実装や Bluetooth Low Energy の動作設定
- FreeRTOS 上で Bluetooth Low Energy と USB を制御するタスクの概要と優先度
- MCU や BLE モジュールを低消費電力状態に遷移させる条件の確認方法
- CLKOUT_RF 端子の出力クロックを USB クロックとして使用する方法

動作確認デバイス

Renesas Solution Starter Kit for RX23W (RTK5523W8AC00001BJ)

目次

1. 想定するシステム構成例

本書は、RX23W MCU で Bluetooth Low Energy(LE)の Central 動作または Peripheral 動作と USB の Peripheral 動作を同時利用する場合に留意すべき内容を示します。

図 1-1 に本書が想定するシステムの構成例を示します。Bluetooth LE 通信には BLE FIT モジュールや QE for BLE がコード生成した GATT サービスを使用します。USB 通信には USB Basic Mini FIT モジュールと PCDC (Peripheral Communication Device Class)などの各 USB Class FIT モジュールを利用します。

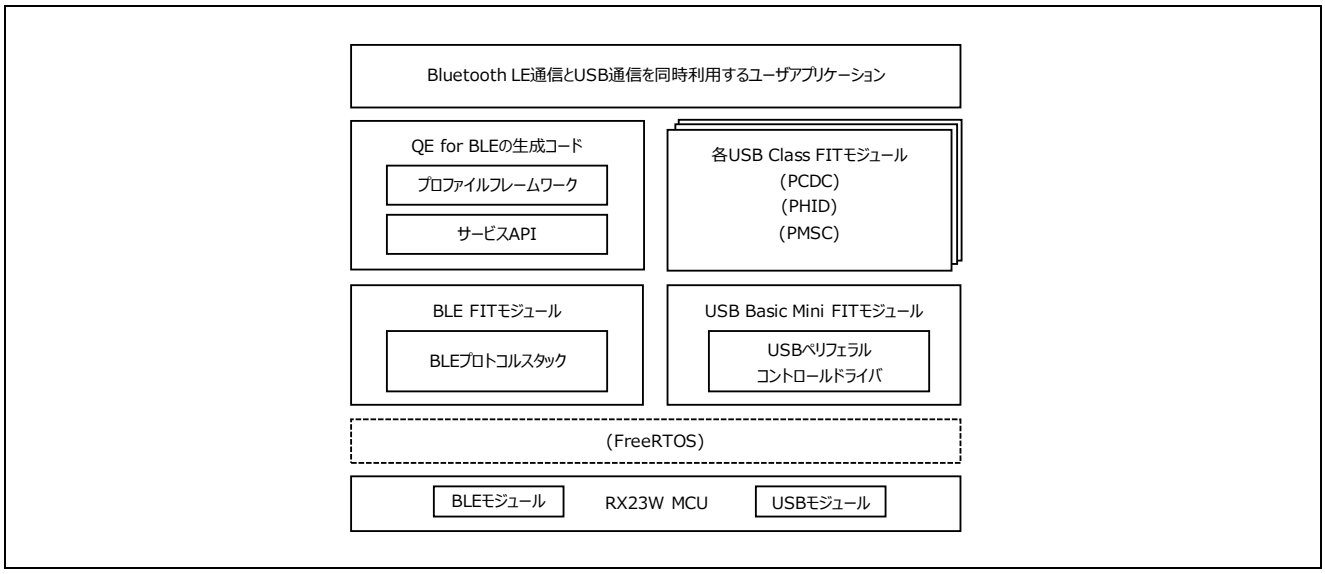

図 1-1　想定するシステム構成例

RX23W MCU や BLE FIT モジュール、QE for BLE、USB Basic Mini FIT モジュール、各 USB Class FIT モジュールの詳細は下記のウェブサイトで公開されているドキュメントをご参照ください。

RX23W
https://www.renesas.com/rx23w

- RX23W グループ ユーザーズマニュアル ハードウェア編 (R01UH0823)

RX ファミリ用 Bluetooth Low Energy プロトコルスタック
https://www.renesas.com/software-tool/bluetooth-low-energy-protocol-stack-rx-family

- RX23W グループ BLE モジュール Firmware Integration Technology (R01AN4860)
- RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)
- RX23W グループ Bluetooth Low Energy プロファイル開発者ガイド (R01AN4553)

Bluetooth Low Energy 対応開発支援ツール QE for BLE
https://www.renesas.com/qe-ble

USB ドライバ
https://www.renesas.com/software-tool/usb-drivers

- RX ファミリ USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) Firmware Integration Technology (R01AN2166)

2. 実装時の留意点

2.1 Bluetooth LE 通信と USB 通信の制御

BLE プロトコルスタックおよび USB ドライバは割り込みハンドラだけでは周辺モジュールの制御が完結せず、メインループ内または OS タスク上での追加の制御処理の実行を必要とします。Bluetooth LE 通信と USB 通信の性能を確保するには、この追加の制御処理の実行が遅延しないようにアプリケーションを実装する必要があります。

2.1.1 BLE プロトコルスタックと USB ドライバの動作概要

BLE プロトコルスタックは、RX23W の BLE モジュールを制御する処理(タスク)をキューで管理します。アプリケーションによる BLE API のコールや、BLE モジュールによる割り込みにより、キューにタスクが追加されます。キューに追加されたタスクは、OS レス環境ではメインループ、FreeRTOS 環境では FreeRTOS タスク上のループでコールされる R_BLE_Execute()によって実行されます。

同様に、USB ペリフェラルコントロールドライバは、RX23W の USB モジュールを制御する処理(タスク)をキューで管理します。アプリケーションによる USB API のコールや、USB モジュールによる割り込みにより、キューにタスクが追加されます。キューに追加されたタスクは、OS レス環境ではメインループで繰り返しコールされる R_USB_GetEvent()、FreeRTOS 環境では USB ペリフェラルコントロールドライバタスク usb_pstd_pcd_task()によって実行されます。

表 2-1　タスク処理関数

周辺モジュール	タスク処理関数	
	OS なし	FreeRTOS
BLE モジュール	R_BLE_Execute()	
USB モジュール	R_USB_GetEvent()	usb_pstd_pcd_task()

図 2-1 に BLE プロトコルスタックと USB ペリフェラルコントロールドライバの動作例を示します。BLE モジュールから割り込み(BLEIRQ)が発生すると、割り込みハンドラ(r_ble_bleirq_interrupt_func)がタスクを BLE 制御キューに追加します。追加されたタスクは R_BLE_Execute()で実行されます。また USB モジュールから割り込み(USBI)が発生すると、割り込みハンドラ(usb_cpu_usb_interrupt)がタスクを USB 制御キューに追加します。追加されたタスクは R_USB_GetEvent()で実行されます。

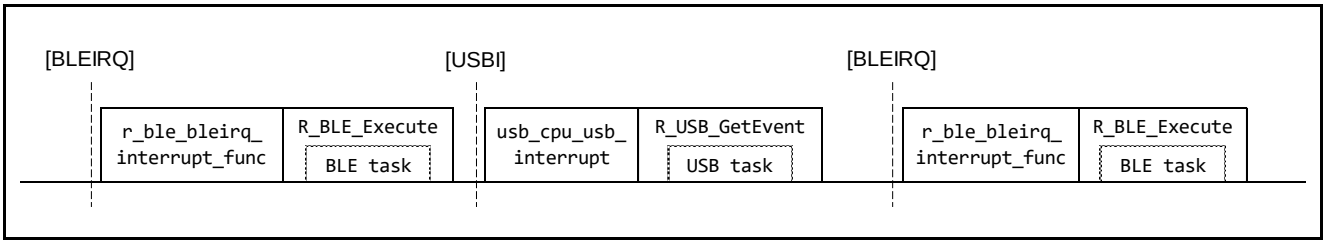

図 2-1　BLE プロトコルスタックと USB ペリフェラルコントロールドライバの動作例

2.1.2 メインループで実行するアプリケーションの処理時間が長い場合の影響

RX23W は 1 つの CPU で BLE 制御と USB 制御とアプリケーションを含むその他の処理を実行します。メインループで実行するアプリケーションが長時間 CPU を占有すると、BLE 制御や USB 制御に影響を及ぼします。特に Bluetooth LE 通信は対向デバイスとタイミングを合わせて送受信するため、BLE 制御が間に合わないと接続が切断される可能性があります。BLE 制御や USB 制御の実行遅延を低減させるため、アプリケーションやその他の処理を実装する際は、割り込みハンドラの実行時間や割り込み禁止時間を可能な限り短くしてください。また OS を使用しない場合、メインループで実行するアプリケーションの処理時間を可能な限り短くしてください。

図 2-2 にアプリケーションと BLE 制御処理の動作例を示します。図中[A]のようにメインループで実行するアプリケーション処理 app_func()の処理時間が長い場合、r_ble_bleirq_interrupt_func()から R_BLE_Execute()を実行するまでの遅延時間が長くなります。図中[B]と[C]のようにメインループのアプリケーション処理を app_func_a()と app_func_b()に分割するなど、R_BLE_Execute()の実行までの遅延時間を短くしてください。特に Bluetooth LE で接続中はこの遅延時間が数 ms 以内となることが推奨されます。アプリケーションの処理時間が長くなる場合は FreeRTOS の利用も検討してください。

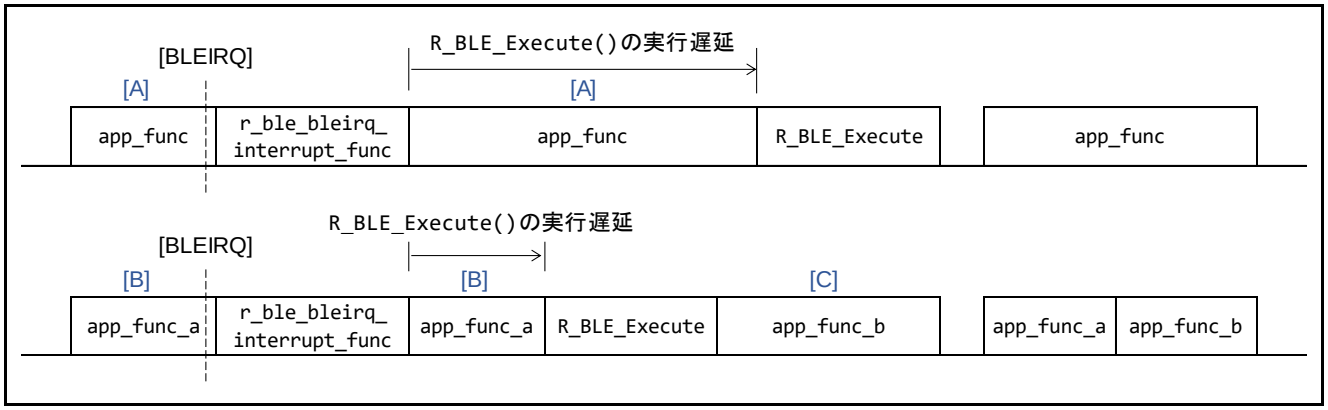

図 2-2　アプリケーションと BLE 制御処理の動作例

2.1.3　R_BLE_Execute()の処理時間が長くなるケース

BLE プロトコルスタックの R_BLE_Execute()は、BLE 制御キューにあるタスクを順次実行します。本関数は、BLE 制御キューにあるすべてのタスクを実行してキューが空となるまで制御を返しません。BLE 制御キューのタスク状況により、R_BLE_Execute()が CPU を長時間占有して、メインループで実行される後続の USB 制御などの処理が遅延する可能性があります。

対向デバイスから連続してデータを受信して、図 2-3 の[A]、[B]、[C]のように割り込みが発生した場合、BLE 制御キューにタスクが連続して追加されるため、R_BLE_Execute()が CPU を長時間占有する可能性があります。R_BLE_Execute()による長時間の CPU 占有を低減するには、スキャンインターバルとスキャンウィンドウ、コネクションインターバルと最大コネクションイベント長(Max_CE_Length)を調整して、スキャンイベントやコネクションイベントの発生頻度を低減させてください。

例えば、スキャンインターバル=60ms かつスキャンウィンドウ=60ms の場合、スキャンイベントのデューティ比は 100%となります。スキャンウィンドウ=30ms とすることでスキャンイベントのデューティ比は 50%となり、Advertising パケット受信による割り込みの頻度が低下し、R_BLE_Execute()による長時間の CPU 占有を低減することができます。

同様に、コネクションインターバル=50ms かつ最大コネクションイベント長=50ms の場合、コネクションイベントの最大デューティ比は 100%となります。最大コネクションイベント長=25ms とすることでコネクションイベントの最大デューティ比は 50%となり、データパケット受信による割り込みの頻度が低下し、R_BLE_Execute()による長時間の CPU 占有を低減することができます。

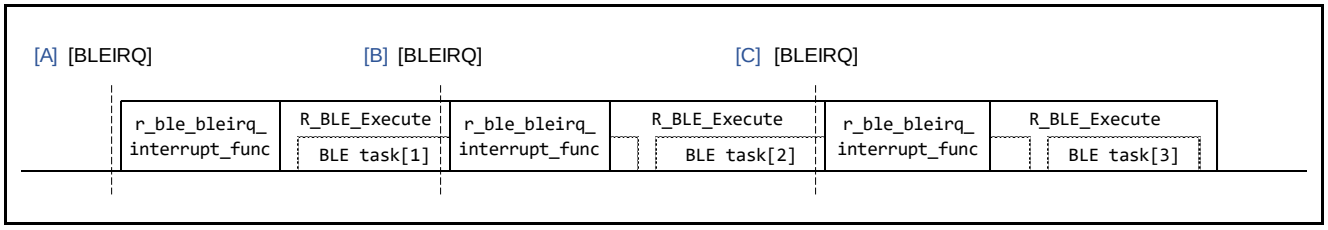

図 2-3　Bluetooth LE 通信のイベント発生頻度が高い場合

BLE プロトコルスタックに登録したコールバック関数もまた R_BLE_Execute()のタスクから実行されます。図 2-4 の図中[A]のようにコールバック関数内の処理時間が長い場合、BLE 制御キューにある後続のタスクが遅延して、Bluetooth LE 通信の性能を低下させる可能性があります。コールバック関数の処理時間は可能な限り短くしてください。

例えば、1M-PHY で接続中に 244 バイトの GATT サービスデータを連続して受信する場合、コネクションイベントの送受信動作は 2.5ms 間隔で実行されます。この場合コールバック関数は 2.5ms 間隔でコールされるため、コールバック関数の処理時間は 2.5ms 以内とすることが推奨されます。

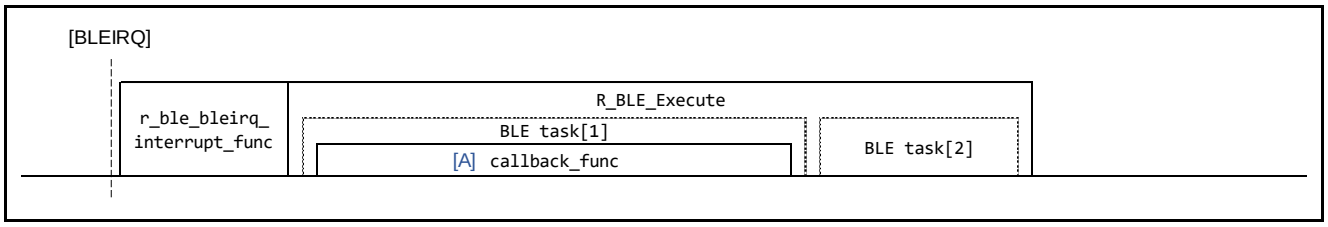

図 2-4　Bluetooth LE イベントのコールバック動作時間が長い動作例

BLE プロトコルスタックの R_BLE_SetEvent()により、任意のユーザ関数をタスクとして BLE 制御キューに追加できます。追加したユーザ関数は R_BLE_Execute()で実行されます。しかしながら、図 2-5 の[A]のように追加されたユーザ関数の処理時間が長い場合、BLE 制御キューにある後続のタスクの実行が遅延して Bluetooth LE 通信の性能を低下させる可能性があります。ユーザ関数の処理時間が長い場合は、図中[B]と[C]の複数の関数に分割して R_BLE_SetEvent()で BLE 制御キューに再度追加してください。

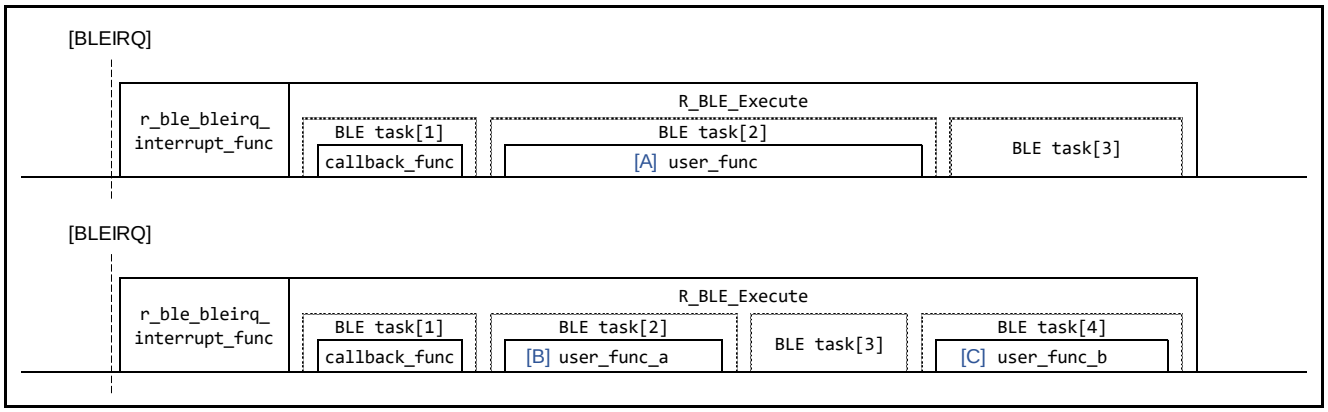

図 2-5　BLE 制御キューに追加されたユーザ関数の動作例

2.1.4　Bluetooth LE のコネクションインターバルの影響

図 2-6 に接続中の Bluetooth LE デバイスの送受信動作例を示します。Bluetooth LE 通信で接続の確立後、Central デバイスと Peripheral デバイスがコネクションインターバルごとに送受信動作を交互に繰り返し、1 つのコネクションイベントで交互にデータを送信できます。図中の[A]または[B]でアプリケーションが送信を要求したデータは、図中[C]で示した次のコネクションイベントで送信されます。

コネクションインターバルを長くするほど(最大 4sec)、送受信動作の頻度が減って CPU 占有率や消費電力を低減できる一方、アプリケーションが BLE プロトコルスタックにデータ送信を要求してから実際に送信するまでの遅延時間が長くなる可能性があります。

コネクションインターバルを短くするほど(最小 7.5msec)、送信遅延時間は短縮されますが、送信すべきデータがない場合も Empty パケットの送受信動作を実行するため、MCU と BLE モジュールによる消費電力が増大します。また BLE プロトコルスタックによる BLE モジュール制御処理の実行頻度が増えるため、アプリケーションが利用可能な CPU 時間が減少します。

上記に留意し、システムの要件に適したコネクションインターバルを設定してください。なおコネクションインターバルは接続後に動的に変更することもできます。

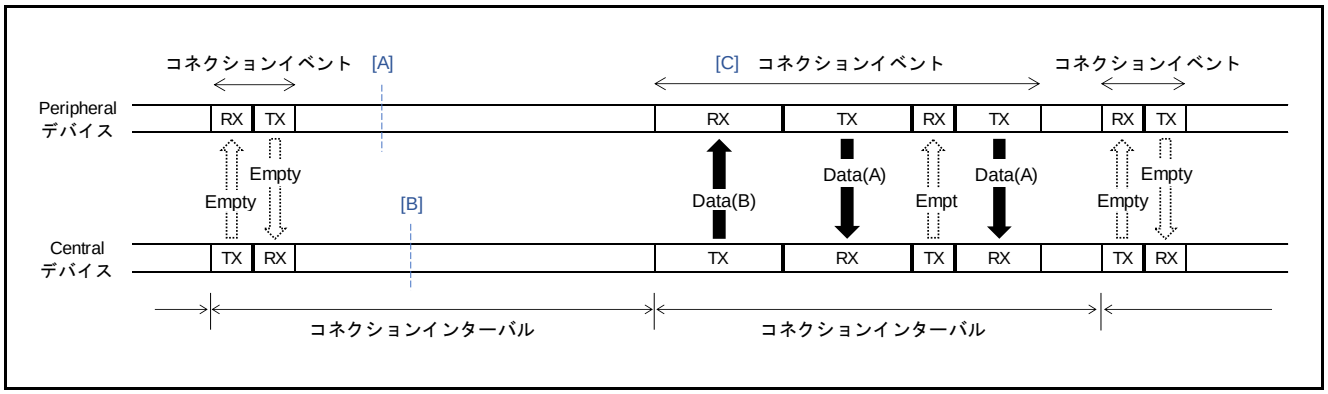

図 2-6　接続中の Bluetooth LE デバイスの送受信動作例

さらに図 2-7 にペリフェラルレイテンシーが 2 の場合の Bluetooth LE デバイスの送受信動作例を示します。ペリフェラルレイテンシーは Peripheral デバイスが実行するコネクションイベントを減らす回数です。ペリフェラルレイテンシーに 0 以外の値を設定することで、Peripheral デバイスは指定した回数だけイベントを実行せず、消費電力を低減できます。また Central デバイスは常にコネクションイベントを実行するため、Peripheral デバイスはペリフェラルレイテンシーに関係なく任意のタイミングでコネクションイベントを実行してデータを送受信することもできます。

図中の[A]は Central デバイスのアプリケーションによるデータの送信要求を示します。Peripheral デバイスがペリフェラルレイテンシーによりコネクションイベントを実行しない場合、Central デバイスから送信されたデータは Peripheral デバイスに受信されません。Central デバイスは Peripheral デバイスがデータを受信するまで、各コネクションイベントでデータを再送します。このようにペリフェラルレイテンシーが設定されている場合、Central デバイスのデータを Peripheral デバイスが受信完了するまでの遅延時間が長くなる可能性があります。

上記に留意し、システムの要件に適したペリフェラルレイテンシーを設定してください。なおペリフェラルレイテンシーは接続後に動的に変更することもできます。

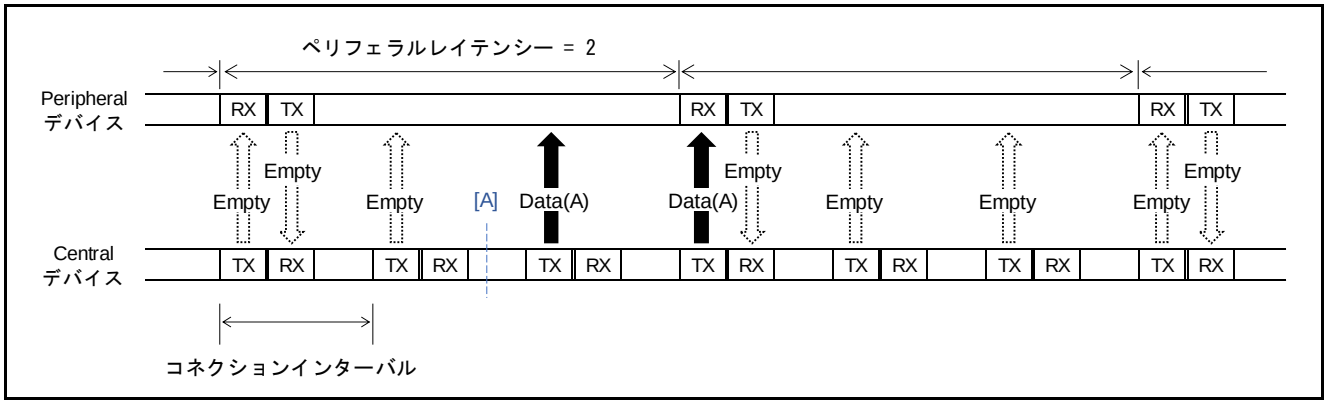

図 2-7　接続中の Bluetooth LE デバイスの送受信動作例 (ペリフェラルレイテンシー = 2)

2.1.5　USB 通信のフロー制御の必要性

図 2-8 に USB 通信のデータ送受信動作例を示します。図中の[A]~[D]はアプリケーションによるデータ送信要求を示します。USB 通信(USB 2.0)は半二重であり、Host デバイスが受信または送信を制御します。図中[C]のように Host デバイスがデータを送信し続けると、Peripheral デバイスからのデータが送信できません。必要に応じて、Host デバイスがデータを長時間受信できない状態を回避するためのフロー制御を Host デバイスのアプリケーションに実装してください。

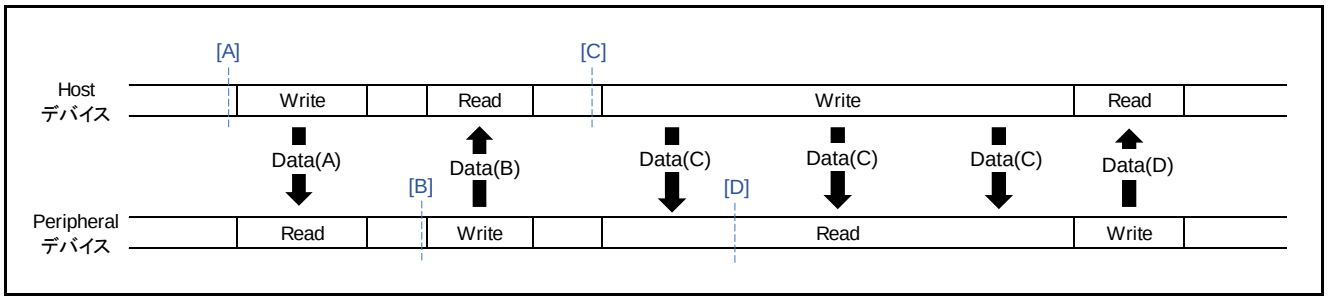

図 2-8　USB 通信のデータ送受信動作例

2.1.6　Bluetooth LE 通信と USB 通信のデータ再送の必要性

BLE プロトコルスタックと USB ドライバはリソースの動的確保の失敗などにより、受信データが欠落する場合があります。必要に応じて、アプリケーションで受信データの完全性を検証し、受信データの欠落を検出した場合にアプリケーションで再送要求する仕組みを実装することを推奨します。

2.2　FreeRTOS タスクの設定

BLE プロトコルスタックと USB ドライバの処理は遅延時間が短くなるようにアプリケーションを実装する必要があります。アプリケーションの処理時間が長くなる場合や複数のアプリケーションを実装する場合は FreeRTOS の利用を推奨します。

2.2.1　BLE 制御と USB 制御のためのタスクの優先度設定

FreeRTOS 上で BLE プロトコルスタックを使用する場合、R_BLE_Execute()を繰り返しコールして BLE モジュールを制御するためのタスクを生成する必要があります。BLE プロトコルスタックに登録したコールバック関数は R_BLE_Execute()で実行されるため、BLE イベントをトリガとするアプリケーションの処理時間が長い場合、アプリケーション向けの新しいタスクを生成してください。

FreeRTOS 上で USB ペリフェラルコントロールドライバを使用する場合、USB モジュールを制御するための USB ペリフェラルコントロールドライバタスク(usb_pstd_pcd_task)が自動生成されます。

表 2-2　FreeRTOS 上で必要となるタスク

タスク	優先度	概要
BLE タスク (app_main)	12 (USB ドライバと同時利用時の推奨値)	R_BLE_Execute()を繰り返しコールして BLE モジュールを制御するためのタスク。 BLE イベントを通知するコールバック関数は本コンテキストで実行される。 BLE タスクは QE for BLE が生成するソースコードに含まれる。
BLE GATT アプリケーションタスク (任意のタスク名)	0～7	GATT サービスを利用したデータ送受信のためのアプリケーションタスク。 ユーザが実装する。
USB ペリフェラルコントロールドライバタスク (usb_pstd_pcd_task)	11 (デフォルト)	USB モジュールを制御するためのタスク。 USB イベントを通知するコールバック関数は本コンテキストで実行される。 本タスクは USB ペリフェラルコントロールドライバが自動生成する。
USB アプリケーションタスク (任意のタスク名)	0～7	USB によるデータ送受信のためのアプリケーションタスク。 ユーザが実装する。

BLE プロトコルスタックの BLE タスクは、コネクションインターバルなどのタイミングパラメータに従って BLE モジュールを制御する必要があります。このため、BLE タスクは BLE GATT アプリタスクよりも優先度を高くしてください。

USB ペリフェラルコントロールドライバタスクは、USB モジュールの送受信性能を確保するため FIFO バッファへの読み書きを高速に制御する必要があります。このため、USB ペリフェラルコントロールドライバのタスクは USB アプリケーションタスクよりも優先度を高くしてください。

Bluetooth LE 通信性能への影響を低減するため、USB ペリフェラルコントロールドライバタスクの優先度よりも BLE タスクの優先度を高くすることを推奨します。また各アプリケーションタスクの優先度は各ユースケースにおいて十分に評価してください。

2.3　消費電力の低減

MCU を低消費電力状態に遷移させると、CPU や周辺モジュールの動作が停止します。製品の機能性を損なうことなく MCU を低消費電力状態に遷移させるには、使用する全ての周辺モジュールが低消費電力状態への遷移条件を満たしていることを確認する必要があります。Bluetooth LE と USB の利用時は、BLE プロトコルスタックと USB ドライバが遷移条件を満たしていることを確認してください。

2.3.1　MCU の低消費電力状態

RX23W MCU をプログラム動作状態である通常動作モードから、低消費電力状態であるスリープモード、ディープスリープモード、ソフトウェアスタンバイモードのいずれかに遷移させることで消費電力を低減することができます。各モードでの動作状態は各周辺モジュールで異なります。アプリケーションは MCU を低消費電力状態のいずれかのモードに遷移させる前に、BLE プロトコルスタックや USB ペリフェラルコントロールドライバが低消費電力状態に遷移可能な状態であることを確認する必要があります。

スリープモード、ディープスリープモードからの復帰はノンマスカブル割り込みや全割り込みをトリガとして使用できるのに対して、ソフトウェアスタンバイモードからの復帰は、ノンマスカブル割り込みと一部の割り込みのみトリガとして使用できます。なお BLE モジュールの BLEIRQ 割り込み、USB モジュールの USBR 割り込みはソフトウェアスタンバイモードから復帰可能な割り込みです。

2.3.2　FreeRTOS 使用時の MCU 低消費電力状態への遷移

デフォルト設定の FreeRTOS では RX23W の CMT モジュールをシステムタイマとして使用します。CMT を含む各周辺モジュールは、ソフトウェアスタンバイモードでは動作が停止します。FreeRTOS 環境において MCU をソフトウェアスタンバイモードに遷移させる場合は、システムタイマが停止しても FreeRTOS 上で動作する各ソフトウェアの動作に影響がないことを検証してください。

2.3.3　BLE の MCU 低消費電力状態への遷移条件

BLE プロトコルスタックを使用する場合、BLE 制御キューが空ならば、MCU を低消費電力状態に遷移させることができます。BLE 制御キューが空であるかは `R_BLE_IsTaskFree()`で確認できます。他の周辺モジュールの停止と再開や、MCU の低消費電力状態を管理するための `R_BLE_LPC_EnterLowPowerMode()`もまた提供されます。停止中の周辺モジュールへのアクセスを防止するため、FreeRTOS 環境において `R_BLE_LPC_EnterLowPowerMode()`の実行中はタスク切り替えを禁止してください。

低消費電力状態への遷移後、BLE プロトコルスタックは BLE モジュールの割り込みによって MCU を通常動作モードに復帰させます。

2.3.4　BLE モジュールの RF スリープモードへの遷移条件

BLE モジュールの状態は MCU の状態に依存しないため、MCU が低消費電力状態であっても BLE モジュールは動作を継続します。BLE プロトコルスタックは自動的に BLE モジュールを RF スリープモードに遷移させることで、消費電力を低減します。BLE モジュールを RF スリープモードに遷移させるためには、Bluetooth LE のイベント間に十分な RF 休止期間がある必要があります。

図 2-9 に Bluetooth LE 通信動作中の RF 休止期間を示します。Bluetooth LE の各動作を実行中、RF 休止期間が 80ms 以上となるように、アドバタイジングインターバル、スキャンインターバルとスキャンウィンドウ、コネクションインターバルを調整してください。なおコネクションイベント(図中[A])の実行間隔が延長されて直後の RF 休止期間(図中[B])が 80ms 未満となった場合、本 RF 休止期間では BLE モジュールは RF スリープモードに遷移しません。また同時に複数の Bluetooth LE デバイスと接続を確立するなどして RF 休止期間が短くなる場合も同様に、BLE モジュールは RF スリープモードに遷移しません。

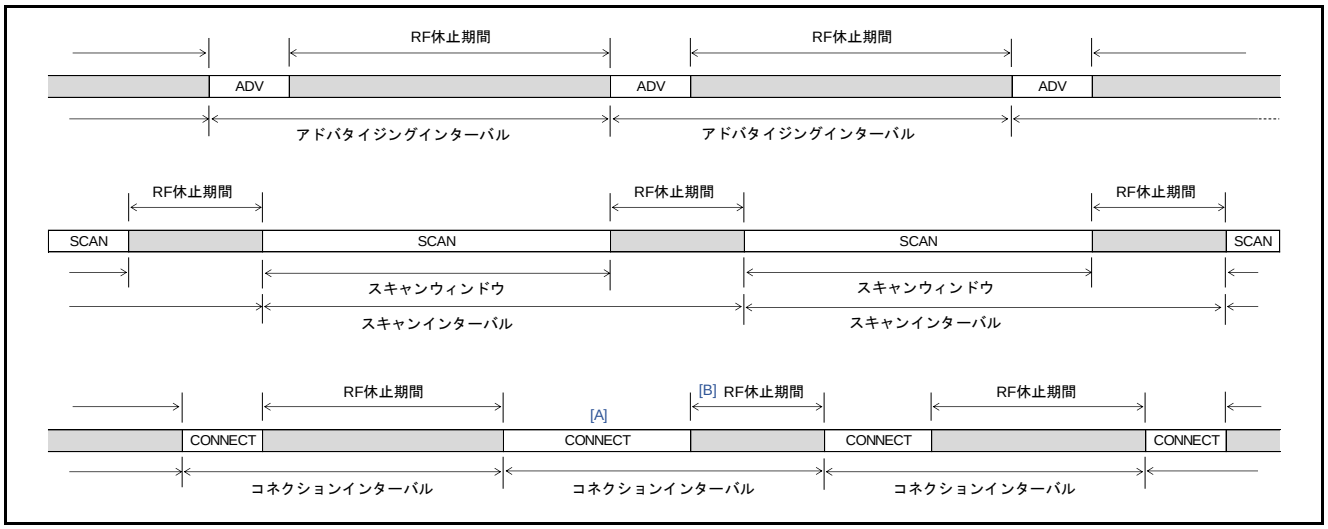

図 2-9　Bluetooth LE 通信動作中の RF 休止期間

2.3.5　USB の MCU 低消費電力状態への遷移条件

USB ドライバを使用する場合、USB モジュールの状態により低消費電力状態のいずれかのモードに遷移させることができます。表 2-3 に下記のウェブサイトで公開されているサンプルプログラムの MCU モード遷移条件を示します。

USB ドライバ
https://www.renesas.com/software-tool/usb-drivers

- RX ファミリ USB ペリフェラル コミュニケーションデバイスクラスドライバ(PCDC) for USB Mini Firmware による USB ホストとの USB 通信を行うサンプルプログラム (R01AN2296)

VBUS の状態は割り込みステータスレジスタ(INTSTS0)の VBSTS ビットで確認できます。また USB デバイスステートは割り込みステータスレジスタ(INTSTS0)の DVSQ[2:0]ビットで確認できます。USB デバイスステートは USB ドライバの `R_USB_GetInformation()`が返す `info.status` の値でも確認できます。

表 2-3　MCU モード遷移のための USB 条件

MCU モード	遷移条件	
	VBUS	USB デバイスステート (`info.status`)
ソフトウェアスタンバイモード	OFF	Powered (`USB_STS_DETACH`)
スリープモード	ON	Suspend (`USB_STS_SUSPEND`)
通常動作モード	ON	Suspend 以外 (`USB_STS_SUSPEND` 以外)

2.4　クロックの設定

USB モジュールの動作に必要な USB クロック(UCLK)の生成のための基準クロックとして、CLKOUT_RF 端子から出力した 4MHz クロックを使用できます。これによりメインクロック発振のための外付け発振子が不要となり、製品の BOM コストを削減することができます。

2.4.1　CLKOUT_RF 端子出力クロックを USB クロックとして使用するための設定

RX23W は、Bluetooth 専用 32MHz クロックを基準とした 1,2,4MHz のいずれかの周波数のクロックを CLKOUT_RF 端子から出力することができます。また CLKOUT_RF 端子の出力クロックは EXTAL 端子から入力してメインクロックとして使用できます。CLKOUT_RF 端子から出力した 4MHz のクロックをメインクロックとして使用し、USB 専用 PLL 回路で 12 逓倍することで、48MHz の USB クロックとして使用できます。CLKOUT_RF 端子の出力クロックを USB クロックとして使用するためのスマートコンフィグレータの設定例は後述の図 3-2 を参照してください。

CLKOUT_RF 端子のクロック出力を許可するには、BLE FIT モジュールの BLE_CFG_RF_CLKOUT_EN マクロを変更してください。CLKOUT_RF 端子からのクロック出力は R_BLE_Open()の実行で開始します。

CLKOUT_RF 端子の出力クロックをメインクロックとして使用する場合、BSP FIT モジュールの BSP_CFG_CLKOUT_RF_MAIN マクロを 1 に設定してください。R_BLE_Open()の実行後に高速オンチップオシレータ(HOCO)の発振を停止することができます。

CLKOUT_RF 端子からのクロック出力は R_BLE_Close()の実行で停止します。CLKOUT_RF 端子の出力クロックをメインクロックとして使用する場合、HOCO の発振停止後は R_BLE_Close()を実行しないでください。また CLKOUT_RF 端子の出力クロックを USB クロックとして使用する場合、USB モジュールの使用中に R_BLE_Close()を実行しないでください。

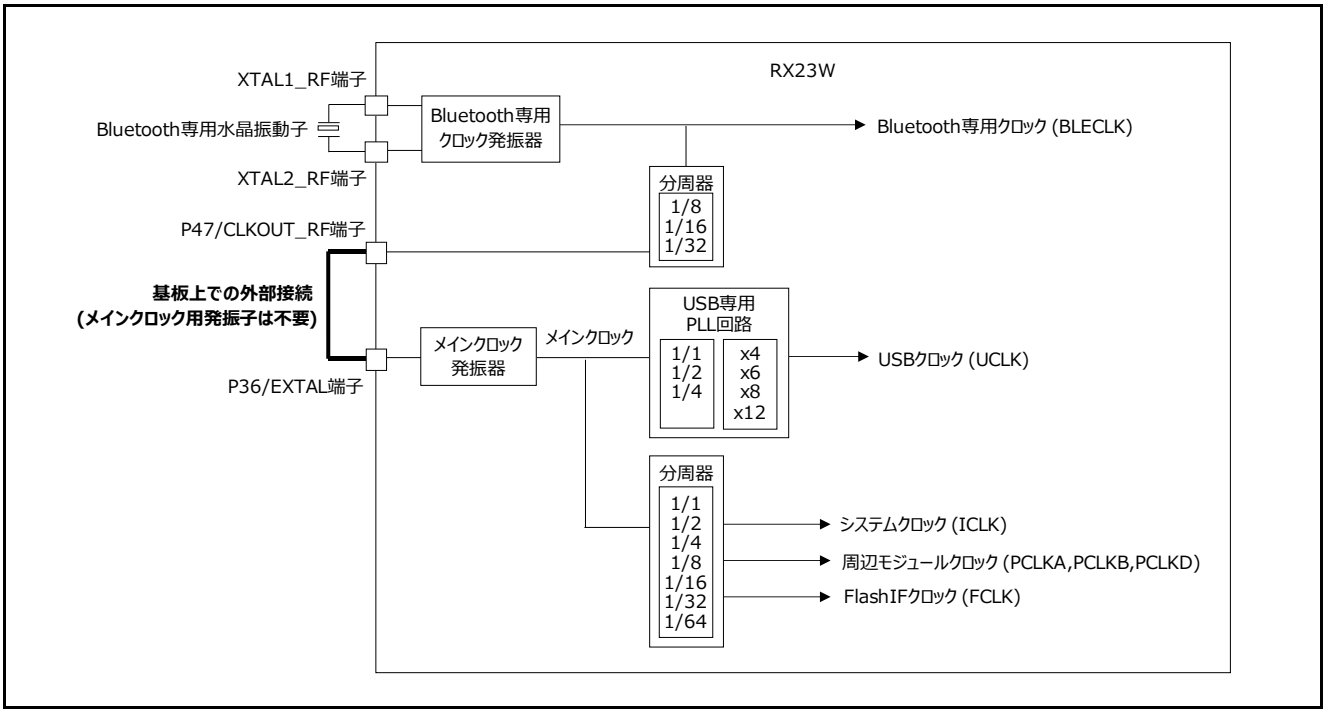

図 2-10　CLKOUT_RF 端子出力クロックの使用例

3. サンプルプログラム

3.1 概要

本章はアプリケーションノートのサンプルプログラムの概要を示します。表 3-1 サンプルプログラムが使用する周辺モジュールを示します。

表 3-1　サンプルプログラムが使用する周辺モジュール

周辺モジュール	用途
Bluetooth Low Energy (BLE)	対向の評価ボードとの Bluetooth LE 無線通信
USB2.0 ホスト/ファンクションモジュール (USB0)	USB Host デバイス(PC)との USB 通信
シリアルコミュニケーションインタフェース (SCI8)	PC 上のコンソールへの UART による動作ログ出力
I/O ポート (P41～P44)	評価ボードの LED 制御

図 3-1 にサンプルプログラムのデモ構成を示します。デモでは評価ボードを２台使用します。PC は２台または１台でも評価可能です。PC と評価ボードは USB 通信のための USB ケーブルと、UART-USB 通信のための USB ケーブルで接続します。USB 通信のための接続と UART-USB 通信のための接続は、それぞれ異なる COM ポートとして認識されます。また２台の評価ボードは自動で Bluetooth LE による接続を確立します。

PC 上では USB 通信のためのコンソールと UART-USB 通信のためのコンソールを使用します。UART-USB 通信のコンソールには Bluetooth LE 通信の接続・切断やデータの送受信、USB 通信の接続・切断やデータの送受信を示す動作ログが表示されます。

USB 通信のコンソールに文字が入力されると、PC から評価ボードに文字列データが送信されます。データを受信した評価ボードは Bluetooth LE で他方の評価ボードにデータを送信します。また Bluetooth LE でデータを受信した評価ボードは USB で PC にデータを送信し、PC 上の USB 通信のコンソールに文字列が表示されます。

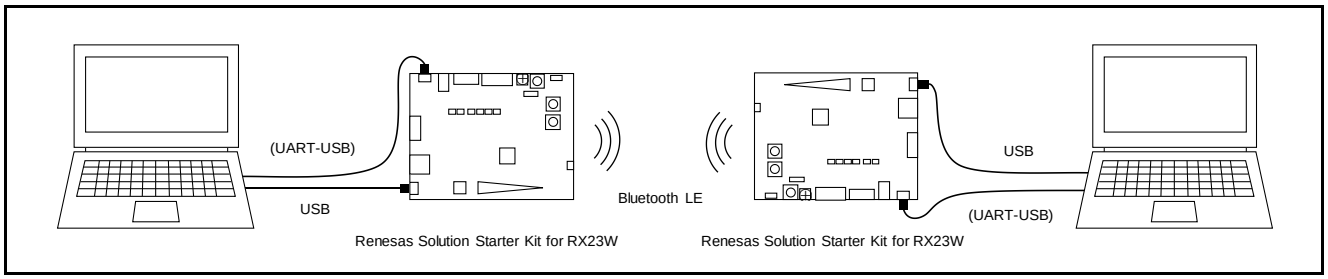

図 3-1　サンプルプログラムのデモ構成

3.2　プロジェクト

表 3-2 にサンプルプログラムのプロジェクト構成を示します。各プロジェクトで RTOS の有無、Bluetooth LE 動作が異なります。いずれのプロジェクトも USB 動作に差異はありません。

表 3-2　サンプルプログラムのプロジェクト構成

デモプロジェクト	RTOS	Bluetooth LE 動作	USB 動作
rsskrx23w_osless_gattclient_pcdc	OS なし	GAP Central Data I/O Service Client	USB Peripheral Communication Device Class (CDC)
rsskrx23w_osless_gattserver_pcdc		GAP Peripheral Data I/O Service Server	
rsskrx23w_freertos_gattclient_pcdc	FreeRTOS	GAP Central Data I/O Service Client	
rsskrx23w_freertos_gattserver_pcdc		GAP Peripheral Data I/O Service Server	

表 3-3 にサンプルプログラムのプログラムサイズを示します。

表 3-3　サンプルプログラムのプログラムサイズ

デモプロジェクト	セクションサイズ	
	ROMDATA+PROGRAM	RAMDATA
rsskrx23w_osless_gattclient_pcdc	212,962byte	44,978byte
rsskrx23w_osless_gattserver_pcdc	212,773byte	45,152byte
rsskrx23w_freertos_gattclient_pcdc	226,796byte	60,761byte
rsskrx23w_freertos_gattserver_pcdc	226,612byte	60,937byte

表 3-4 にサンプルプログラムが使用する FIT モジュールとそのバージョンを示します。

表 3-4　サンプルプログラムの FIT モジュール構成

FIT モジュール	バージョン
RX23W グループ BLE FIT モジュール (r_ble_rx23w)	2.30
RX ファミリ ボードサポートパッケージモジュール (r_bsp)	6.21
RX ファミリ バイト型キューバッファモジュール (r_byteq)	1.90
RX ファミリ CMT モジュール (r_cmt_rx)	5.00
RX ファミリ LPC モジュール (r_lpc_rx)	2.03
RX ファミリ Renesas FreeRTOS (FreeRTOS_Kernel)	1.0.102
RX ファミリ Renesas FreeRTOS (FreeRTOS_Object)	1.0.104
RX ファミリ GPIO モジュール (r_gpio_rx)	4.20
RX ファミリ IRQ モジュール (r_irq_rx)	3.90
RX ファミリ SCI モジュール (r_sci_rx)	4.00
RX ファミリ USB Basic Mini モジュール (r_usb_basic_mini)	1.20
RX ファミリ USB ペリフェラル コミュニケーションデバイスクラスドライバ (r_usb_pcdc_mini)	1.20

表 3-5、表 3-6、表 3-7、表 3-8 にサンプルプログラムで使用する BSP FIT モジュール、BLE FIT モジュール、USB Basic Mini FIT モジュール、USB PCDC FIT モジュールの主な設定を示します。

表 3-5　サンプルプログラムの BSP FIT モジュール設定(r_bsp_config.h)

項目	設定値	設定内容
BSP_CFG_CLOCK_SOURCE	1	システムクロックのソース：HOCO
BSP_CFG_USB_CLOCK_SOURCE	1	USB クロックのソース：USB 専用 PLL 回路
BSP_CFG_MAIN_CLOCK_SOURCE	1	メインクロックの発振器：外部発振入力
BSP_CFG_XTAL_HZ	4000000	外部発振入力のクロック周波数：4MHz
BSP_CFG_UPLL_DIV BSP_CFG_UPLL_MUL	1 12	USB 専用 PLL 回路の逓倍率：12 倍
BSP_CFG_HOCO_FREQUENCY	0	HOCO の周波数：32MHz
BSP_CFG_CLKOUT_RF_MAIN	1	EXTAL 端子の入力クロックソース：CLKOUT_RF 端子
BSP_CFG_MCU_VCC_MV	3300	VCC 電圧：3.3V

表 3-6　サンプルプログラムの BLE FIT モジュール設定(r_ble_rx23w_config.h)

項目	設定値	設定内容
BLE_CFG_LIB_TYPE	1	BLE プロトコルスタックライブラリのタイプ：Balance
BLE_CFG_RF_CONN_MAX	1	最大同時接続台数：1
BLE_CFG_RF_CONN_DATA_MAX	251	最大接続データ長：251byte
BLE_CFG_RF_ADV_DATA_MAX	31	最大アドバタイジングデータ長：31byte
BLE_CFG_RF_DDC_EN	0	RF トランシーバ用電源：リニアレギュレータ (DC-DC コンバータは無効)
BLE_CFG_RF_EXT32K_EN	0	Bluetooth 専用低速クロック：Bluetooth 専用低速オンチップオシレータ
BLE_CFG_RF_SCA	250	Sleep Clock Accuracy(SCA)：250ppm
BLE_CFG_RF_MAX_TX_POW	1	最大送信パワー：+4dBm
BLE_CFG_RF_DEF_TX_POW	0	デフォルト送信パワー：+4dBm
BLE_CFG_RF_CLKOUT_EN	5	CLKOUT_RF 端子からの出力クロック：4MHz
BLE_CFG_RF_DEEP_SLEEP_EN	1	RF スリープモードへの遷移：有効
BLE_CFG_GATT_MTU_SIZE	247	Exchange MTU で通知する ATT_MTU サイズ：247byte
BLE_CFG_CMD_LINE_EN	1	コマンドライン：有効
BLE_CFG_CMD_LINE_CH	8	コマンドラインの SCI チャネル：8
BLE_CFG_BOARD_LED_SW_EN	1	ボード LED&スイッチ：有効
BLE_CFG_BOARD_TYPE	2	ボードタイプ：RSSK for RX23W
BLE_CFG_ABS_API_EN	1	Abstraction API：有効
BLE_CFG_SOFT_TIMER_EN	1	ソフトウェアタイマ：有効
BLE_CFG_MCU_LPC_EN	1	MCU 低消費電力：有効

表 3-7　サンプルプログラムの USB Basic Mini FIT モジュール設定(r_usb_basic_config.h)

項目	設定値	設定内容
USB_CFG_MODE	USB_CFG_PERI	USB 動作モード：USB Peripheral モード
USB_CFG_DEVICE_CLASS	USB_CFG_PCDC_USE	デバイスクラス：Peripheral Communication Device Class
USB_CFG_DTC	USB_CFG_DISABLE	DTC：使用しない
USB_CFG_DMA	USB_CFG_DISABLE	DMA：使用しない
USB_CFG_BC	USB_CFG_DISABLE	Battery Charging 機能：使用しない
USB_CFG_REGULATOR	USB_CFG_DISABLE	USB レギュレータ：使用しない

表 3-8　サンプルプログラムの USB PCDC FIT モジュール設定(r_usb_pcdc_mini_config.h)

項目	設定値	設定内容
USB_CFG_PCDC_BULK_IN	USB_PIPE1	1st VCOM CDC Data class Bulk In Pipe：パイプ 1
USB_CFG_PCDC_BULK_OUT	USB_PIPE2	1st VCOM CDC Data class Bulk Out Pipe：パイプ 2
USB_CFG_PCDC_INT_IN	USB_PIPE6	1st VCOM CDC Data class Interrupt In Pipe：パイプ 6

図 3-2 にスマートコンフィグレータのクロック設定を示します。サンプルプログラムは CLKOUT_RF 端子から出力した 4MHz のクロックをメインクロックとして使用し、さらに USB 専用 PLL 回路でメインクロックを 12 逓倍することで、48MHz の USB クロックとして使用します。

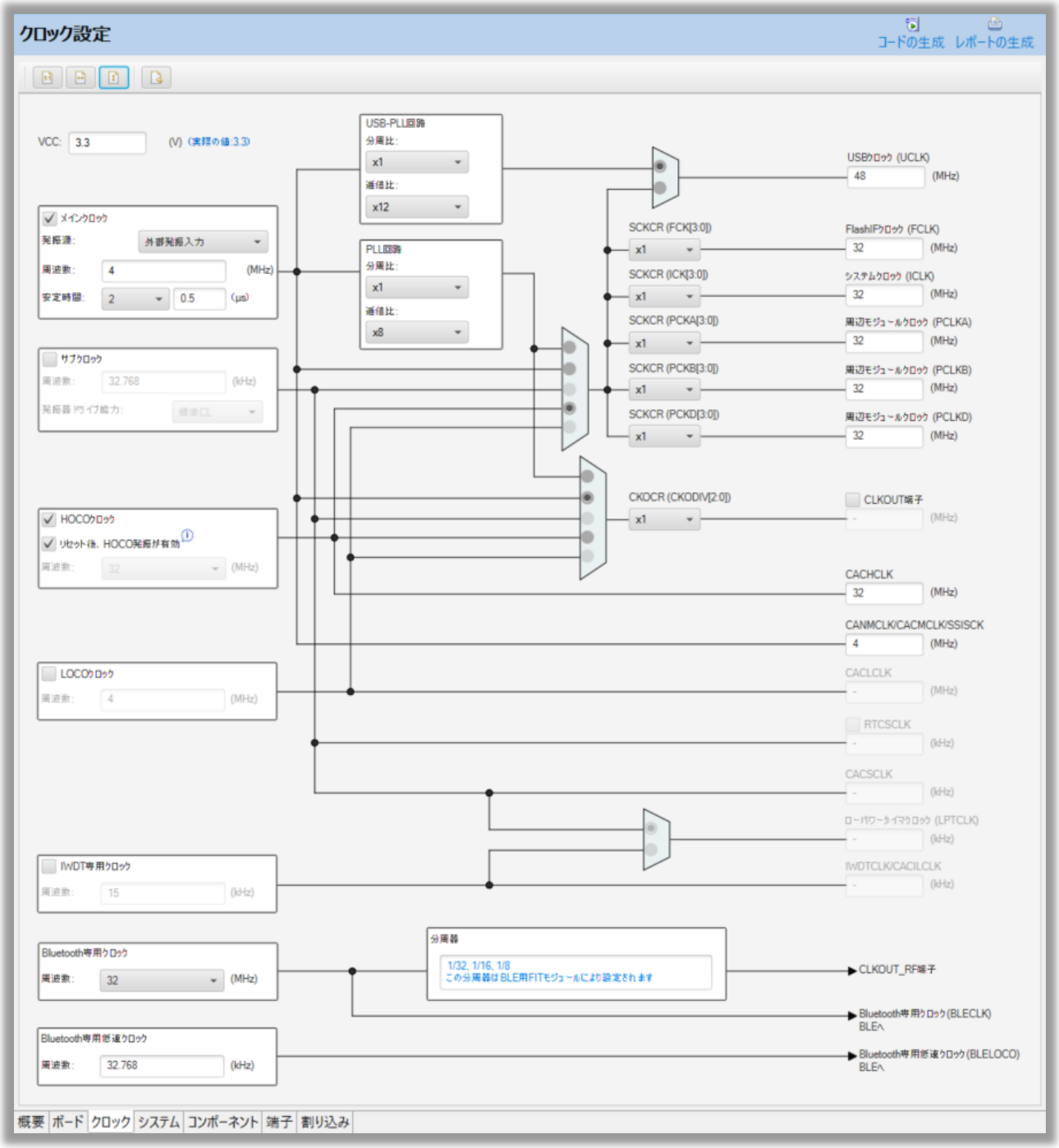

図 3-2　サンプルプログラムのクロック設定

3.3　動作確認条件

サンプルプログラムは表 3-9 に示す動作条件で確認済みです。

表 3-9　動作確認条件

項目	条件
MCU	RX23W (F523W8ADBL) ROM サイズ：512Kbytes、RAM サイズ：64Kbytes、ピン数：85
動作周波数	HOCO クロック：リセット後、HOCO 発振が有効 メインクロック：CLKOUT_RF 端子から出力して EXTAL 端子から入力した 4MHz クロックを使用 システムクロック(ICLK) ：32MHz 周辺モジュールクロック(PCLKA、PCLKB、PCLKD) ：32MHz USB クロック(UCLK)：48MHz フラッシュ IF クロック(FCLK) ：32MHz Bluetooth 専用クロック(BLECLK)：32MHz Bluetooth 専用低速クロック(BLELOCO)：32.768kHz
評価ボード	Renesas Solution Starter Kit for RX23W (RTK5523W8AC00001BJ) ジャンパ設定：電源供給 → USBCN0(FT234_5V) J3：1-2 Short J4：1-2 Short J5：1-2 Short J10：Short J13：Short ジャンパ設定：P26 → USB0-VBUSEN J7：1-2 Short
エミュレータ	E2 エミュレータ Lite (RTE0T0002LKCE00000R)
統合開発環境	e^2 studio 64-bit 2021-10
コンパイラ	C/C++ Compiler for RX Family (CC-RX) V2.08.01
フラッシュ書き込みツール	Renesas Flash Programmer V3.09.00
Bluetooth Low Energy 対応開発支援ツール ※1	QE for BLE[RA,RE,RX] V1.4.0 (1.4.0.v20220120-803) QE for BLE[RA,RE,RX] Utility (1.4.0.v20220118-1104)
エンディアン	リトルエンディアン
FreeRTOS	Renesas FreeRTOS (kernel only) 10.4.3-rx-1.0.1
ターミナルソフト	Tera Term Version 4.106
シリアルポート設定	ボーレート：115,200 bps データ：8 bits パリティ：なし ストップ：1 bit フロー制御：なし

※1:　Bluetooth Low Energy 対応開発支援ツール QE for BLE[RA,RE,RX]のインストール方法は下記のウェブサイトを参照してください。

各種アプリケーション対応開発支援ツール QE ユーザ向け情報
https://www.renesas.com/software-tool/qe-support

QE for BLE[RA,RE,RX]のプラグインである QE for BLE[RA,RE,RX] Utility の更新方法は下記のウェブサイトを参照してください。

Bluetooth® Low Energy 対応開発支援ツール QE for BLE ユーザ向け情報
https://www.renesas.com/software-tool/qe-ble-development-assistance-tool-bluetooth-low-energy-information-users

3.4　構成と動作

図 3-3 にサンプルプログラムのソフトウェア構成を示します。サンプルプログラムは RX23W の BLE モジュールと USB モジュールを制御するために必要な FIT モジュールや QE for BLE の生成コードに加えて、BLE 制御アプリケーション、USB 制御アプリケーション、デモアプリケーションを含みます。

BLE 制御アプリケーションは、Bluetooth LE 通信を容易に実行できる R_BLECTRL インタフェースを上位のアプリに提供します。なお BLE 制御アプリケーションは QE for BLE の生成するプロファイルフレームワークを利用します。R_BLECTRL インタフェースの詳細は 3.6 節を参照してください。

USB 制御アプリケーションは、USB 通信を容易に利用できる R_USBCTRL インタフェースを上位のアプリに提供します。R_USBCTRL インタフェースの詳細は 3.7 節を参照してください。

デモアプリケーションは R_BLECTRL インタフェースと R_USBCTRL インタフェースを利用して、Bluetooth LE 通信と USB 通信に加えて、MCU の低消費電力状態の制御を実行します。

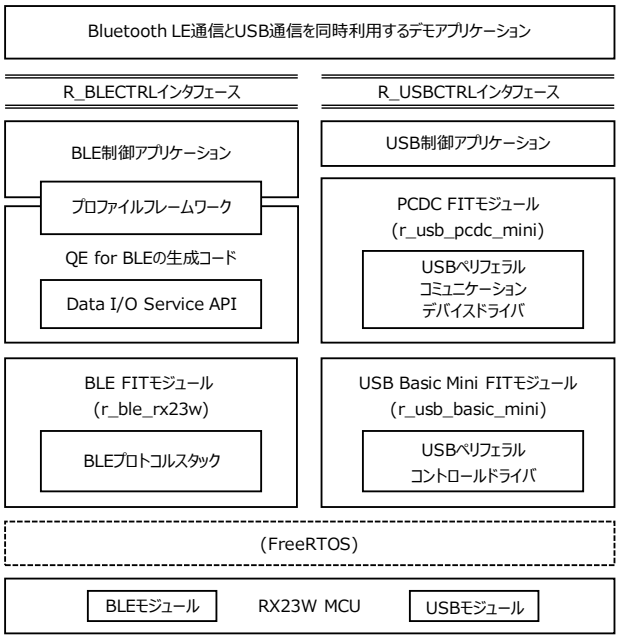

図 3-3　サンプルプログラムのソフトウェア構成

サンプルプログラムでは以下の独自サービスを定義します。2 台の評価ボードは Bluetooth LE 通信で本サービスを利用して任意のデータを送受信します。Central デバイスとして動作時は Data In キャラクタリスティックに Write Without Response でデータを送信します。Peripheral デバイスとして動作時は Data Out キャラクタリスティックから Notify でデータを送信します。

表 3-10　データ I/O サービス

項目	UUID	Properties Attribute Permission
Data I/O Service	7dbe3201-f5ab-498a-a012-676fef22f735	-
Data In Characteristic	7dbe3202-f5ab-498a-a012-676fef22f735	Write Without Response (max.244byte) No Authentication, No Authorization
Data Out Characteristic	7dbe3203-f5ab-498a-a012-676fef22f735	Notify (max.244byte)
Client Characteristic Configuration Descriptor	0x2902	Read, Write No Authentication, No Authorization

図 3-4 に RSSK for RX23W に搭載された LED、表 3-11 にサンプルプログラムの LED 制御を示します。サンプルプログラムが書き込まれた評価ボードに電源供給を開始すると、BLE 制御アプリケーションは Bluetooth LE の接続動作を自動で開始します。対向評価ボードとの接続が確立し、データ I/O サービスによるデータ通信が可能な状態となると、LED0 が点灯します。また Bluetooth LE 接続が切断されると、LED0 は消灯します。

PC と USB ケーブルで接続すると、USB 制御アプリケーションは Configured ステートまでの遷移処理を自動で実行します。さらに PC 上のターミナルソフトが COM ポートをオープンして、CDC でデータ通信が可能な状態となると、LED1 が点灯します。また PC との USB 接続が解除されると、LED1 は消灯します。

Bluetooth LE と USB の接続順序に制約はありません。Bluetooth LE の接続後に USB を接続、および USB の接続後に Bluetooth LE を接続することができます。

表 3-12 にサンプルプログラムの MCU 低消費電力状態の制御を示します。MCU 低消費電力状態のいずれかのモードに遷移すると LED2 が点灯します。

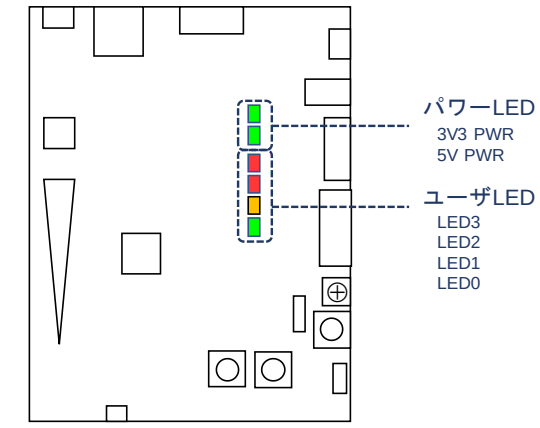

図 3-4　RSSK for RX23W に搭載された LED

表 3-11　サンプルプログラムの LED 制御

ユーザ LED	制御タイミング	概要
LED0 (緑)	点灯：BLECTRL_EVT_SERVICE_ENABLED 消灯：BLECTRL_EVT_DISCONNECTED	Bluetooth LE 通信のデータ I/O サービスでデータ通信可能であることを示す
LED1 (橙)	点灯：USBCTRL_EVT_CDC_ENABLED 消灯：USBCTRL_EVT_DETACH	USB 通信の CDC でデータ通信可能であることを示す
LED2 (赤)	点灯：R_BLE_LPC_EnterLowPowerMode()の直前 消灯：R_BLE_LPC_EnterLowPowerMode()の直後	MCU 低消費電力状態のいずれかのモードであること示す
LED3 (赤)	-	未使用

表 3-12　サンプルプログラムの MCU 低消費電力状態の制御

BLE 状態	USB 状態	MCU 低消費電力状態の制御
切断/接続	Detach	OS 無し：ソフトウェアスタンバイモードに遷移 FreeRTOS 利用時：ディープスリープモードに遷移
	Suspend	スリープモードに遷移
	上記以外	通常動作モード (MCU 低消費電力状態には遷移しない)

表 3-13 にデモアプリケーションのデータ送受信処理、図 3-5 にサンプルプログラムのデータ送受信動作を示します。BLE 状態と USB 状態がともに切断中の場合、デモアプリケーションはデータの送受信処理を実行しません。BLE 状態または USB 状態が接続中ならば、データの送受信時にコンソールへログを出力します。さらに BLE 状態と USB 状態がともに接続中ならば、USB 通信で受信したデータを Bluetooth LE 通信で送信し、Bluetooth LE 通信で受信したデータを USB 通信で送信します。

表 3-13　デモアプリケーションのデータ送受信処理

BLE 状態	USB 状態	デモ動作
切断	切断	・何もしない
接続		・Bluetooth LE 通信でデータの受信時、コンソールにログを出力
切断	接続	・USB 通信でデータの受信時、コンソールにログを出力
接続		・USB 通信でデータの送受信時、コンソールにログを出力 ・Bluetooth LE 通信でデータの受信時、コンソールにログを出力 ・USB 通信で受信したデータを Bluetooth LE 通信で送信 ・Bluetooth LE 通信で受信したデータを USB 通信で送信

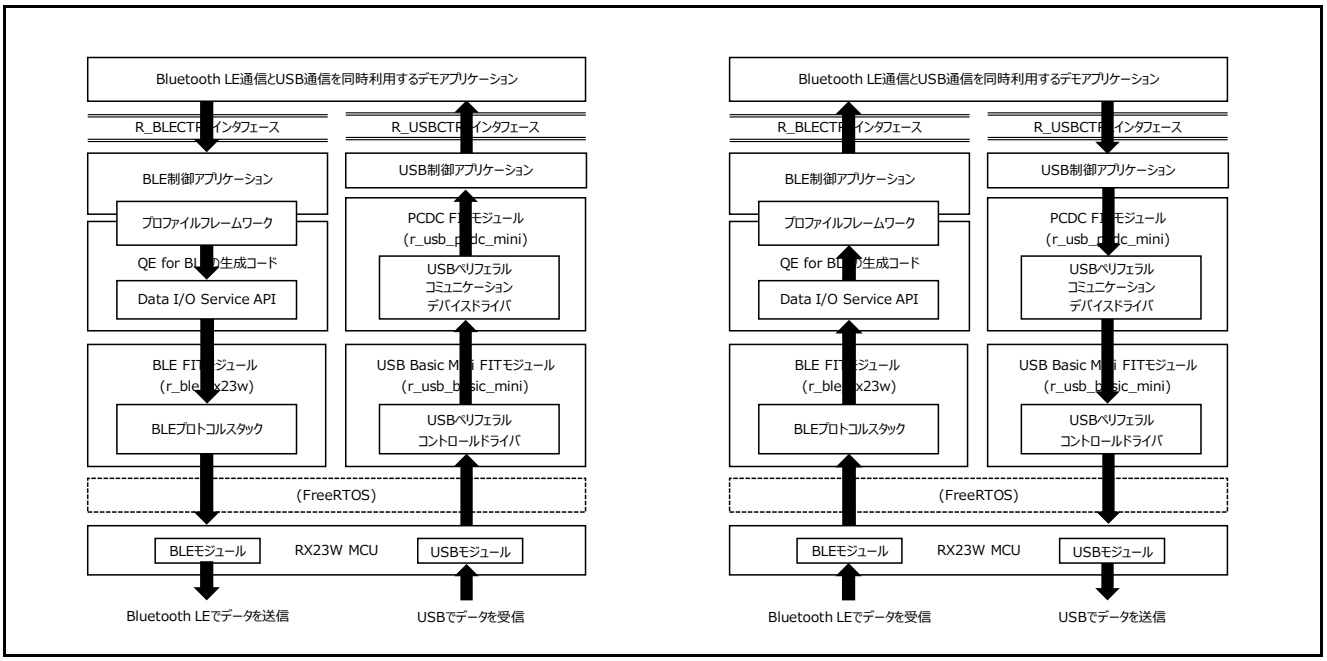

図 3-5　サンプルプログラムのデータ送受信 (左：USB→Bluetooth LE、右：Bluetooth LE→USB)

3.5　使用方法

3.5.1　e² studio へのプロジェクトインポートとビルド

1. e² studio を起動し、任意のフォルダをワークスペースとして指定します。

図 3-6　ワークスペースの指定

2. メニューの[ファイル]→[インポート...]を選択して、インポートダイアログを表示します。

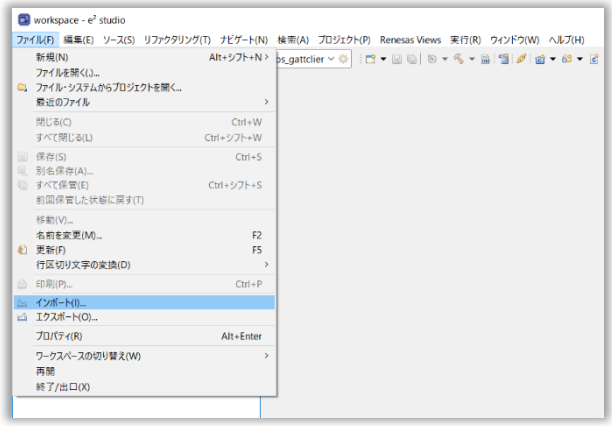

図 3-7　[ファイル]→[インポート...]

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]ボタンをクリックします。

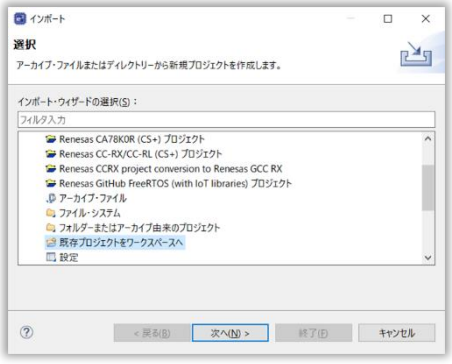

図 3-8　[一般]→[既存プロジェクトをワークスペースへ]

4. [アーカイブ・ファイルの選択]でサンプルプログラムのいずれかのプロジェクトファイル(.zip)を選択し、[終了]ボタンをクリックします。

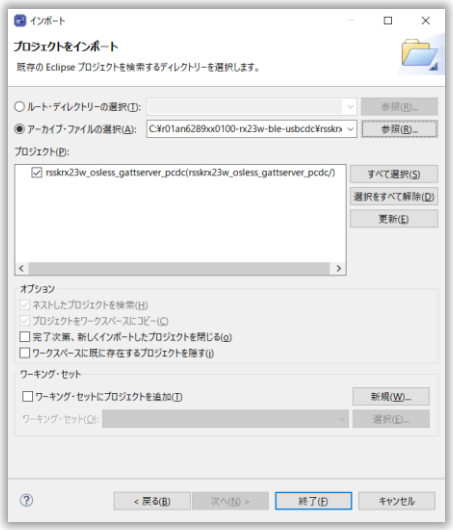

図 3-9　アーカイブ・ファイルの選択

5. 手順 1～4 を繰り返して、サンプルプログラムの各プロジェクトをインポートします。

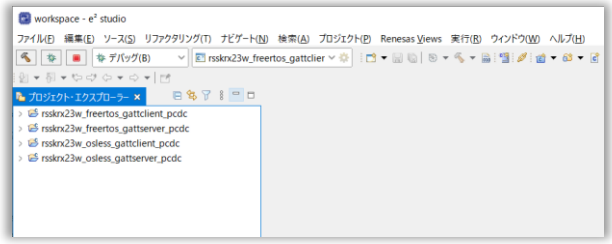

図 3-10　サンプルプログラムのプロジェクトインポート完了

6. 手順 5 の完了後、メニューの[プロジェクト]→[すべてビルド]を選択します。コンソールに"Build complete"と表示されればビルド成功です。ビルドしたプログラムファイル(.mot)は、各プロジェクトフォルダの Hardware Debug フォルダに格納されます。

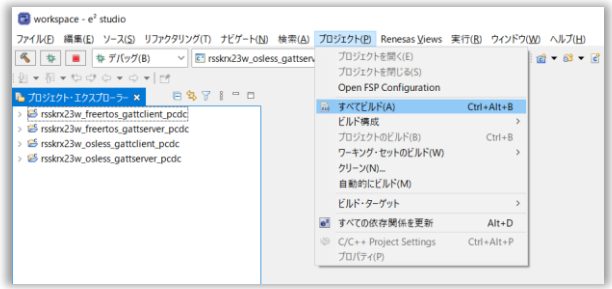

図 3-11　[プロジェクト]→[すべてビルド]

コンソールに"No toolchain set or toolchain not integrated"と表示されてビルドが失敗する場合、各プロジェクトの右クリックメニューで[プロパティ]を選択してプロパティダイアログを開き、[C/C++ビルド]→[設定]→[Toolchain]タブに移動して、ツールチェーンを選択してください。

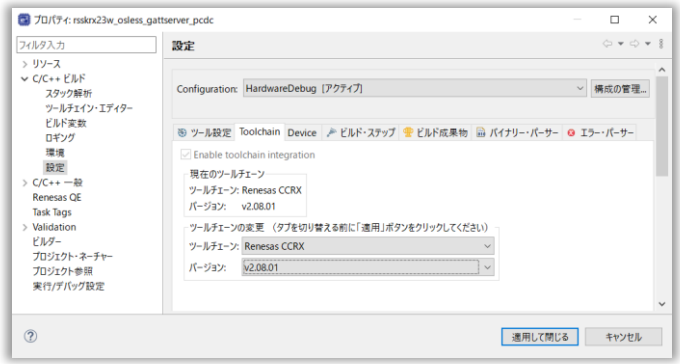

図 3-12　ツールチェーンの選択

3.5.2　評価ボードへのプログラムファイル書き込み

1. 評価ボードとエミュレータを接続し、エミュレータを PC と接続します。また評価ボードの USBCN0 と PC をケーブルで接続します。評価ボードへの電源は USBCN0 から供給されます。

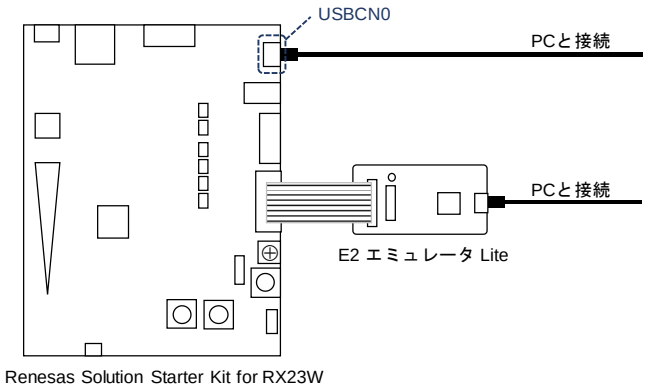

図 3-13　プログラムファイル書き込み時の評価ボード接続

2. Renesas Flash Programmer を起動し、メニューで[ファイル]→[新しいプロジェクトを作成...]を選択します。新しいプロジェクトの作成ダイアログで、任意のプロジェクト名を入力し、ツールを選択、インタフェースを FINE に設定して、[接続]ボタンをクリックします。"操作が成功しました"と表示されることを確認します。

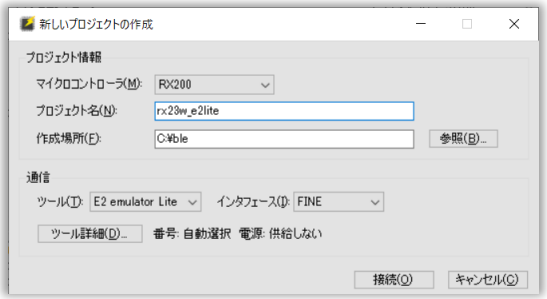

図 3-14　新しいプロジェクトの作成

3. e² studio でビルドしたプログラムファイル(.mot)を指定し、[スタート]ボタンをクリックします。"操作が成功しました"と表示されれば、プログラムファイルの書き込みは完了です。

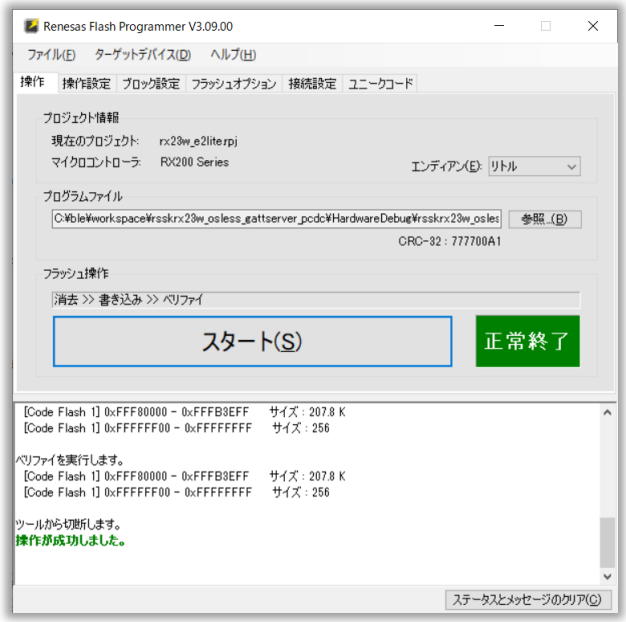

図 3-15　プログラムファイルの書き込み

4. 手順 1 と 3 を繰り返して、2 台の評価ボードに下記のいずれかの組合せでプログラムファイルを書き込みます。

表 3-14　サンプルプログラム書き込みの組合せ

組合せ	評価ボード (Central 動作)	評価ボード (Peripheral 動作)
(1)	rsskrx23w_osless_gattclient_pcdc.mot	rsskrx23w_osless_gattserver_pcdc.mot
(2)	rsskrx23w_freertos_gattclient_pcdc.mot	rsskrx23w_freertos_gattserver_pcdc.mot
(3)	rsskrx23w_osless_gattclient_pcdc.mot	rsskrx23w_freertos_gattserver_pcdc.mot
(4)	rsskrx23w_freertos_gattclient_pcdc.mot	rsskrx23w_osless_gattserver_pcdc.mot

3.5.3　Bluetooth LE 通信と USB 通信の評価

1. 評価ボードの USBCN0 と PC をケーブルで接続します。またボードの USB0_2 と PC をケーブルで接続します。評価ボードへの電源は USBCN0 から供給されます。
   上記の手順で 2 台の評価ボードを PC に接続します。

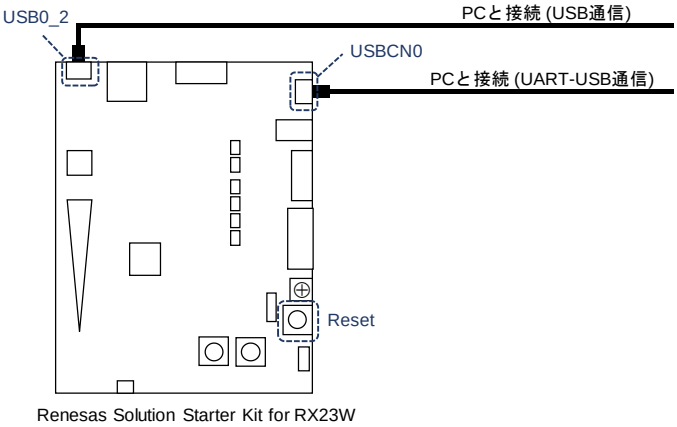

図 3-16　通信評価時の評価ボードの接続

2. 評価ボードを PC に USB ケーブルで接続すると、各 USB 接続は COM ポートとして認識されます。ターミナルソフトを起動して、COM ポートを開きます。ターミナルソフトのシリアルポート設定は表 3-9 を参照してください。

   図 3-17 は 2 台の評価ボードを PC に接続後、各 COM ポートをターミナルソフトで開いた場合の表示例です。

   左上：rsskrx23w_osless_gattserver_pcdc、UART-USB 通信の動作ログ

   左下：rsskrx23w_osless_gattserver_pcdc、USB 通信の受信データ

   右上：rsskrx23w_osless_gattclient_pcdc、UART-USB 通信の動作ログ

   右下：rsskrx23w_osless_gattclient_pcdc、USB 通信の受信データ

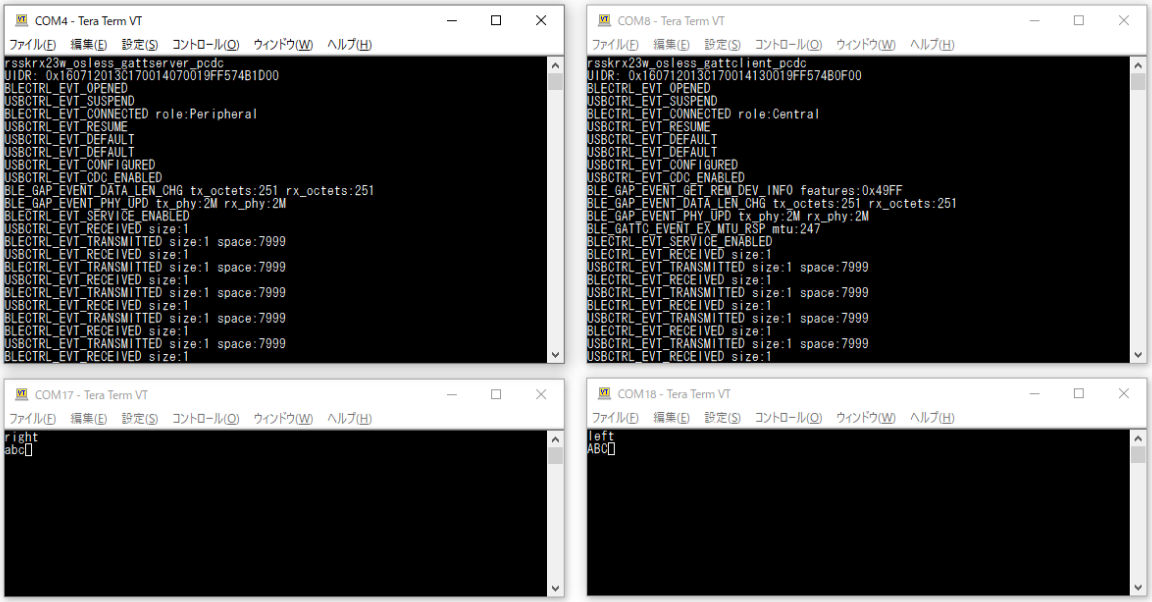

図 3-17　ターミナルソフトの表示例

3. 評価ボードのリセットボタンを押下するとプログラムが再起動し、UART-USB 通信でコンソールに動作ログが出力されます。動作ログの例は後述の図 3-18 を参照してください。

4. 動作ログで BLECTRL_EVT_SERVICE_ENABLED と USBCTRL_EVT_CDC_ENABLED の表示を確認します。一方の評価ボードの USB 通信のターミナルソフトで文字列を入力すると、他方の評価ボードの USB 通信のターミナルソフトに文字列が表示されます。

   図 3-17 に示すターミナルソフトの表示例では、左下のターミナルソフトに"left"と入力することで、右下のターミナルソフトに"left"と表示されました。右下のターミナルソフトに"right"と入力することで、左下のターミナルソフトに"right"と表示されました。

下記は rsskrx23w_osless_gattserver_pcdc を書き込んだ評価ボードの動作ログの表示例です。

```
 1  rsskrx23w_osless_gattserver_pcdc
 2  UIDR: 0x160712013C170014070019FF574B1D00
 3  BLECTRL_EVT_OPENED
 4  USBCTRL_EVT_SUSPEND
 5  USBCTRL_EVT_RESUME
 6  USBCTRL_EVT_DEFAULT
 7  USBCTRL_EVT_DEFAULT
 8  USBCTRL_EVT_CONFIGURED
 9  USBCTRL_EVT_CDC_ENABLED
10  BLECTRL_EVT_CONNECTED role:Peripheral
11  BLE_GAP_EVENT_DATA_LEN_CHG tx_octets:251 rx_octets:251
12  BLE_GAP_EVENT_PHY_UPD tx_phy:2M rx_phy:2M
13  BLECTRL_EVT_SERVICE_ENABLED
14  USBCTRL_EVT_RECEIVED size:1
15  BLECTRL_EVT_TRANSMITTED size:1 space:7999
16  USBCTRL_EVT_RECEIVED size:1
17  BLECTRL_EVT_TRANSMITTED size:1 space:7999
18  BLECTRL_EVT_RECEIVED size:1
19  USBCTRL_EVT_TRANSMITTED size:1 space:7999
20  BLECTRL_EVT_RECEIVED size:1
21  USBCTRL_EVT_TRANSMITTED size:1 space:7999
```

図 3-18　サンプルプログラムの動作ログ例

1 行目 プログラムファイル名

2 行目 UIDR：RX23W を識別するための 16 バイト長のユニーク ID、USB の iSerialNumber として使用

Device ID、Product ID、iSerialNumber を含む"デバイスインスタンスパス"は Windows OS のデバイスマネージャーの[ポート(COM と LPT)]→[USB Serial Bus (COM **)]のプロパティで確認可能

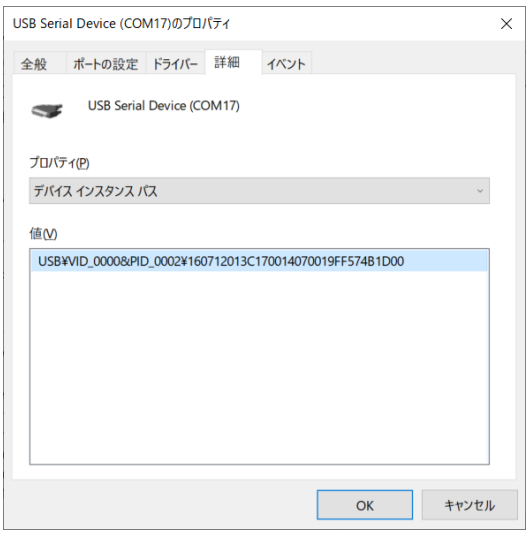

図 3-19　デバイスインスタンスパスの表示例

8 行目 USBCTRL_EVT_CONFIGURED：USB の Configured ステートへの遷移完了

9 行目 USBCTRL_EVT_CDC_ENABLED：COM ポートのオープン完了、CDC でのデータ通信可能

10 行目 BLECTRL_EVT_CONNECTED：Bluetooth LE 接続の確立完了

13 行目 BLECTRL_EVT_SERVICE_ENABLED：データ I/O サービスの有効化完了、データ通信可能

14 行目 USBCTRL_EVT_RECEIVED：USB でのデータ受信完了

15 行目 BLECTRL_EVT_TRANSMITTED：Bluetooth LE でのデータ送信完了

18 行目 BLECTRL_EVT_RECEIVED：Bluetooth LE でのデータ送信完了

19 行目 USBCTRL_EVT_TRANSMITTED：USB でのデータ受信完了

3.6　R_BLECTRL インタフェース

BLE 制御アプリケーションが提供する R_BLECTRL インタフェースの仕様を示します。

3.6.1　R_BLECTRL_Open

R_BLECTRL_Open

BLE 制御処理を初期化

ヘッダ	r_blectrl_if.h	
宣　言	e_blectrl_status_t R_BLECTRL_Open(void);	
引　数	なし	
返り値	BLECTRL_SUCCESS	成功
	BLECTRL_ERR_STATE	失敗：初期化完了状態
	BLECTRL_ERR_API	失敗：BLE プロトコルスタックの初期化に失敗
補　足	BLE 制御処理の初期化完了は BLECTRL_EVT_OPENED イベントで通知される。 CLKOUT_RF 端子のクロック出力を有効化した場合、本関数の実行でクロック出力が開始する。 CLKOUT_RF 端子の出力クロックを USB クロック(UCLK)の基準クロックとする場合、USB モジュールの動作開始前に本関数を実行すること。	

3.6.2　R_BLECTRL_Close

R_BLECTRL_Close

BLE 制御処理を終了

ヘッダ	r_blectrl_if.h	
宣　言	e_blectrl_status_t R_BLECTRL_Close(void);	
引　数	なし	
返り値	BLECTRL_SUCCESS	成功
	BLECTRL_ERR_STATE	失敗：未初期化状態または終了状態
	BLECTRL_ERR_API	失敗：BLE プロトコルスタックの終了に失敗
補　足	本関数を実行すると、CLKOUT_RF 端子の出力クロックが停止する。 CLKOUT_RF 端子の出力クロックをメインクロックとして使用する場合、USB モジュールの動作が停止するため、本関数は実行しないこと。	

3.6.3　R_BLECTRL_RegisterCb

R_BLECTRL_RegisterCb

BLE 制御イベントコールバック関数を登録

ヘッダ	r_blectrl_if.h		
宣　言	e_blectrl_status_t R_BLECTRL_RegisterCb(blectrl_evt_cb_t cb);		
引　数	(IN)	cb	BLE 制御イベントコールバック関数
返り値	BLECTRL_SUCCESS		成功
	BLECTRL_ERR_ARG		失敗：引数 cb が NULL
	BLECTRL_ERR_STATE		失敗：未初期化状態または通信動作状態
補　足	R_BLECTRL_Open()の実行後、R_BLECTRL_StartPeripheral()と R_BLECTRL_StartCentral()の実行前に本関数を実行すること。		

3.6.4　R_BLECTRL_RegisterTxBuffer

R_BLECTRL_RegisterTxBuffer

BLE 送信バッファを登録

ヘッダ	r_blectrl_if.h		
宣　言	e_blectrl_status_t R_BLECTRL_RegisterTxBuffer(uint8_t *buffer, uint16_t size);		
引　数	(IN)	buffer	送信バッファ
	(IN)	size	送信バッファサイズ
返り値	BLECTRL_SUCCESS		成功
	BLECTRL_ERR_ARG		失敗：引数 buffer が NULL または引数 buffer が 0
	BLECTRL_ERR_STATE		失敗：未初期化状態または通信動作状態
補　足	R_BLECTRL_Open()の実行後、R_BLECTRL_StartPeripheral()と R_BLECTRL_StartCentral()の実行前に本関数を実行すること。		

3.6.5　R_BLECTRL_StartPeripheral

R_BLECTRL_StartPeripheral

Peripheral デバイスとして接続を開始

ヘッダ	r_blectrl_if.h	
宣　言	e_blectrl_status_t R_BLECTRL_StartPeripheral(void);	
引　数	なし	
返り値	BLECTRL_SUCCESS	成功
	BLECTRL_ERR_UNSUPPORT	失敗：Peripheral 動作に未対応
	BLECTRL_ERR_STATE	失敗：未初期化状態
	BLECTRL_ERR_API	失敗：Advertising の開始に失敗
補　足	Central デバイスとの接続完了は BLECTRL_EVT_CONNECTED イベントで通知される。 接続完了後、GATT サービスの有効化完了は BLECTRL_EVT_SERVICE_ENABLED イベントで通知される GATT サービスの有効化完了後、データ受信は BLECTRL_EVT_RECEIVED イベントで通知される。 Central デバイスとの接続の切断は BLECTRL_EVT_DISCONNECTED イベントで通知される。	
制限事項	1.　同時に接続できる Central デバイスは 1 台のみ。	

3.6.6　R_BLECTRL_StartCentral

R_BLECTRL_StartCentral		
Central デバイスとして接続を開始		
ヘッダ	r_blectrl_if.h	
宣　言	`e_blectrl_status_t R_BLECTRL_StartCentral(void);`	
引　数	なし	
返り値	`BLECTRL_SUCCESS`	成功
	`BLECTRL_ERR_UNSUPPORT`	失敗：Central 動作に未対応
	`BLECTRL_ERR_STATE`	失敗：未初期化状態
	`BLECTRL_ERR_API`	失敗：Scan の開始に失敗
補　足	Peripheral デバイスとの接続の確立は `BLECTRL_EVT_CONNECTED` イベントで通知される。 接続完了後、GATT サービスの有効化は `BLECTRL_EVT_SERVICE_ENABLED` イベントで通知される。 GATT サービスの有効化完了後、データ受信は `BLECTRL_EVT_RECEIVED` イベントで通知される。 Peripheral デバイスとの接続の切断は `BLECTRL_EVT_DISCONNECTED` イベントで通知される。	
制限事項	1.　同時に接続できる Peripheral デバイスは 1 台のみ。	

3.6.7　R_BLECTRL_Disconnect

R_BLECTRL_Disconnect		
対向デバイスとの接続を切断		
ヘッダ	r_blectrl_if.h	
宣　言	`e_blectrl_status_t R_BLECTRL_Disconnect(uint16_t conn_hdl);`	
引　数	(IN)　`conn_hdl`	コネクションハンドル
返り値	`BLECTRL_SUCCESS`	成功
	`BLECTRL_ERR_STATE`	失敗：対向デバイスと未接続
	`BLECTRL_ERR_API`	失敗：切断要求に失敗
補　足	対向デバイスとの接続の切断完了は `BLECTRL_EVT_DISCONNECTED` イベントで通知される。	

3.6.8　R_BLECTRL_GetTxBufferSpace

R_BLECTRL_GetTxBufferSpace		
BLE 送信バッファの空サイズを取得		
ヘッダ	r_blectrl_if.h	
宣　言	`uint16_t R_BLECTRL_GetTxBufferSpace(uint16_t conn_hdl);`	
引　数	(IN)　`conn_hdl`	コネクションハンドル
返り値	送信バッファの空サイズ[byte]	
補　足	なし	

3.6.9 R_BLECTRL_TransmitData

R_BLECTRL_TransmitData			
接続済みの対向デバイスにデータを送信			
ヘッダ	r_blectrl_if.h		
宣　言	`uint16_t R_BLECTRL_TransmitData(uint16_t conn_hdl, const uint8_t *data, uint16_t size);`		
引　数	(IN)	`conn_hdl`	コネクションハンドル
	(IN)	`data`	送信データ
	(IN)	`size`	送信データサイズ[byte]
返り値	送信バッファに格納されたデータのサイズ[byte]		
補　足	本関数は `BLECTRL_EVT_SERVICE_ENABLED` イベントの通知後に実行すること。 Central デバイスとして動作時、Data I/O サービスの Data In キャラクタリスティックに Write Without Response でデータを送信する。 Peripheral デバイスとして動作時、Data I/O サービスの Data Out キャラクタリスティックから Notify でデータを送信する。 データ送信は `BLECTRL_EVT_TRANSMITTED` イベントで通知される。		
制限事項	FreeRTOS 利用時 1. 本関数は `xTaskNotifyGive()`を使用するため、割り込みハンドラからコールしないこと。 2. 本関数の送信バッファ操作はクリティカルセクションとしてガードされていないため、複数タスクからコールしないこと。		

3.6.10 blectrl_main

blectrl_main			
BLE 制御アプリのメインコンテキスト			
ヘッダ	r_blectrl_if.h		
宣　言	`void blectrl_main(void);`		
引　数	なし		
返り値	なし		
補　足	OS を使用しない場合、メインループで本関数をコールすること。 FreeRTOS を利用する場合、本関数を実行するためのタスクを生成し、タスク内のループでコールすること。		

3.6.1 blectrl_evt_cb_t

blectrl_evt_cb_t			
BLE 制御イベントコールバック関数			
ヘッダ	r_blectrl_if.h		
宣　言	`typedef void (*blectrl_evt_cb_t)(e_blectrl_evt_t type, st_blectrl_evt_param_t *param);`		
引　数	(OUT)	`type`	BLE 制御イベント
	(OUT)	`param`	BLE 制御イベントパラメータ
返り値	なし		
補　足	BLE 制御イベントコールバック関数は上位アプリで実装し、`R_BLECTRL_RegisterCb()`で登録すること		

3.6.2　e_blectrl_status_t

e_blectrl_status_t

R_BLECTRL インタフェース関数の実行結果

ヘッダ　　r_blectrl_if.h

宣　言

```
typedef enum e_blectrl_status
{
    BLECTRL_SUCCESS = 0,
    BLECTRL_ERR_UNSUPPORT,      /* Unsupported API */
    BLECTRL_ERR_ARG,           /* Invalid Argument */
    BLECTRL_ERR_STATE,         /* Invalid State */
    BLECTRL_ERR_API,           /* RBLE API Error */
} e_blectrl_status_t;
```

3.6.3　e_blectrl_evt_t

e_blectrl_evt_t

BLE 制御イベント

ヘッダ　　r_blectrl_if.h

宣　言

```
typedef enum e_blectrl_evt
{
    BLECTRL_EVT_NONE = 0,
    BLECTRL_EVT_OPENED,
    BLECTRL_EVT_CONNECTED,
    BLECTRL_EVT_DISCONNECTED,
    BLECTRL_EVT_SERVICE_ENABLED,
    BLECTRL_EVT_TRANSMITTED,
    BLECTRL_EVT_RECEIVED
} e_blectrl_evt_t;
```

補　足　　BLE 制御イベントは R_BLECTRL_RegisterCb()に登録したコールバック関数で通知される。

BLE 制御イベントの通知タイミングは 3.7.15 項の状態遷移図を参照。

3.6.4　st_blectrl_evt_param_t

st_blectrl_evt_param_t

BLE 制御イベントパラメータ

ヘッダ　　r_blectrl_if.h

宣　言

```
typedef struct st_blectrl_evt_conn
{
    /* role - 0x00: Central, 0x01: Peripheral */
    uint8_t role;
    /* device address type - 0x00: Public, 0x01: Random */
    uint8_t remote_addr_type;
    /* device address */
    uint8_t remote_addr[6];
} st_blectrl_evt_conn_t;

typedef struct st_blectrl_evt_disconn
{
    /* reason for disconnection */
    uint8_t reason;
} st_blectrl_evt_disconn_t;

typedef struct st_blectrl_evt_tx
{
    /* bytes of the data size transmitted */
    uint16_t size;
} st_blectrl_evt_tx_t;

typedef struct st_blectrl_evt_rx
{
    /* pointer to the data received */
    uint8_t *data;
    /* bytes of the data size received */
    uint16_t size;
} st_blectrl_evt_rx_t;

/* Event Parameter */
typedef struct st_blectrl_evt_param
{
    uint16_t conn_hdl;          /* Connection Handle */
    union
    {
        /* BLECTRL_EVT_CONNECTED event parameter */
        st_blectrl_evt_conn_t    conn;
        /* BLECTRL_EVT_DISCONNECTED event parameter */
        st_blectrl_evt_disconn_t disconn;
        /* BLECTRL_EVT_TRANSMITTED event parameter */
        st_blectrl_evt_tx_t      tx;
        /* BLECTRL_EVT_RECEIVED event parameter */
        st_blectrl_evt_rx_t      rx;
    } evt;
} st_blectrl_evt_param_t;
```

補　足　　BLE 制御イベントパラメータは R_BLECTRL_RegisterCb()に登録したコールバック関数で通知される。

evt.conn は BLECTRL_EVT_CONNECTED 通知時に参照すること。

evt.disconn は BLECTRL_EVT_DISCONNECTED 通知時に参照すること。

evt.tx は BLECTRL_EVT_TRANSMITTED 通知時に参照すること。

evt.rx は BLECTRL_EVT_RECEIVED 通知時に参照すること。

3.7　R_USBCTRL インタフェース

USB 制御アプリケーションが提供する R_USBCTRL インタフェースの仕様を示します。

3.7.1　R_USBCTRL_Open

R_USBCTRL_Open

USB 制御処理を初期化

ヘッダ	r_usbctrl_if.h	
宣　言	e_usbctrl_status_t R_USBCTRL_Open(void);	
引　数	なし	
返り値	USBCTRL_SUCCESS	成功
	USBCTRL_ERR_API	失敗：USB ドライバの初期化に失敗
補　足	USB の挿入後、Configured ステートへの遷移は USBCTRL_EVT_CONFIGURED イベントで通知される。 コミュニケーションデバイスクラスの有効化は USBCTRL_EVT_CDC_ENABLED イベントで通知される。 USB の抜去後、Powered ステートへの遷移は USBCTRL_EVT_DETACH イベントで通知される。 CLKOUT_RF 端子の出力クロックを USB クロック(UCLK)の基準クロックとする場合、本関数の実行前に R_BLECTRL_Open()を実行すること。	

3.7.2　R_USBCTRL_Close

R_USBCTRL_Close

USB 制御処理を終了

ヘッダ	r_usbctrl_if.h	
宣　言	e_usbctrl_status_t R_USBCTRL_Close(void);	
引　数	なし	
返り値	USBCTRL_SUCCESS	成功
	USBCTRL_ERR_API	失敗：USB ドライバの終了に失敗
補　足	なし	

3.7.3　R_USBCTRL_RegisterCb

R_USBCTRL_RegisterCb

USB 制御イベントコールバック関数を登録

ヘッダ	r_usbctrl_if.h		
宣　言	e_usbctrl_status_t R_USBCTRL_RegisterCb(usbctrl_evt_cb_t cb);		
引　数	(IN)	cb	USB 制御イベントコールバック関数
返り値	USBCTRL_SUCCESS		成功
	USBCTRL_ERR_ARG		失敗：引数 cb が NULL
補　足	なし		

3.7.4　R_USBCTRL_RegisterTxBuffer

R_USBCTRL_RegisterTxBuffer			
USB 送信バッファを登録			
ヘッダ	r_usbctrl_if.h		
宣　言	e_usbctrl_status_t R_USBCTRL_RegisterTxBuffer(uint8_t *buffer, uint16_t size);		
引　数	(IN)	buffer	送信バッファ
	(IN)	size	送信バッファサイズ
返り値	USBCTRL_SUCCESS		成功
	USBCTRL_ERR_ARG		失敗：引数 buffer が NULL または引数 buffer が 0
	USBCTRL_ERR_STATE		失敗：未初期化状態または通信動作状態
補　足	なし		

3.7.5　R_USBCTRL_SuspendRx

R_USBCTRL_SuspendRx		
USB 受信を休止		
ヘッダ	r_usbctrl_if.h	
宣　言	e_usbctrl_status_t R_USBCTRL_SuspendRx(void);	
引　数	なし	
返り値	USBCTRL_SUCCESS	成功
	USBCTRL_ERR_STATE	失敗：CDC が無効状態
補　足	本関数は USBCTRL_EVT_CDC_ENABLED イベントの通知後に実行すること。 R_USBCTRL_ResumeRx()を実行すると、USB 受信が再開される。 USB 受信の休止状態は、USB を抜去すると解除される。	

3.7.6　R_USBCTRL_ResumeRx

R_USBCTRL_ResumeRx		
USB 受信を再開		
ヘッダ	r_usbctrl_if.h	
宣　言	e_usbctrl_status_t R_USBCTRL_ResumeRx(void);	
引　数	なし	
返り値	USBCTRL_SUCCESS	成功
	USBCTRL_ERR_STATE	失敗：Communication Device Class(CDC)が無効状態
補　足	なし	

3.7.7　R_USBCTRL_GetTxBufferSpace

R_USBCTRL_GetTxBufferSpace	
USB 送信バッファの空サイズを取得	
ヘッダ	r_usbctrl_if.h
宣　言	uint16_t R_USBCTRL_GetTxBufferSpace(void);
引　数	なし
返り値	送信バッファの空サイズ[byte]
補　足	なし

3.7.8 R_USBCTRL_TransmitData

R_USBCTRL_TransmitData

Host デバイスにデータを送信

ヘッダ　r_usbctrl_if.h

宣　言　`uint16_t R_USBCTRL_TransmitData(const uint8_t *data, uint16_t size);`

引　数

(IN)	`data`	送信データ
(IN)	`size`	送信データサイズ[byte]

返り値　送信バッファに格納されたデータのサイズ[byte]

補　足　本関数は `USBCTRL_EVT_CDC_ENABLED` イベントの通知後に実行すること。
データ送信は `USBCTRL_EVT_TRANSMITTED` イベントで通知される。

制限事項　FreeRTOS 利用時

1. 本関数は `xTaskNotifyGive()`を使用するため、割り込みハンドラからコールしないこと。
2. 本関数の送信バッファ操作はクリティカルセクションとしてガードされていないため、複数タスクからコールしないこと。

3.7.9 R_USBCTRL_GetAllowedLpcMode

R_USBCTRL_GetAllowedLpcMode

遷移可能な低消費電力モードを取得

ヘッダ　r_usbctrl_if.h

宣　言　`lpc_low_power_mode_t R_USBCTRL_GetAllowedLpcMode(void);`

引　数　なし

返り値　遷移可能な低消費電力状態のモードを示す

返り値	低消費電力状態のモード		
	SLEEP	DEEP_SLEEP	SW_STANDBY
`LPC_LP_INVALID_MODE`	×	×	×
`LPC_LP_SLEEP`	○	×	×
`LPC_LP_DEEP_SLEEP`	本関数は返却しない		
`LPC_LP_SW_STANDBY`	○	○	○

補　足　なし

3.7.10 usbctrl_main

usbctrl_main

USB 制御アプリのメインコンテキスト

ヘッダ　r_usbctrl_if.h

宣　言　`void usbctrl_main(void);`

引　数　なし

返り値　なし

補　足　OS を使用しない場合、メインループで本関数をコールすること。
FreeRTOS を利用する場合、本関数を実行するためのタスクを生成し、タスク内のループでコールすること。

3.7.11 usbctrl_evt_cb_t

usbctrl_evt_cb_t

USB 制御イベントコールバック関数

ヘッダ　　r_usbctrl_if.h

宣　言
```
typedef void (*usbctrl_evt_cb_t)(e_usbctrl_evt_t type,
st_usbctrl_evt_param_t *param);
```

引　数
(OUT)　type　　USB 制御イベント
(OUT)　param　　USB 制御イベントパラメータ

返り値　　なし

補　足　　USB 制御イベントコールバック関数は上位アプリで実装し、R_USBCTRL_RegisterCb()で登録すること

3.7.12 e_usbctrl_status_t

e_usbctrl_status_t

R_USBCTRL インタフェース関数の実行結果

ヘッダ　　r_usbctrl_if.h

宣　言
```
typedef enum e_usbctrl_status
{
    USBCTRL_SUCCESS = 0,
    USBCTRL_ERR_UNSUPPORT,      /* Unsupported API */
    USBCTRL_ERR_ARG,           /* Invalid Argument */
    USBCTRL_ERR_STATE,         /* Invalid State */
    USBCTRL_ERR_API,           /* USB API Error */
} e_usbctrl_status_t;
```

3.7.13 e_usbctrl_evt_t

e_usbctrl_evt_t

BLE 制御イベント

ヘッダ　　r_usbctrl_if.h

宣　言
```
typedef enum e_usbctrl_evt
{
    USBCTRL_EVT_NONE = 0,
    USBCTRL_EVT_SUSPEND,
    USBCTRL_EVT_RESUME,
    USBCTRL_EVT_DETACH,
    USBCTRL_EVT_DEFAULT,
    USBCTRL_EVT_CONFIGURED,
    USBCTRL_EVT_CDC_ENABLED,
    USBCTRL_EVT_TRANSMITTED,
    USBCTRL_EVT_RECEIVED
} e_usbctrl_evt_t;
```

補　足　　USB 制御イベントは R_USBCTRL_RegisterCb()に登録したコールバック関数で通知される。

USB 制御イベントの通知タイミングは 3.7.16 項の状態遷移図を参照。

3.7.14 st_usbctrl_evt_param_t

st_usbctrl_evt_param_t

USB 制御イベントパラメータ

ヘッダ　　r_usbctrl_if.h

宣　言

```
typedef struct st_usbctrl_evt_cdc
{
    /* dte(data terminal equipment) rate in bps */
    uint32_t rate;
    /* data bits */
    uint8_t  bits;
} st_usbctrl_evt_cdc_t;

typedef struct st_usbctrl_evt_tx
{
    /* transmitted data size in bytes */
    uint16_t size;
} st_usbctrl_evt_tx_t;

typedef struct st_usbctrl_evt_rx
{
    /* pointer to the data received */
    uint8_t *data;
    /* bytes of the data size received */
    uint16_t size;
} st_usbctrl_evt_rx_t;

typedef struct st_usbctrl_evt_param
{
    union
    {
        /* USBCTRL_EVT_CDC_ENABLED event parameter */
        st_usbctrl_evt_cdc_t     cdc;
        /* USBCTRL_EVT_TRANSMITTED event parameter */
        st_usbctrl_evt_tx_t      tx;
        /* USBCTRL_EVT_RECEIVED event parameter */
        st_usbctrl_evt_rx_t      rx;
    } evt;
} st_usbctrl_evt_param_t;
```

補　足　　USB 制御イベントパラメータは R_USBCTRL_RegisterCb()に登録したコールバック関数で通知される。

evt.cdc は USBCTRL_EVT_CDC_ENABLED 通知時に参照すること。

evt.tx は USBCTRL_EVT_TRANSMITTED 通知時に参照すること。

evt.rx は USBCTRL_EVT_RECEIVED 通知時に参照すること。

3.7.15 BLE 制御処理の状態遷移

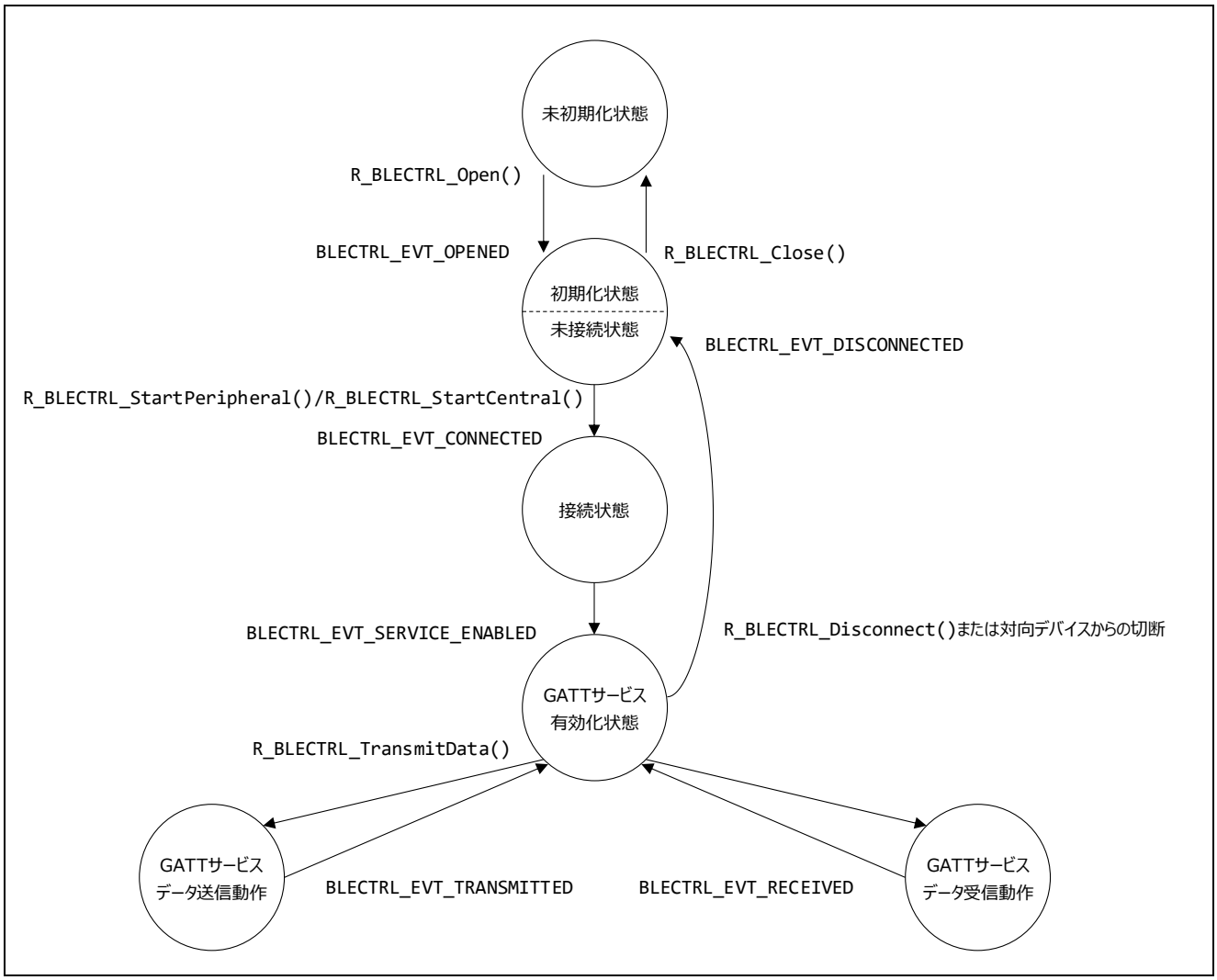

図 3-20　BLE 制御処理の状態遷移

3.7.16 USB 制御処理の状態遷移

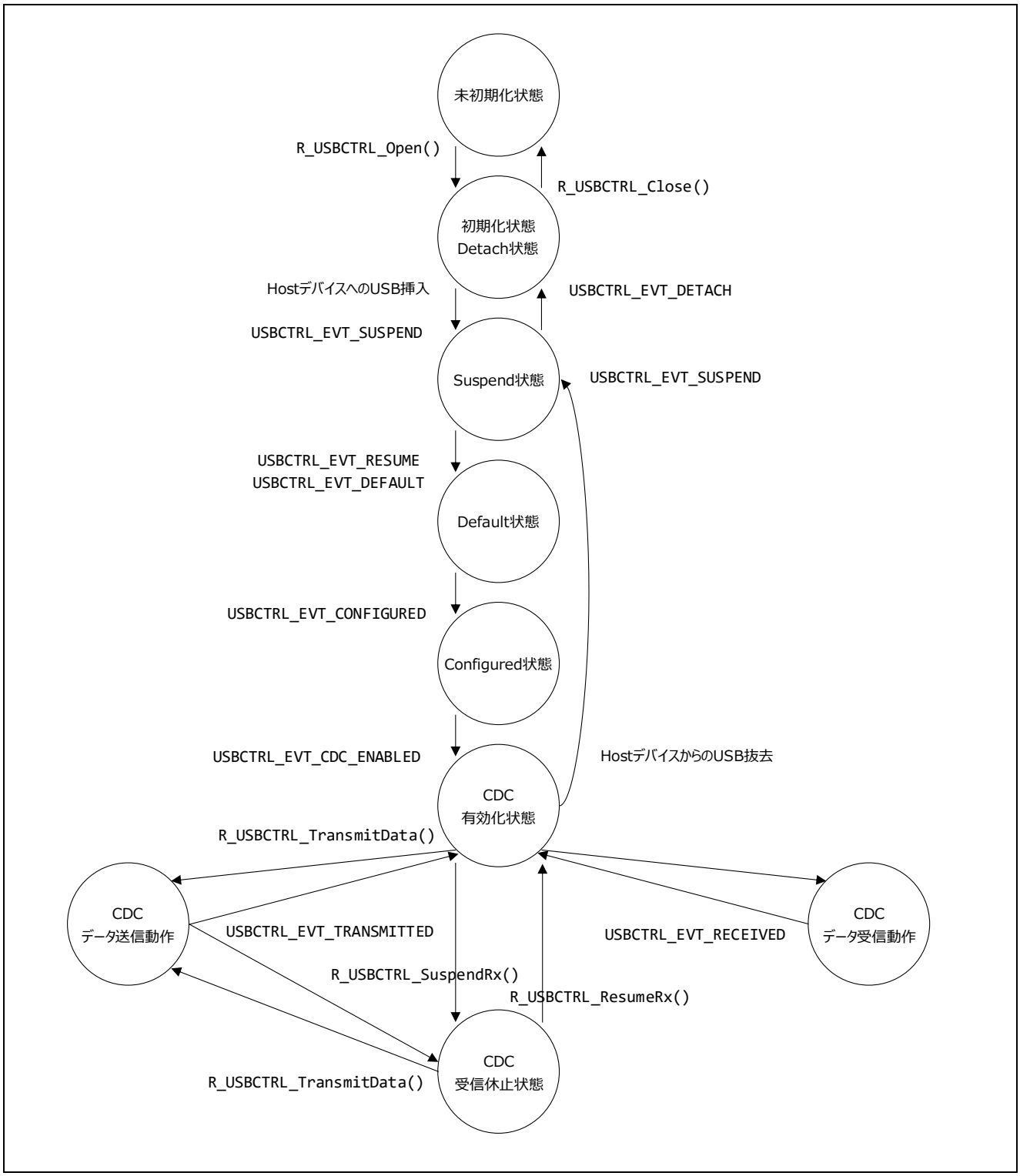

図 3-21　USB 制御処理の状態遷移

3.8　カスタマイズ方法と製品化時の注意点

3.8.1　BLE 制御アプリケーションのサービス構成の変更

本サンプルプログラムは QE for BLE[RA,RE,RX]を使用してデータ I/O サービスを生成しました。データ I/O サービスのデフォルト構成は表 3-10 を参照してください。QE for BLE を操作してソースコードを生成することで、データ I/O サービスの機能拡張や、新しいサービスを定義することができます。QE for BLE については下記のウェブサイトを参照してください。

Bluetooth® Low Energy 対応開発支援ツール　QE for BLE
https://www.renesas.com/qe-ble

QE for BLE[RA,RE,RX]を起動してプロファイル構成を更新する手順を以下に示します。

1. QE for BLE[RA,RE,RX]の起動は e² studio のメニューで[Renesas Views]→[Renesas QE]→[R_BLE カスタムプロファイル　RA,RE,RX (QE)]を選択してください。

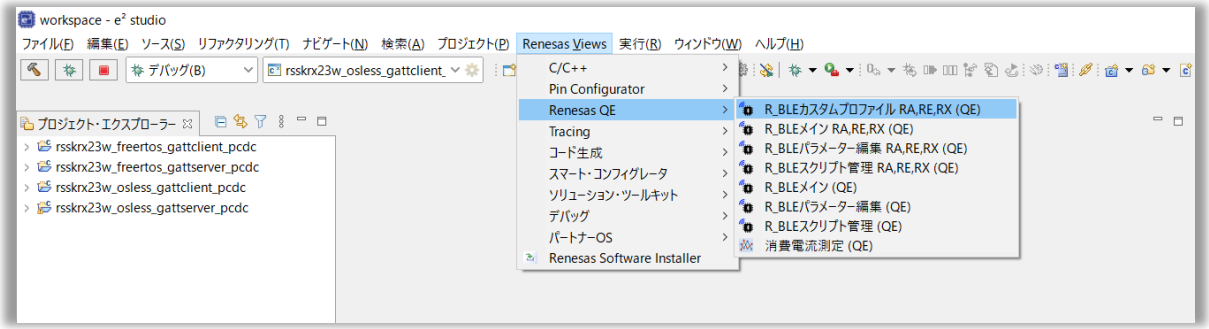

図 3-22　[Renesas Views]→[Renesas QE]→[R_BLE カスタムプロファイル　RA,RE,RX (QE)]

2. [R_BLE カスタムプロファイル　RA,RE,RX (QE)]でプロジェクトを選択します。

3. [R_BLE カスタムプロファイル　RA,RE,RX (QE)]に現在のプロファイル構成が表示されます。

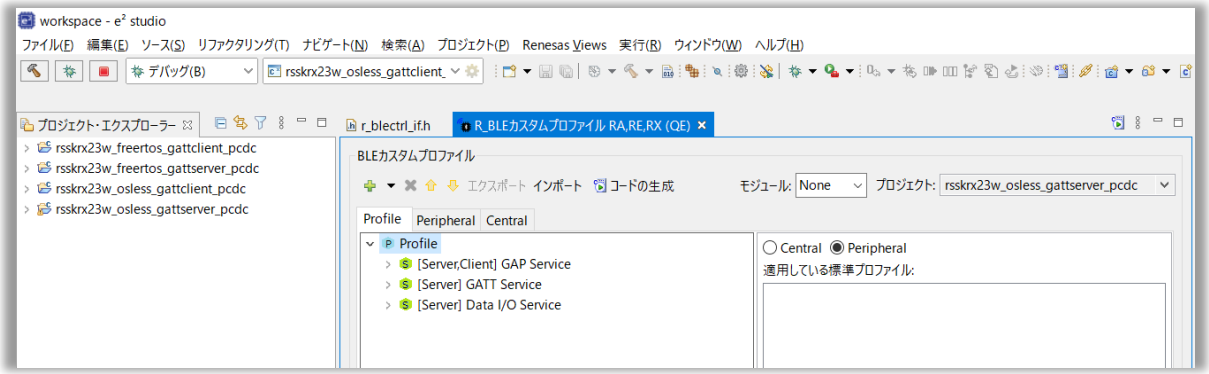

図 3-23　R_BLE カスタムプロファイル　RA,RE,RX (QE)

4. 下記のドキュメントを参照して、プロファイルの設計と QE for BLE が生成したソースコードへの実装を行ってください。

RX23W グループ　Bluetooth Low Energy プロファイル開発者ガイド (R01AN4553)
https://www.renesas.com/document/apn/rx23w-group-bluetooth-low-energy-profile-developers-guide-rev110

3.8.2　BLE 制御アプリケーションのスループット向上のための動作設定

本サンプルプログラムは QE for BLE が生成した app_main.c に対して、Bluetooth LE 接続中のスループット向上のため下記の処理を追加しています。

- Central デバイスとして動作時、`R_BLE_GAP_SetPhy()`をコールして 2M PHY に変更
- Central デバイスとして動作時、`R_BLE_GATTC_ReqExMtu()`をコールして ATT MTU サイズを 23byte から 247byte に拡張

3.8.3　BLE 制御アプリケーションのセキュリティ機能

本サンプルプログラムは Bluetooth LE 通信のペアリング、暗号化、暗号化、プライバシーを実行しません。これらのセキュリティ機能を使用する場合は、下記を参照してください。

RX23W グループ Bluetooth Low Energy アプリケーション開発者ガイド (R01AN5504)
https://www.renesas.com/document/apn/rx23w-group-bluetooth-low-energy-application-developers-guide-rev110

→9 章「セキュリティ」

3.8.4　BLE 制御アプリケーションのソフトウェアスタンバイモードの許可

本サンプルプログラムは BLE FIT モジュールの `R_BLE_LPC_SetInhibitSoftwareStandby(false)`をコールすることで、低消費電力状態のソフトウェアスタンバイモードへの遷移を許可します。ソフトウェアスタンバイモードでは SCI が停止します。本サンプルプログラムは SCI8 をコンソールへの動作ログの送信のみ使用しますが、SCI8 でコンソールからの文字列の受信にも使用する場合、ソフトウェアスタンバイモードへの遷移は禁止してください。

3.8.5　Bluetooth SIG 認証

Bluetooth 技術を使用する最終製品は、Bluetooth SIG による認証を取得する必要があります。ルネサス製の Bluetooth 対応 MCU またはモジュールを利用した最終製品の Bluetooth 認証を取得する方法については、以下を参照してください。

Bluetooth LE マイコン／モジュール Bluetooth 認証取得 (R01AN3177)
https://www.renesas.com/document/apn/bluetooth-le-microcomputer-module-bluetooth-qualification-acquisition-application-note-rev140

3.8.6　電波法の遵守と認証の取得

無線技術を使用する最終製品は、販売する各国の電波法を遵守する必要があります。認証の取得については、各国の認証機関または認証代行機関が発行する文書等をご確認ください。

なお RX23W モジュール(R5F523W8CDLN/ R5F523W8DDLN)は日本(技適)、北米(FCC/ISED)、欧州(CE)の電波法認証を取得済みです。

3.8.7　USB ドライバの DTC 転送または DMA 転送

USB Basic Mini FIT モジュールは DTC 転送または DMA 転送を使用できます。DTC 転送または DMA 転送を使用する手順を以下に示します。

1. e² studio の[プロジェクト・エクスプローラー]でスマートコンフィグレータ設定ファイル(.scfg)をダブルクリックして、スマートコンフィグレータを開きます。

2. [ソフトウェアコンポーネント設定]の[コンポーネントの追加]ボタン(緑の+アイコン)をクリックして、コンポーネントの追加ダイアログで DTC FTT モジュール(r_dtc_rx)または DMA FIT モジュール(r_dmaca_rx)を追加します。

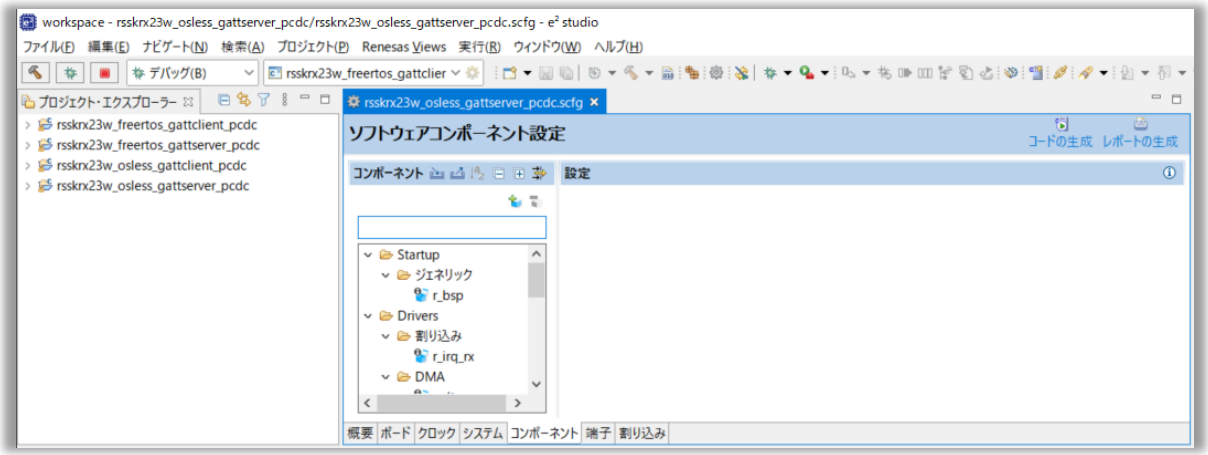

図 3-24　スマートコンフィグレータのソフトウェアコンポーネント設定

3. [ソフトウェアコンポーネント設定]で USB Basic Mini FIT モジュール(r_usb_basic_mini)を選択します。DTC 転送を使用する場合は[DTC use setting] (USB_CFG_DTC)を"Using DTC"に設定します。DMA 転送を使用する場合は[DMA use setting] (USB_CFG_DMA)を"Using DMA"に設定します。

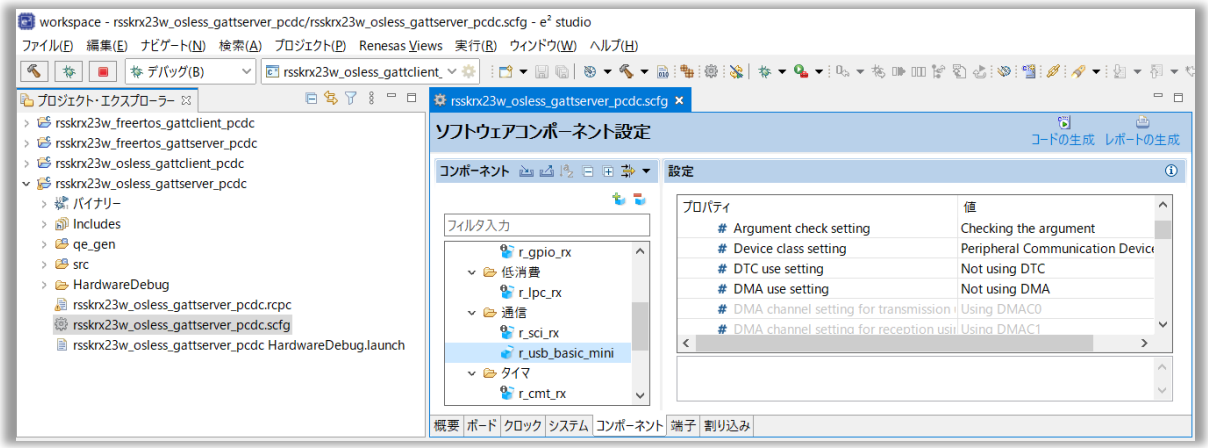

図 3-25　USB Basic Mini FIT モジュール設定(r_usb_basic_mini)

4. [ソフトウェアコンポーネント設定]の[コードの生成]ボタンをクリックして、ソフトウェアコンポーネントのソースコードを生成します。DMA 転送を使用する場合、以降の手順は不要です。

5. DTC 転送を使用する場合、DTC FIT モジュールが DTC ベクタテーブル領域を動的に確保するため、ヒープ領域を拡張する必要があります。DTC FIT モジュールが必要とするヒープサイズは、スマートコンフィグレータが生成した以下のコードファイルの DTC_VECTOR_TABLE_SIZE_BYTES マクロを参照してください。なお下記は DTC FIT モジュール V3.80 のマクロ定義です。

{プロジェクトフォルダ}¥src¥smc_gen¥r_dtc_rx¥src¥targets¥rx23w¥r_dtc_rx_target.h

```
/* Size of DTC Vector table (in byte units) */
#define DTC_VECTOR_TABLE_SIZE_BYTES (0x3E8 + 0x400)
```

[ソフトウェアコンポーネント設定]でボードサポートパッケージモジュール (r_bsp)を選択します。アプリケーションやシステムが必要とするヒープサイズに DTC_VECTOR_TABLE_SIZE_BYTES で指定されたサイズを加算して、[Heap size] (BSP_CFG_HEAP_BYTES)に設定します。

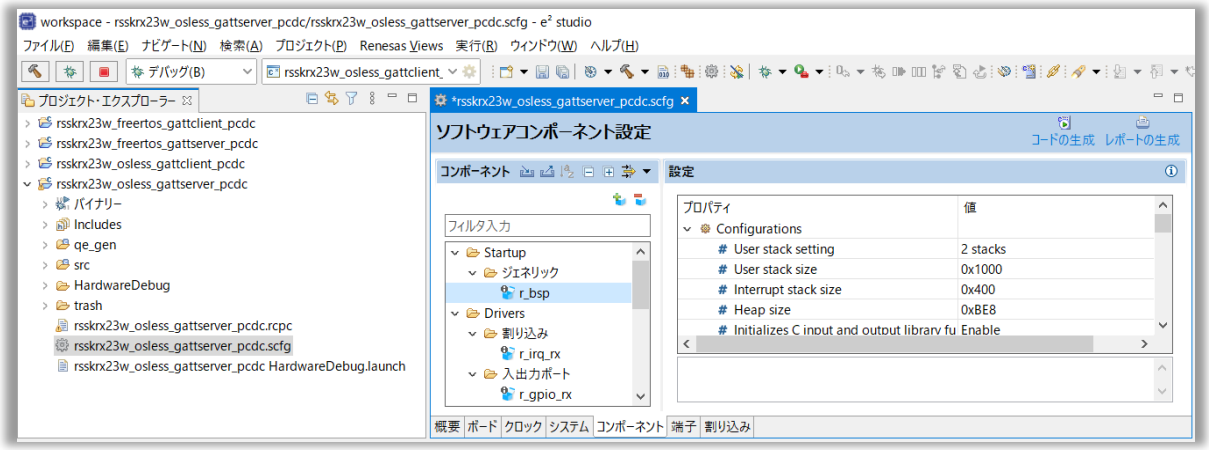

図 3-26　ボードサポートパッケージモジュール設定(r_usb_basic_mini)

6. [コードの生成]ボタンをクリックして、ソフトウェアコンポーネントのソースコードを生成します。

3.8.8　USB Vendor ID の取得

USB 製品の Vendor ID は、必ず最終製品として販売するお客様(ベンダー)が取得する必要があります。Vendor ID の取得については、USB Implementers Forum (USB-IF)のウェブサイトを参照してください。

Getting a Vendor ID - USB Implementers Forum, Inc.
https://www.usb.org/developers/vendor/

取得した Vendor ID のほか、ベンダーで割り当てることができる Product ID や Release Number などは以下のコードファイルに設定してください。

{プロジェクトフォルダ}¥src¥usbctrl¥r_usb_pcdc_descriptor.c

```
/* Vendor ID */
#define USB_VENDORID                    (0x0000u)
/* Product ID */
#define USB_PRODUCTID                   (0x0002u)
/* Release Number */
#define USB_RELEASE                     (0x0200u)
```

図 3-27　Vendor ID、Product ID、Release Number のマクロ定義

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022.03.04	-	初版発行

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. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   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. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「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。
5. クロックについて
   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。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   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、将来の拡張機能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。
8. 製品間の相違について
   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、フラッシュメモリ、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、ノイズ幅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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。