RYZ014A and RA MCU

RYZ014A LTE 通信サンプルアプリケーション

要旨

RYZ014A は LTE Cat M1 通信することができるモジュールです。RYZ014A はホスト MCU と UART 通信を介して接続し、AT コマンドによって動作をコントロールすることができます。本サンプルアプリケーションは RA MCU をホスト MCU として使用し、RYZ014A を制御して MQTT 通信を行うためのプログラムを提供します。本サンプルアプリケーションはホスト MCU から RYZ014A へ AT コマンドを送信するプログラムを、AT コマンドマネジメントフレームワークを使用して実装しています。AT コマンドマネジメントフレームワークは使用することで RYZ014A の LTE Cat M1 通信機能がサポートする様々な通信プロトコルを利用するアプリケーションを実装可能です。本ドキュメントでは本サンプルアプリケーションに実装された MQTT 通信アプリケーションと AT コマンドマネジメントフレームワークの説明を行います。

動作確認デバイス

RYZ014A

EK-RA6M5

目次

1. 概要

1.1 動作概要

RYZ014A は LTE Cat M1 通信機能を持つモジュールです。この機能は UART 経由で AT コマンドを文字列として入力することで制御することができます。

図 1.1 RYZ014A

本サンプルアプリケーションは、ホスト MCU として RA MCU を使用し RYZ014A を制御するソフトウェアです。RA MCU は UART 通信で AT コマンドを文字列として RYZ014A へ送信します。AT コマンドに対する応答文字列も UART 通信で受信します。これらのやり取りを介して RA MCU は RYZ014A の LTE Cat M1 通信機能を利用します。

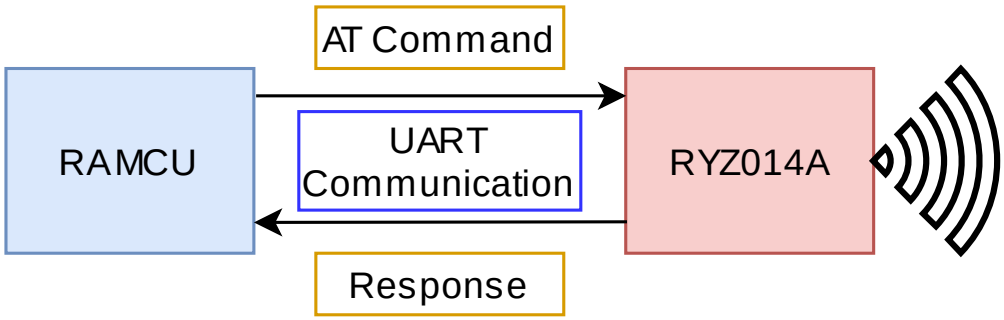

図 1.2 RYZ014A とホスト MCU の通信

1.2 ソフトウェア構成

本サンプルアプリケーションのソフトウェア構成は以下の通りです。

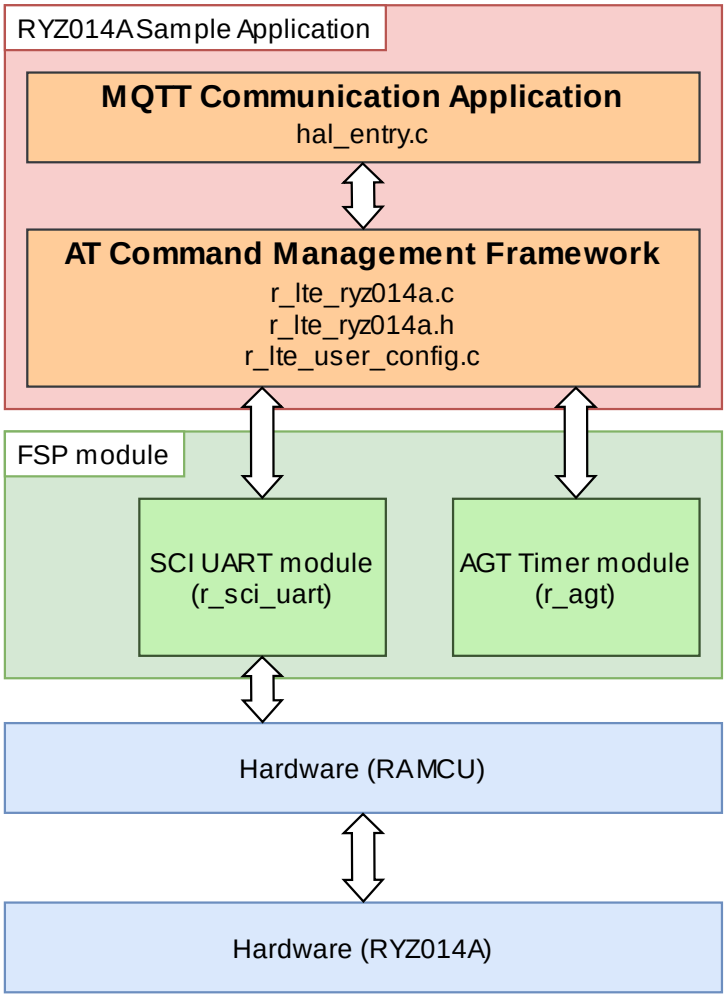

図 1.3 ソフトウェア構成

本サンプルアプリケーションは RYZ014A を使用して MQTT 通信を行うプログラムがベアメタル上に実装されています。このプログラムは RYZ014A へ AT コマンドを送信するための AT コマンドマネジメントフレームワークと MQTT 通信するための API を呼び出す MQTT 通信アプリケーションの 2 つで実装されています。

MQTT 通信アプリケーションでは MQTT サーバに接続した後、ボタンの押下に対応してデータを送信するプログラムを実装しています。MQTT 通信アプリケーションは AT コマンドマネジメントフレームワークに実装された API を利用して実装されています。MQTT 通信アプリケーションの詳しい説明については「2 MQTT 通信アプリケーション」を参照してください。

AT コマンドマネジメントフレームワークは RYZ014A への AT コマンド送信と RYZ014A から受信したレスポンスの処理を実装するためのフレームワークです。アプリケーションで AT コマンドマネジメントフレームワーク上に実装された API 関数を呼び出すことで複数の AT コマンドを RYZ014A へ送信し、実行結果をコールバック関数でアプリケーションへ通知します。本サンプルアプリケーションでは RA6M5 が RYZ014A を通じて MQTT 通信ができるようにフレームワークを使用してフレームワークベースプログラムを実装しています。なお、AT コマンドマネジメントフレームワークは MQTT 通信以外の RYZ014A の機能を利用する場合も、そのアプリケーション開発を行う場合のベースとして利用されることを想定しています。AT コマンドマネジメントフレームワークの詳しい説明については「3 AT コマンドマネジメントフレームワーク」を参照して下さい。

2. MQTT 通信アプリケーション

2.1 アプリケーションの動作環境

MQTT 通信アプリケーションを動作させるための環境について説明します。
本サンプルプログラムは以下のハードウェア環境で動作します。

表 2.1 アプリケーションのハードウェア環境

Hardware	Description
PMOD Expansion Board for RYZ014A	RYZ014A モジュール (RTKYZ014A0B00000BE)
EK-RA6M5	ホスト MCU となる RA MCU 搭載評価ボード (RTK7EKA6M5S00001BE)
Windows PC	RA MCU のアプリ開発環境及び動作確認用デバッグコンソール

本サンプルプログラムは以下のソフトウェア環境で開発と動作確認を行っています。

表 2.2 アプリケーションのソフトウェア環境

Software	Version	Description
e2 studio	2022-04	Renesas の IDE (http://www.renesas.com/e2studio)
Flexible Software Package (FSP)	3.8.0	RA MCU で使用できるドライバ (http://www.renesas.com/fsp)
SEGGER J-Link RTT Viewer	7.64	Debug Write を表示するビューア (https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/)

以下の手順でアプリケーションの実行準備を行います。

1. EK-RA6M5 と RYZ014A を Pmod コネクタで接続します。
   この時、EK-RA6M5 の PMOD2(J25)に接続します。

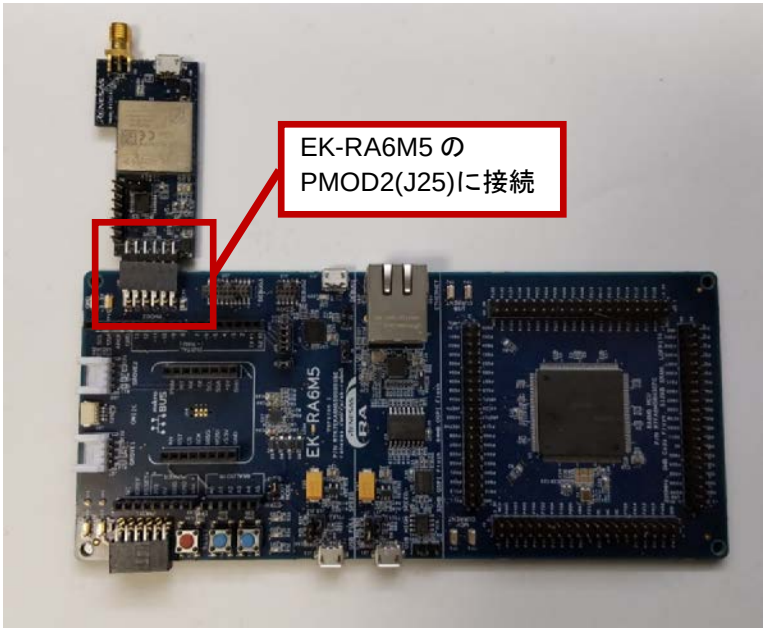

図 2.1 RYZ014A PMOD と EK-RA6M5 を接続

2. EK-RA6M5 と RYZ014A に USB ケーブルを接続します。
また、RYZ014A にアンテナを接続します。

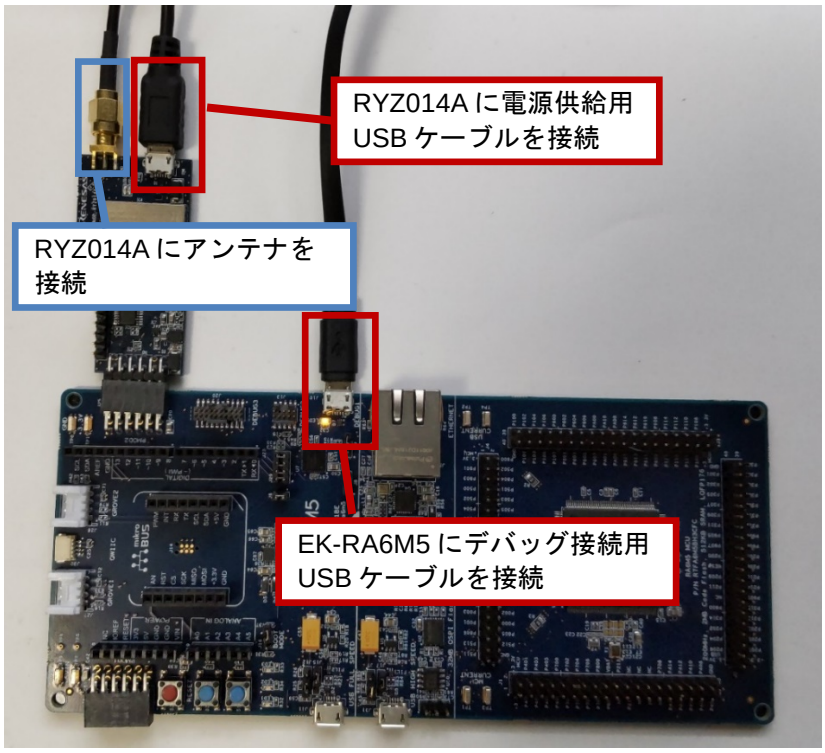

図 2.2 USB ケーブルとアンテナの接続

3. e2 studio にサンプルプロジェクトをインポートします。

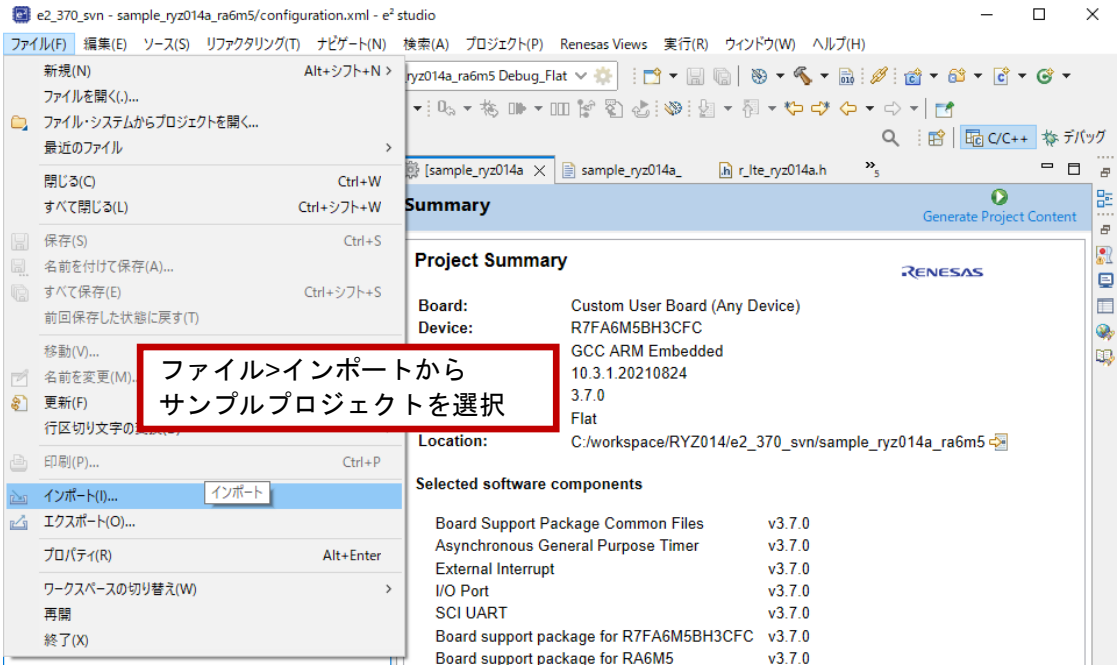

図 2.3 サンプルプロジェクトの追加

4. アクセスポイント名と LTE バンドを変更します。hal_entry.c には LTE ネットワークへ接続する際に使用するアクセスポイント名と LTE バンドが指定されています。ユーザのアプリケーションに合わせてこれらの値を変更してください。

アクセスポイント名 (APN)

APN は基本的に使用する SIM によって変化します。利用できる APN については SIM の提供元に問い合わせてください。ルネサスから提供されるキットに付属する SIM のアクティベート方法及び利用可能な APN については各キットのマニュアルを参照してください。

LTE バンド

LTE バンドは使用する地域や使用するネットワーク網などによって変化します。使用される LTE バンドがわかっている場合そのバンドを指定してください。以下に LTE バンドの例を示します。

- “1,19”
  - DOCOMO バンドを指定する場合。
- “2,4,12”
  - AT&T バンドを指定する場合。
- “1,2,3,4,5,8,12,13,17,18,19,20,25,26,28,66”
  - 使用するバンドが不明である場合。この値を指定した場合、初回にバンドを選択されるまで数分程度の時間がかかります。

本アプリケーションでは以下の APN と LTE バンドを使用して動作確認を行っています。

- APN: soracom.io
- LTE バンド: 1,19

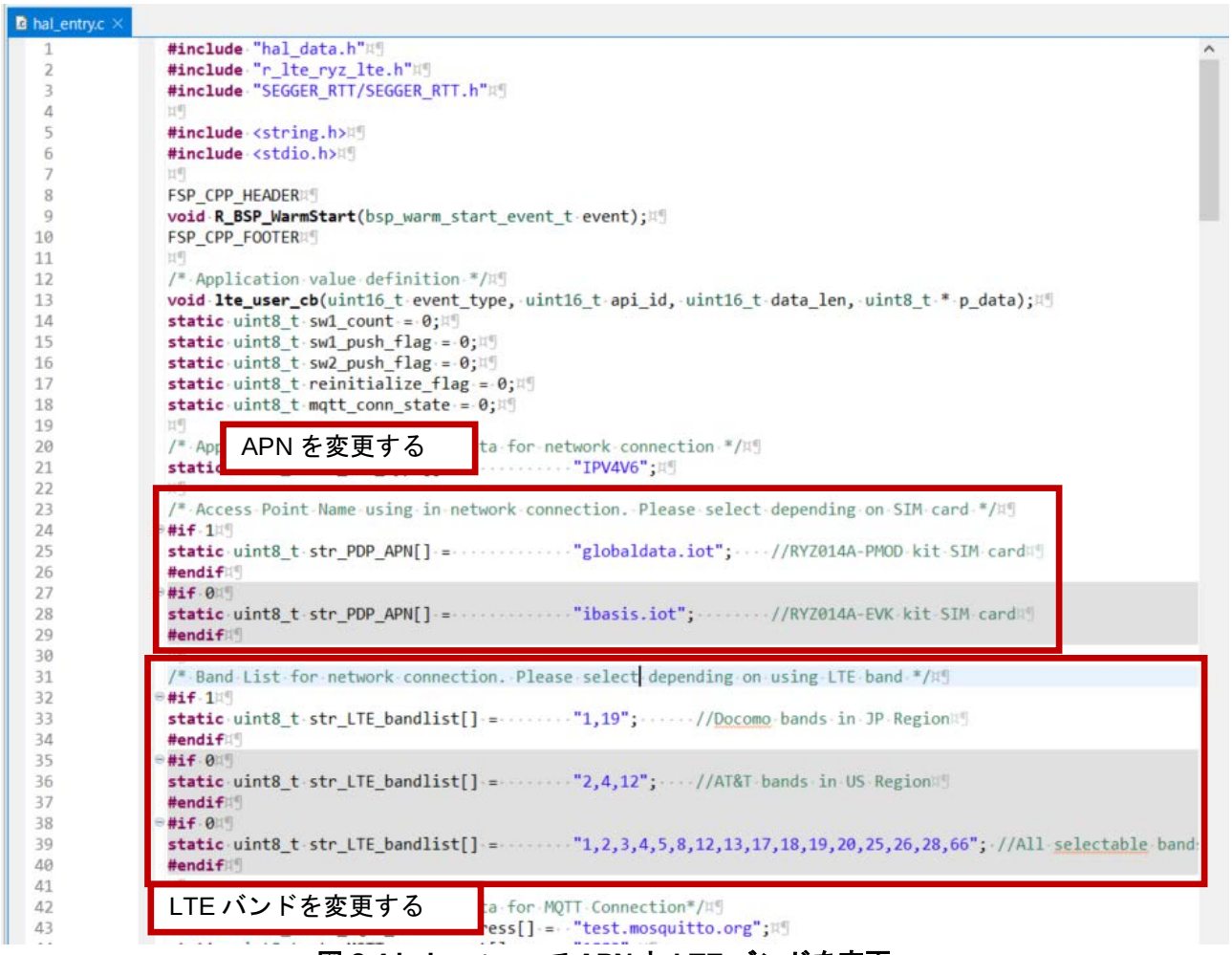

図 2.4 hal_entry.c で APN と LTE バンドを変更

5. サンプルプロジェクトをビルドします。本サンプルプロジェクトでは SEGGER J-Link RTT Viewer を用いて実行の状況をモニタリングします。その設定のため、ビルド後に生成される Debug フォルダ内の.map ファイルから_SEGGER_RTT のアドレスを確認します。

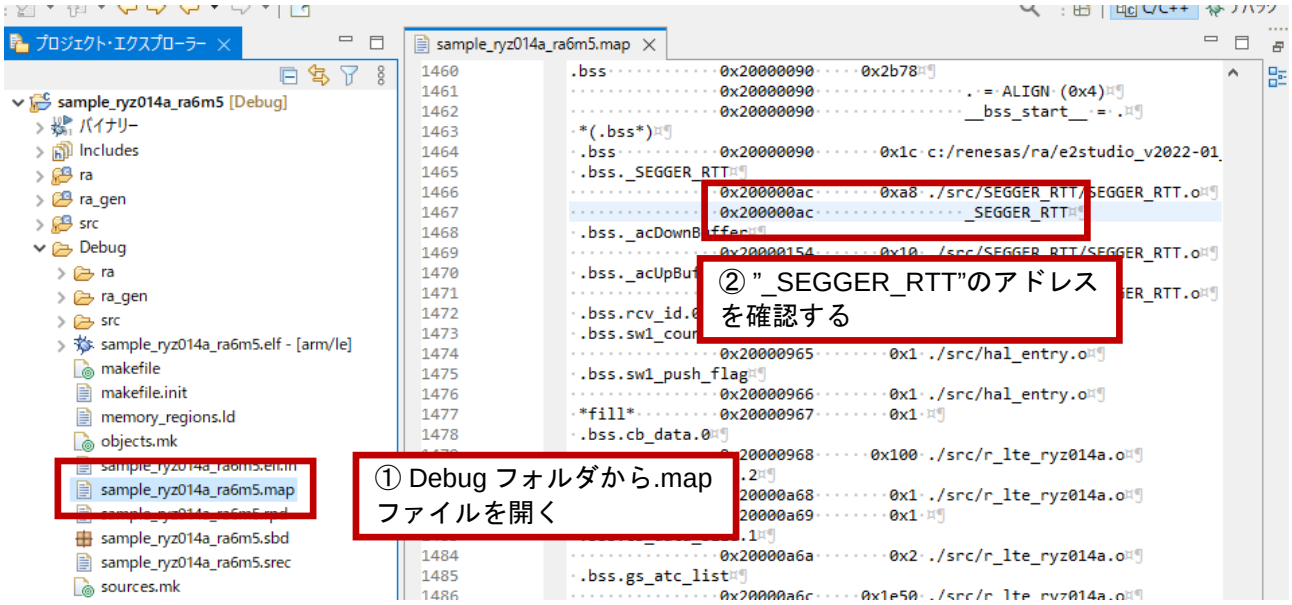

図 2.5 .map ファイルの確認

6. RTT Viewer を起動し、EK-RA6M5 に接続します。接続には上記アドレスを入力します。

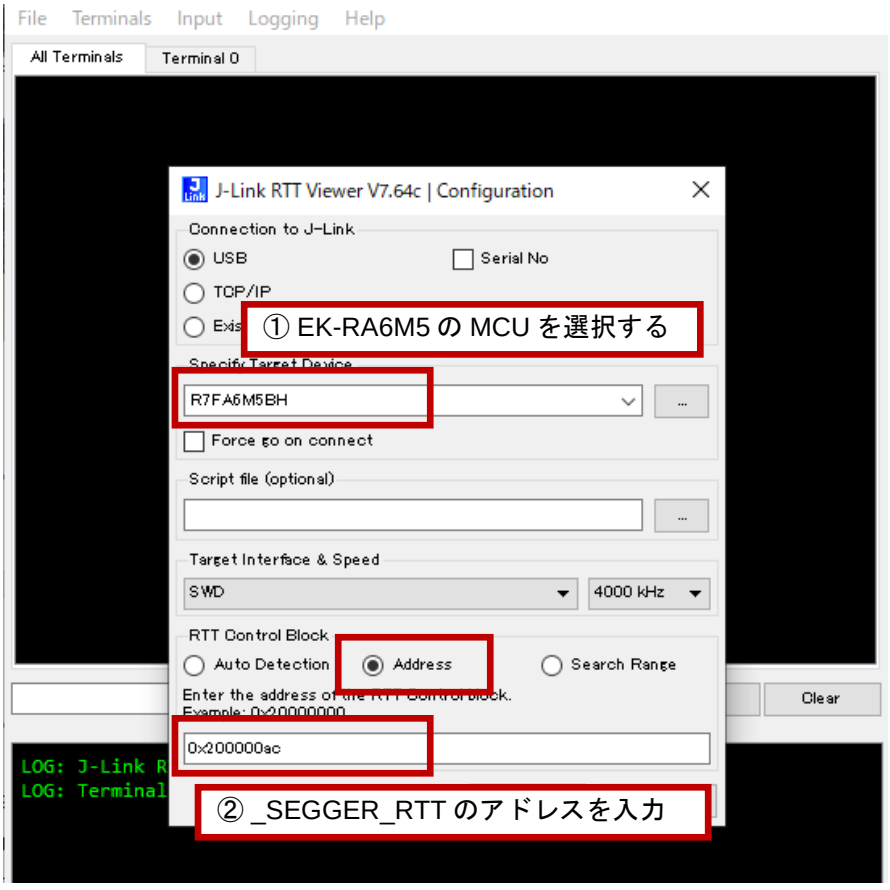

図 2.6 RTT Viewer の起動

2.2 アプリケーションの動作概要

本サンプルプログラムで提供されるプログラムの動作について説明します。

本サンプルプログラムの実行とともに RYZ014A モジュールをリセットします。RYZ014A のリセット後、LTE 経由でネットワークに接続し、続いて MQTT サーバ(ここでは公開されている「test.mosquitto.org」を利用しています)に接続します。次に MQTT サーバに Subscribe 要求を行った後、「SW READY」の文字列を RTT Viewer に表示します。この状態でボード上のスイッチで操作可能となります。

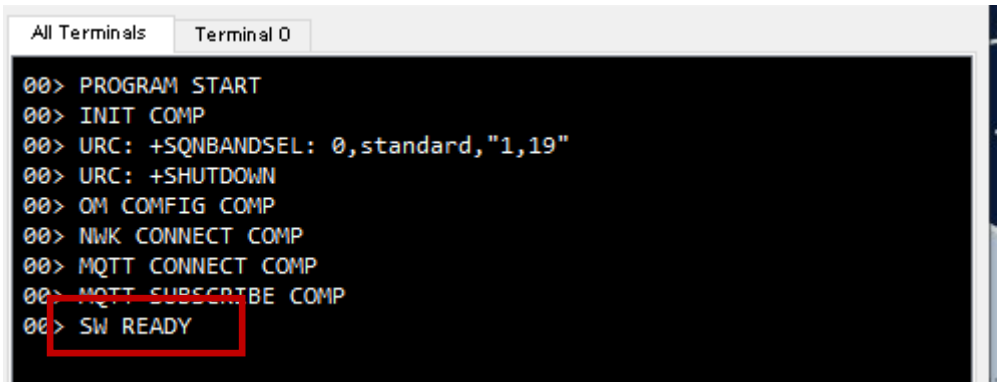

図 2.7 MQTT サーバへ接続

「SW READY」が表示されたあと SW1 を押すことで MQTT サーバへ Publish によりメッセージを送信します。先に MQTT サーバには Subscribe 要求を行っているので、そのメッセージの ID などが送信されます。それを受けてメッセージ受信要求を送信し、受信したメッセージを RTT Viewer に表示します。SW1 を押す回数によって送信する文字列データは変化します。

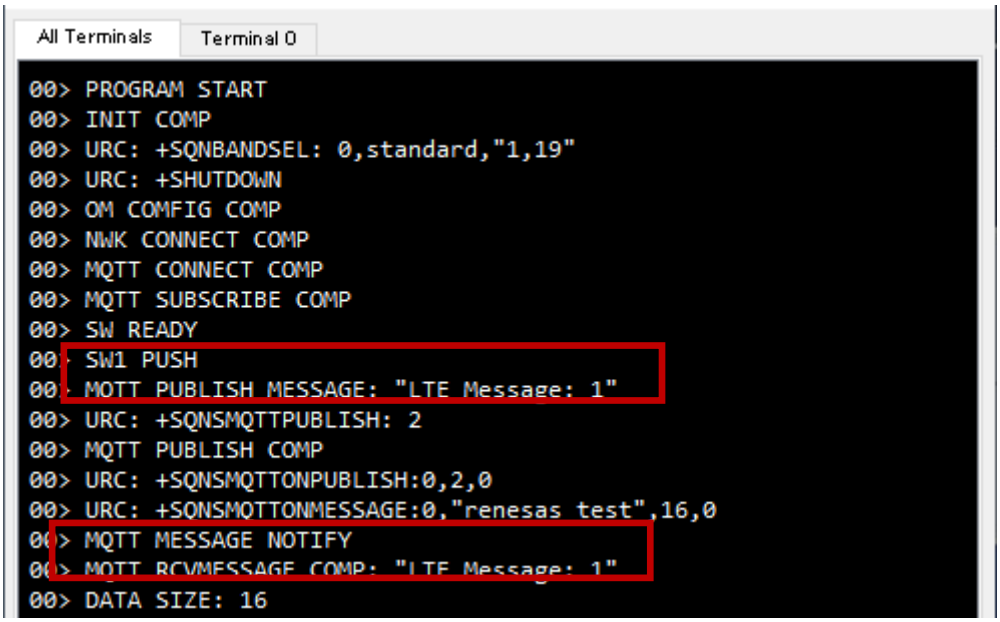

図 2.8 SW1 を押下

SW2 を押すと MQTT サーバから切断した後、ネットワークからも切断します。

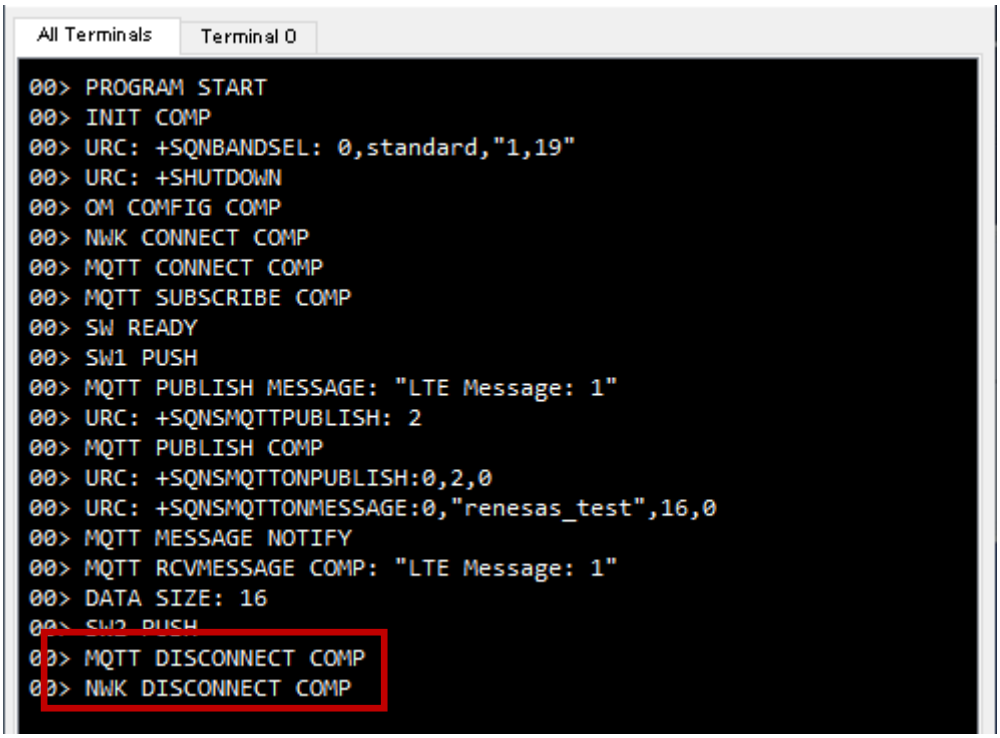

図 2.9 SW2 を押下

なお、電波状況の悪化などの理由でネットワークや MQTT サーバから切断された場合、本サンプルアプリケーションでは再度 MQTT サーバへ接続しようとします。そのため電波状況が回復した後、ボタンなどを押すことなく MQTT サーバへ再接続し、Subscribe 要求を行った後「SW READY」が表示されます。この後スイッチの操作が可能になります。

3. ATコマンドマネジメントフレームワーク

3.1 フレームワーク概要

ホストMCUからのRYZ014Aを操作するはUART通信を使用してATコマンドとレスポンスの送受信を通じて行います。ATコマンドマネジメントフレームワークはATコマンドとレスポンスの送受信を効率よく実装するためのフレームワークです。本サンプルプログラムではATコマンドマネジメントフレームワークを使用してMQTT通信をするためのフレームワークベースプログラムを実装しています。

本サンプルプログラムのフレームワークベースプログラムで実装されているAPIはマネジメントAPIとATコマンドAPIの2つに分類されます。マネジメントAPIはフレームワークベースプログラムの初期化や、一連のATコマンドをレスポンスに応じて送信するための APIです。ATコマンドAPIはATコマンドを送信するための APIです。ATコマンドAPIで送信されたATコマンドの実行結果はコールバック関数としてアプリケーションに通知されます。

ATコマンドマネジメントフレームワークはFSPモジュールであるSCI UARTモジュールとAGT Timerモジュールを使用して実装されています。RYZ014Aへの ATコマンド送信とRYZ014Aからのレスポンスの受信でSCI UARTモジュールを使用しています。またATコマンド実行後のタイムアウトを計測するため、AGT Timerモジュールを使用しています。

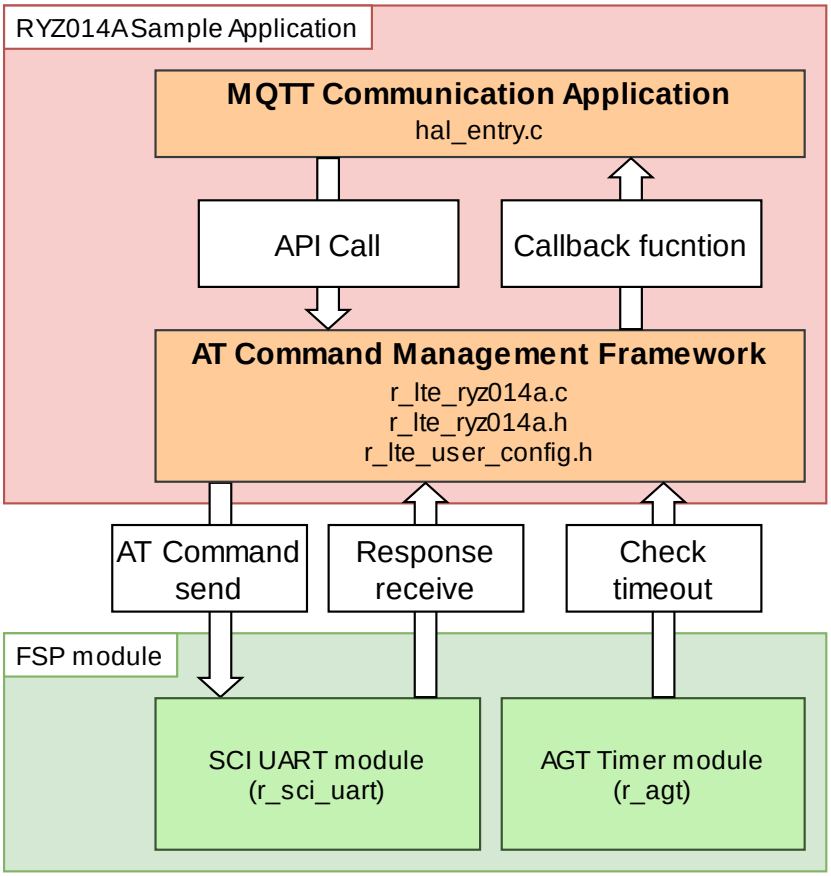

図 3.1 ATコマンドマネジメントフレームワーク

本サンプルプログラムのフレームワークベースプログラムから提供されるAPI関数は「3.2 API関数」を参照してください。

ATコマンドAPIの実行結果はアプリケーションへコールバック関数で通知されます。コールバック関数と通知されるデータについては「3.3 コールバック関数」を参照してください。

またATコマンドマネジメントフレームワークを他のRA MCUで使用する場合は「r_lte_user_config.h」を編集してできます。設定可能な値は「3.4 ユーザ固有値の設定」を参照してください。

3.2 API 関数

本サンプルプログラムのフレームワークベースプログラムを使用して実装されている API 関数はマネジメント API と AT コマンド API の 2 種類に分類されます。マネジメント API はフレームワークベースプログラムの初期化や、一連の AT コマンドをレスポンスに応じて送信するための API です。AT コマンド API は AT コマンドを送信するための API です。マネジメント API は「3.2.1 マネジメント API」、AT コマンド API は「3.2.2 AT コマンド API」で説明します。

3.2.1 マネジメント API

マネジメント API はフレームワークベースプログラムの初期化や、一連の AT コマンドをレスポンスに応じて送信するための API です。マネジメント API はアプリケーションのメインループに実装する必要があります。本サンプルプログラムのフレームワークベースプログラムをベースに機能追加等を行う場合でも、基本的にマネジメント API のプログラムは変更不要です。

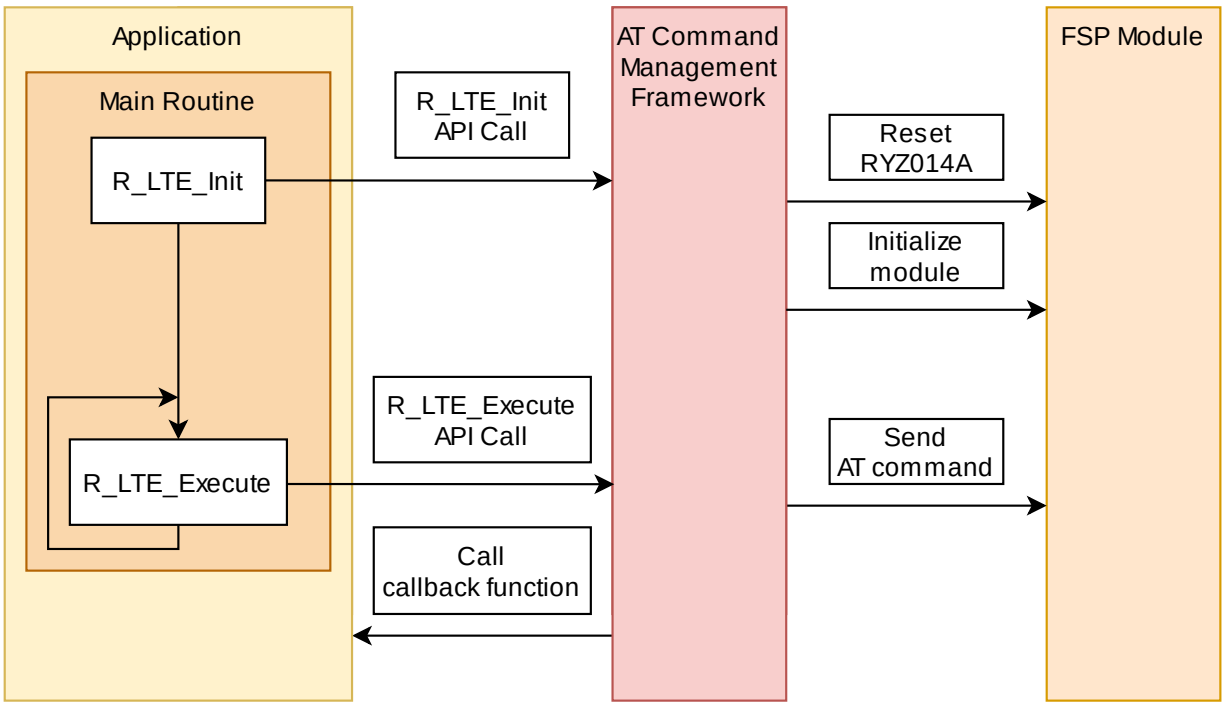

図 3.2 マネジメント API

3.2.1.1 R_LTE_Init

関数名	R_LTE_Init	
概要	フレームワークベースプログラムの初期化を行う	
引数	lte_cb_t * p_callback_fun (IN)	登録するコールバック関数。 コールバック関数の型については「3.3 コールバック関数」を参照。
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
詳細機能	初期化として、以下を実行します。 ・ 使用する FSP module の初期化 ・ RYZ014A のハードウェアリセット ・ API 関数の実行結果やイベントをアプリケーションに通知するコールバック関数の登録 本 API を実行後、RYZ014A をリセットするための AT コマンドが送信されます。 この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_INIT (0xFF) 本 API はアプリケーションのメインループ前に必ず呼び出してください。	

3.2.1.2 R_LTE_Execute

関数名	R_LTE_Execute	
概要	フレームワークベースプログラムの実行を行う	
引数	void	なし
返り値	void	なし
詳細機能	フレームワークベースプログラムで実行する各種処理を実行します。 本関数では以下の動作を実行します。 ・ RYZ014A から送信されるデータの受信と解析 ・ AT コマンドの送信待機リストに登録されたデータの順次送信 ・ 受信したデータに合わせたコールバック関数の呼び出し 本関数はアプリケーションのメインループ内で必ず繰り返しコールしてください。	

3.2.2 AT コマンド API

AT コマンド API は実行する動作に合わせて一連の AT コマンドを送信するための API です。アプリケーションから AT コマンド API を呼び出すことで送信待機リストに 1 つあるいは複数の AT コマンドが追加されます。送信待機リストに追加された AT コマンドは RYZ014A からのレスポンスに応じて順次 RYZ014A に送信されます。AT コマンド API で指定したすべての AT コマンドが送信されたら AT コマンドの送信結果をコールバック関数としてアプリケーションに通知します。AT コマンド API の呼出し後、コールバック関数で結果が通知される前に次の AT コマンド API を呼び出さないでください。また AT コマンド API は割り込みハンドラから実行しないでください。メインルーチン(AT コマンドマネジメントフレームワークのコールバック関数を含む)から実行してください。

本サンプルプログラムのフレームワークベースプログラムでは RYZ014A で MQTT 通信を行うために必要な API を実装しています。MQTT 通信アプリケーションで利用していない AT コマンドを用いた機能を実装したい場合、AT コマンドマネジメントフレームワークを使用して新しく AT コマンド API をユーザが追加し、アプリケーションを開発することを想定しています。

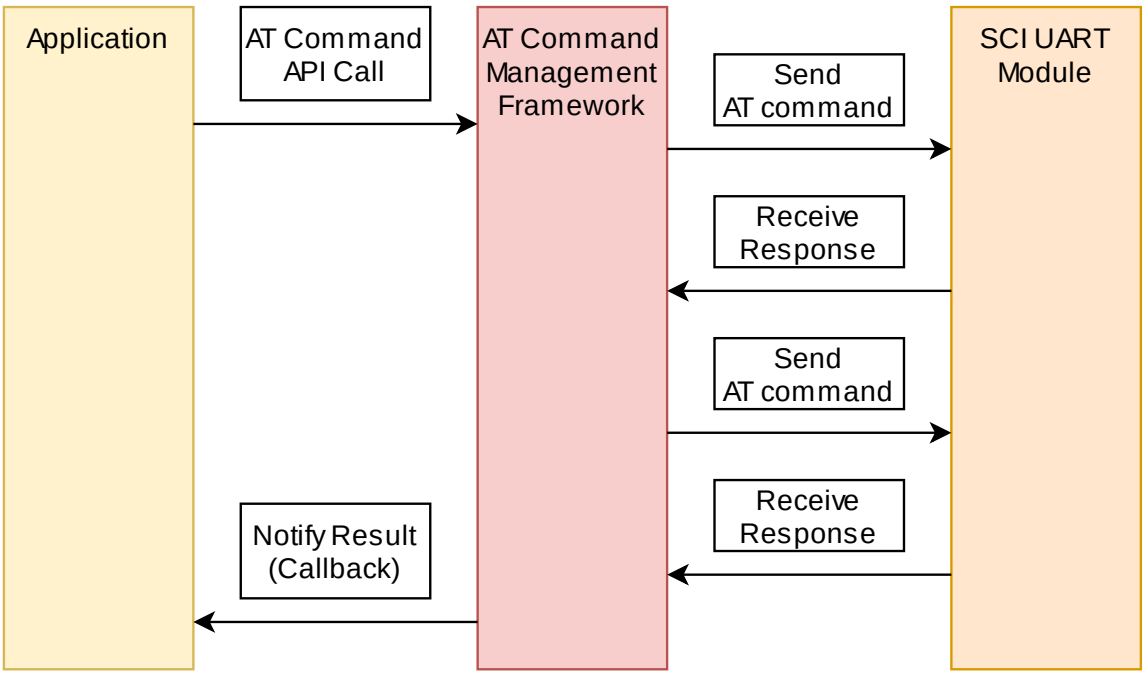

図 3.3 AT コマンド API

3.2.2.1 R_LTE_OM_Config

関数名	R_LTE_OM_Config	
機能概要	オペレータモードを設定する	
引数	uint8_t * p_pdp_type (IN)	PDP コンテキストの種類 例：”IPV4V6”
	uint8_t * p_pdp_apn (IN)	PDP コンテキストのアクセスポイント名 例：”soracom.io”
	uint8_t * p_bandlist (IN)	LTE のバンド 例：”1,19”
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを順番に送信します。 1. “AT+CFUN=0" 2. “AT+CGDCONT=1,[p_pdp_type],[p_pdp_apn]” 3. “AT+SQNCTM=“standard” 4. “AT+SQNBANDSEL=0," standard ",[p_bandlist]” 5. “AT^RESET” 6. “AT+CMEE=1” この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_OM_CONFIG (0x01)	

3.2.2.2 R_LTE_NWK_Connect

関数名	R_LTE_NWK_Connect	
機能概要	ネットワークに接続する	
引数	uint8_t mode	送信する AT コマンドを選択する
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	Mode の値に応じて以下の AT コマンドを順番に送信します。(現在は 0 のみ使用可能) • Mode = 0 1. “AT+CEREG=5" 2. “AT+CFUN=1” この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_NWK_CONNECT (0x02)	

3.2.2.3 R_LTE_NWK_Disconnect

関数名	R_LTE_NWK_Disconnect	
機能概要	ネットワークから切断する	
引数	uint8_t mode	送信する AT コマンドを選択する
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	Mode の値に応じて以下の AT コマンドを送信します。(現在は 0 のみ使用可能) • Mode = 0 1. “AT+CFUN=0” この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_NWK_DISCONNECT (0x03)	

3.2.2.4 R_LTE_MQTT_Connect

関数名	R_LTE_MQTT_Connect	
機能概要	MQTT 接続の設定を行い、サーバに接続する	
引数	uint8_t * p_username (IN)	MQTT 通信で使用するユーザ名 例：”sqn/gm01q”
	uint8_t * p_host (IN)	接続する MQTT サーバのアドレス 例：”test.mosquitto.org”
	uint8_t * p_port (IN)	接続する MQTT サーバのポート 例：”1883”
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを順番に送信します。 1. “AT+SQNSMQTTCFG=0,[p_username]” 2. “AT+SQNSMQTTCONNECT=0,[p_host] ,[p_port]” この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_MQTT_CONNECT (0x04)	

3.2.2.5 R_LTE_MQTT_Subscribe

関数名	R_LTE_MQTT_Subscribe	
機能概要	サブスクライブするトピックを指定する	
引数	uint8_t * p_topic (IN)	MQTT 通信でサブスクライブするトピック 例：”sqn/test”
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを送信します。 1. AT+SQNSMQTTSUBSCRIBE=0,[p_topic] ,1" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_MQTT_SUBSCRIBE (0x05)	

3.2.2.6 R_LTE_MQTT_Publish

関数名	R_LTE_MQTT_Publish	
機能概要	MQTT のトピックにメッセージをパブリッシュする	
引数	uint8_t * p_topic (IN)	メッセージをパブリッシュするトピック 例：” sqn/test”
	uint16_t length (IN)	パブリッシュするメッセージのサイズ
	uint8_t * p_message (IN)	パブリッシュするメッセージ
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSMQTTPUBLISH=0,[p_topic],1,[length]" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_MQTT_PUBLISH (0x06)	

3.2.2.7 R_LTE_MQTT_RcvMessage

関数名	R_LTE_MQTT_RcvMessage	
機能概要	MQTT からメッセージを受信する	
引数	uint8_t * p_topic (IN)	受信するメッセージのトピック 例：”sqn/test”
	uint8_t message_id (IN)	受信するメッセージの ID
	uint16_t message_size (IN)	受信するメッセージのサイズ
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	message_id の値に応じて以下の AT コマンドを送信します。 • Message_id = 0 1. “AT+SQNSMQTTRCVMESSAGE=0,[p_topic]” • Message_id = 0 以外 1. “AT+SQNSMQTTRCVMESSAGE=0,[p_topic],[message_id]” この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 また受信したメッセージもコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_MQTT_RCVMESSAGE (0x07) 本 API は通常、”+SQNSMQTTONMESSAGE”の URC を受信した後に実行します。 引数に指定した message_id に該当する MQTT メッセージをまだ受信していない場合、コールバック関数でエラー” LTE_CME_ERR_OPERATION_NOT_SUPPORTED” が通知されます。	

3.2.2.8 R_LTE_MQTT_Disconnect

関数名	R_LTE_MQTT_Disconnect	
機能概要	MQTT サーバから切断する	
引数	void	なし
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSMQTTDISCONNECT=0" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_MQTT_DISCONNECT (0x08)	

3.2.2.9 R_LTE_SEC_CertificateAdd

関数名	R_LTE_SEC_CertificateAdd	
機能概要	証明書を追加する	
引数	uint8_t cet_id	追加する証明書の ID
	uint16_t cet_len	追加する証明書のデータ長
	uint8_t * p_cet_data	追加する証明書の文字列データ
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSNVW="certificate",[cet_id],[cet_len]" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_SEC_CERTIFICATEADD (0x09)	

3.2.2.10 R_LTE_SEC_CertificateRemove

関数名	R_LTE_SEC_CertificateRemove	
機能概要	証明書を削除する	
引数	uint8_t cet_id	削除する証明書の ID
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSNVW="certificate",[cet_id],0" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_SEC_CERTIFICATEREMOVE (0x0A)	

3.2.2.11 R_LTE_SEC_PrivateKeyAdd

関数名	R_LTE_SEC_PrivateKeyAdd	
機能概要	プライベートキーを追加する	
引数	uint8_t prk_id	追加するプライベートキーの ID
	uint16_t prk_len	追加するプライベートキーのデータ長
	uint8_t * p_prk_data	追加するプライベートキーの文字列データ
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_POINTER_NULL (0x0001)	引数のポインタが NULL になっている
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
	LTE_ERR_DATASIZE_OVERFLOW (0x0003)	引数のデータサイズが送信待機リストに登録できるサイズを超過した
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSNVW="privatekey",[prk_id],[prk_len]" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_SEC_PRIVATEKEYADD (0x0B)	

3.2.2.12 R_LTE_SEC_PrivateKeyRemove

関数名	R_LTE_SEC_PrivateKeyRemove	
機能概要	プライベートキーを削除する	
引数	uint8_t prk_id	削除するプライベートキーの ID
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	以下の AT コマンドを送信します。 1. “AT+SQNSNVW="privatekey",[prk_id],0" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_SEC_PRIVATEKEYREMOVE (0x0C)	

3.2.2.13 R_LTE_NWK_ConnectionConfig

関数名	R_LTE_NWK_ConnectionConfig	
機能概要	接続とセキュリティを設定する	
引数	uint8_t ca_cer_id	CA 証明書の ID
	uint8_t client_cer_id	クライアント証明書の ID
	uint8_t prk_id	プライベートキーの ID
返り値	LTE_SUCCESS (0x0000)	成功
	LTE_ERR_IN_PROCESS (0x0002)	他の LTE API が実行中である
詳細機能	以下の AT コマンドを順番に送信します。 1. “AT+SQNSCFG=1,1,1" 2. “AT+SQNSPCFG=1,2,,5,[ca_cer_id],[client_cer_id],[prk_id],"""" この API で送信された AT コマンドの実行結果はコールバック関数で通知されます。 以下の API_ID で通知されます。 LTE_API_NWK_CONNECTIONCONFIG (0x0D)	

3.3 コールバック関数

RYZ014A へ AT コマンドを送るとレスポンスが返ってきます。また、RYZ014A の状態が変化した場合は RYZ014A から Unsolicited Response Code (URC)が送信されます。本サンプルプログラムのフレームワークベースプログラムでは RYZ014 からそれらのデータを受信した後、R_LTE_Execute 関数内で解析します。アプリケーションに通知する必要がある場合、R_LTE_Execute 関数内でコールバック関数を呼び出してアプリケーションに通知します。これにより、アプリケーションは AT コマンド API の実行結果を確認したり、RYZ014A の URC を確認したりすることができます。ここではコールバック関数の構造やコールバック関数によって通知されるイベントやデータについて説明します。

コールバック関数は以下の構造になっています。

型名	void * lte_cb_t	
引数	uint16_t event_type (In)	通知されるイベントの ID。 設定される値は表 3.1 を参照してください。
	uint16_t api_id (In)	フレームワークベースプログラムがどの API を実行しているかを示す ID。 設定される値は表 3.2 を参照してください。
	uint16_t data_len (in)	p_data で通知されるデータのサイズ。
	void * p_data (out)	アプリケーションに通知するデータのポインタ。 データの内容はイベント種類に応じて変化します。 格納されるデータはコールバック関数が呼び出されるたびに上書きされるので必要に応じてアプリケーションで保持して下さい。

event_type と api_id の値はフレームワークベースプログラム内のマクロ形式で定義された値を使用します。以下にそれぞれの値を示します。

表 3.1 event_type の値

定義名	値	説明
LTE_EVENT_API_COMPLETE	0x0000	API 関数で指定した動作が正常に完了したことを通知するイベント。 p_data には呼び出した API に合わせたデータが設定される。
LTE_EVENT_ERROR	0x0001	API 関数で指定した動作でエラーが発生したことを通知するイベント。 p_data にはエラーが数値データとして設定される。
LTE_EVENT_RCVURC	0x0002	URC を受信したことを通知するイベント。 p_data には URC が文字列データとして設定される。
LTE_EVENT_TIMEOUT_ERROR	0x0003	AT コマンドの送信後、レスポンスの受け取りまでにタイムアウトが発生したことを通知するイベント。 AT コマンドの送信後、60s 経過するとタイムアウトが発生する。
LTE_EVENT_FATAL_ERROR	0x0004	致命的なエラーが発生した場合に通知されるイベント。 意図しないタイミングで URC”+SYSSTART”を受信したときにコールバック関数を呼び出す。

表 3.2 api_id の値

定義名	値	対応する API
LTE_API_NO_CURRENT_API	0x0000	なし
LTE_API_OM_CONFIG	0x0001	R_LTE_OM_Config
LTE_API_NWK_CONNECT	0x0002	R_LTE_NWK_Connect
LTE_API_NWK_DISCONNECT	0x0003	R_LTE_NWK_Disconnect
LTE_API_MQTT_CONNECT	0x0004	R_LTE_MQTT_Connect
LTE_API_MQTT_DISCONNECT	0x0005	R_LTE_MQTT_Disconnect
LTE_API_MQTT_SUBSCRIBE	0x0006	R_LTE_MQTT_Subscribe
LTE_API_MQTT_PUBLISH	0x0007	R_LTE_MQTT_Publish
LTE_API_MQTT_RCVMESSAGE	0x0008	R_LTE_MQTT_RcvMessage
LTE_API_SEC_CERTIFICATEADD	0x0009	R_LTE_SEC_CertificateAdd
LTE_API_SEC_CERTIFICATEREMOVE	0x000A	R_LTE_SEC_CertificateRemove
LTE_API_SEC_PRIVATEKEYADD	0x000B	R_LTE_SEC_PrivateKeyAdd
LTE_API_SEC_PRIVATEKEYREMOVE	0x000C	R_LTE_SEC_PrivateKeyRemove
LTE_API_NWK_CONNECTIONCONFIG	0x000D	R_LTE_NWK_ConnectionConfig
LTE_API_INIT	0x00FF	R_LTE_Init

コールバック関数は特定の状況で R_LTE_Execute 関数から呼び出されます。以下にコールバック関数が呼び出される状況とその時に設定されるデータを示します。

- AT コマンド API で送信する AT コマンドとそれに対するレスポンスがすべて送受信でき、レスポンスにエラーがない場合：
  - “event_type”に”LTE_EVENT_API_COMPLETE”が設定されます。
  - “p_data”には実行する AT コマンドに合わせてデータが設定されます。
    - AT コマンドの実行結果として URC を受信する場合、受信した URC の文字列データが登録されます。通知される文字列データのサイズは”data_len”に設定されています。
    - R_LTE_MQTT_RcvMessage など別途データを受信する AT コマンド API を呼び出している場合、受信した文字列データが登録されます。受信したデータサイズが”LTE_DATA_STR_SIZE”を超過する場合、超過した分のデータは破棄され、前半部分のデータが登録されます。通知される文字列データのサイズは”data_len”に設定されています。
    - 上記以外の場合、データは設定されません。”data_len”は 0 に設定されています。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(LTE_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case LTE_API_OM_CONFIG:
            {
                /* Connect to network after configuration of operation mode complete
*/
                SEGGER_RTT_printf(0, "OM COM                    ");
                R_LTE_NWK_Connect(1);
            } break;

/* 省略 */

            case LTE_API_MQTT_RCVMESSAGE
            {
                /* Display received message */
                SEGGER_RTT_printf(0, "MQTT RCVMESSAGE COMP: %s\n",p_data);
            } break;
```

LTE_EVENT_API_COMPLETE のイベント通知

api_id でどの AT コマンド API の結果か判断する

データを受信する場合 p_data に登録されている

図 3.4 LTE_EVENT_API_COMPLETE のイベント通知 (hal_entry.c)

- RYZ014A に送信した AT コマンドに対するレスポンスがエラーだった場合：
  - “event_type”に”LTE_EVENT_ERROR”が設定されます。
  - “p_data”にはエラーを示す値が登録されています。この値を確認するため、LTE_ERROR_DECODE 関数を使用して 16bit の値として確認してください。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
/* 省略 */

    if(LTE_EVENT_ERROR == event_type)
    {
        /* Display API ID and Error code when err
        uint16_t err_code;
        LTE_ERROR_DECODE(&err_code, p_data);
        SEGGER_RTT_printf(0, "Error Response\n");
        SEGGER_RTT_printf(0, "API ID: %d, Error Code: %d\n", api_id, err_code);
    }

/* 省略 */
```

LTE_EVENT_ERROR のイベント通知

LTE_ERROR_DECODE で
エラーコードを解析

図 3.5 LTE_EVENT_ERROR のイベント通知 (hal_entry.c)

- RYZ014A に送信した AT コマンドがタイムアウトした場合：
  - “event_type”に”LTE_EVENT_TIMEOUT_ERROR”が設定されます。
  - “p_data”にはデータは設定されていません。
  - タイムアウトが発生した場合、多くの場合 RYZ014A の動作でエラーが発生したことが想定されます。そのため、初期化を実行することを推奨しています。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
/* 省略 */
    if(LTE_EVENT_TIMEOUT_ERROR == event_type)
    {
        /* Set flag to initialize in ma    loop */
        reinitate_flag = 1;
    }
}
/* 省略 */

void hal_entry(void)
{
/* 省略 */
       if(1 == reinitate_flag)
       {
           /* Initialize when timeout occ
           R_LTE_Init(lte_user_cb);
           reinitate_flag = 0;
       }
```

LTE_EVENT_TIMEOUT_ERROR のイベント通知

フレームワークベースプログラムの初期化

図 3.6 LTE_EVENT_TIMEOUT_ERROR のイベント通知 (hal_entry.c)

- RYZ014A から意図しないタイミングで URC"+SYSSTART"を受信した場合：
  - "event_type"に"LTE_EVENT_FATAL_ERROR"が設定されます。
  - "p_data"にはデータは設定されていません。
  - 本イベントが発生した場合、RYZ014A が再起動したことが想定されます。そのため、初期化からユーザのアプリケーションを再度開始することを推奨します。

【注】RYZ014A の通常の動作において意図しないタイミングで URC"+SYSSTART"を受信することはありません。本ケースは万一の発生に備えてフェールセーフの目的で実装しています。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
/* 省略 */
    if(LTE_EVENT_FATAL_ERROR == event_type)
    {
        /* Set flag to initialize in main loop */
        reinitialize_flag = 1;
    }
}
/* 省略 */

void hal_entry(void)
{
/* 省略 */
       if(1 == reinitialize_flag )
       {
           /* Initialize to restart user application */
           R_LTE_Init(lte_user_cb);
           reinitialize_flag  = 0;
       }
```

LTE_EVENT_FATAL_ERROR のイベント通知

フレームワークベースプログラムの初期化

図 3.7 LTE_EVENT_FATAL_ERROR のイベント通知 (hal_entry.c)

- RYZ014A から URC を受信した場合：
  - “event_type”に”LTE_EVENT_RCVURC”が設定されます。
  - “p_data”には受信した URC の文字列データが設定されます。URC に合わせて処理を実行してください。受信した URC のデータサイズが” LTE_DATA_STR_SIZE”を超過する場合、超過した分のデータは破棄され、前半部分のデータが登録されます。通知される文字列データのサイズは”data_len”に設定されています。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
/* 省略 */

    if(LTE_EVENT_RCVURC == event_type)
    {
        /* Receive message from MQTT server when RYZ014A received subscribe
notification */
        const uint8_t str_onmessage[] = "+SQNSMQTTONMESSAGE";
        if(0 == memcmp(p_data, str_onmessage, (sizeof(str_onmessage) - 1)))
        {
            uint8_t rcv_id = 0;
            char * ptr;
            uint8_t msg_count = 0;


            /* Display subscribed notification */
            SEGGER_RTT_printf(0, "MQTT MESSAGE NOTIFY\n");
            SEGGER_RTT_printf(0, "MESSAGE: %s\n",p_data);

            /* Get message ID from received URC string data */
            ptr = strtok((char *)p_data,",");
            while(ptr != NULL)
            {
                ptr = strtok(NULL,",");
                if(ptr != NULL)
                {
                    msg_count++;
                    if(2 == msg_count)
                    {
                        mqtt_rcvdata_len = (uint8_t )atoi(ptr);
                    }
                    if(4 == msg_count)
                    {
                        rcv_id = (uint8_t )atoi(ptr);
                    }
                }
            }
            /* request message receive */
            R_LTE_MQTT_RcvMessage(str_MQTT_topic, rcv_id);
        }
    }
/* 省略 */
```

LTE_EVENT_RCVURC のイベント通知

受信 URC の確認
“+SQNSMQTTONMESSAGE”の場合、処理を実行

URC のパラメータを解析

解析したパラメータを使用して
AT コマンド API を呼び出し

図 3.8 LTE_EVENT_RCVURC のイベント通知 (hal_entry.c)

3.4 ユーザ固有値の設定

本サンプルアプリケーションをベースにしてアプリケーションを開発する場合、ユーザが使用する RA MCU に応じて一部の設定値を変更します。AT コマンドマネジメントフレームワークではこれらのユーザ固有の設定値を設定するプログラムを「r_lte_user_config.h」に定義しています。ユーザはこのファイルを変更することで環境にあった設定で AT コマンドマネジメントフレームワークを使用することができます。ここでは設定できる値について説明します。

RYZ014A の各端子に対応する端子を指定する設定です。ホスト MCU として使用するボードを変更するなどの場合にご確認ください。

表 3.3 RYZ014A の端子機能設定

定義名	デフォルト値	説明
RYZ014A_RESET_PIN	BSP_IO_PORT_04_PIN_04	RYZ014A のリセット端子に対応する端子。 デフォルト値は EK-RA6M5 の PMOD2 に対応した値が設定されている。
RYZ014A_RESET_ENABLE	BSP_IO_LEVEL_HIGH	RYZ014A のリセット端子を有効化するための信号設定。 デフォルト値は Low→High に設定するための値が設定されている。
RYZ014A_RESET_DISABLE	BSP_IO_LEVEL_LOW	RYZ014A のリセット端子を無効化するための信号設定。 デフォルト値は High→Low に設定するための値が設定されている。

AT コマンドマネジメントフレームワーク内で FSP モジュールを使用するための設定です。RA コンフィグレータを編集する、使用する RA MCU を変更するなどの場合にご確認ください

表 3.4 RYZ014A の FSP モジュール設定

定義名	デフォルト値	説明
RYZ014A_UART_CTRL	g_uart0.p_ctrl	SCI UART モジュールのコントロール構造。 デフォルトでは EK-RA6M5 の PMOD2 で UART を使用する値が設定されている。
RYZ014A_UART_CFG	g_uart0.p_cfg	SCI UART モジュールのコンフィグレーション構造。 デフォルトでは EK-RA6M5 の PMOD2 で UART を使用する値が設定されている。
RYZ014A_TIMER_CTRL	g_timer0.p_ctrl	AGT Timer モジュールのコントロール構造。 デフォルトではチャンネル 0 を使用する値が設定されている。
RYZ014A_TIMER_CFG	g_timer0.p_cfg	AGT Timer モジュールのコンフィグレーション構造。 デフォルトではチャンネル 0 を使用する値が設定されている。

AT コマンドマネジメントフレームワーク内で使用する各種データのサイズ設定です。アプリケーションで使用する AT コマンドや文字列のデータサイズ、使用する MCU のスタックサイズに合わせて変更してください。

表 3.5 AT コマンド送信待機リストのサイズ設定

定義名	デフォルト値	説明
LTE_ATC_STR_SIZE	100	AT コマンドの文字列の最大長。
LTE_DATA_STR_SIZE	100	RYZ014A から受信するデータの最大長。 受信するデータがこのサイズを超過する場合、超過する分のデータは破棄される。
LTE_ATC_LIST_SIZE	6	送信待機リストに登録することのできる AT コマンドの数。 デフォルト値は本サンプルアプリケーションの AT コマンド API で登録される最大数に設定されている。

3.5 フレームワーク内で使用される FSP モジュール

AT コマンドマネジメントフレームワークでは FSP モジュールを使用して機能を実装しています。FSP モジュールはコードだけでなく RA コンフィグレータで設定しています。ここでは AT コマンドマネジメントフレームワークで使用する FSP モジュールについて利用方法や設定方法について説明します。

3.5.1 SCI UART モジュール

AT コマンドマネジメントフレームワークでは SCI UART モジュールを使用して RYZ014A とホスト MCU の間の UART 通信を実装しています。

ホスト MCU から RYZ014A へ AT コマンドを送信するとき、SCI UART モジュールの書き込み関数(R_SCI_UART_Write)を使用しています。アプリケーションから AT コマンド API を呼び出した後、フレームワーク内の送信待機リストに一連の AT コマンドが登録されます。送信待機リストの先頭から書き込み関数を使用して順番に送信されます。

RYZ014A からホスト MCU へレスポンスを送信するとき、SCI UART モジュールのコールバック関数を使用してデータを受信します。このコールバック関数では 1 文字ずつ文字データを受信し、フレームワーク内のリングバッファに格納されます。リングバッファに格納された文字データは R_LTE_Execute 関数の呼び出しのたびに 1 文字ずつ処理されます。

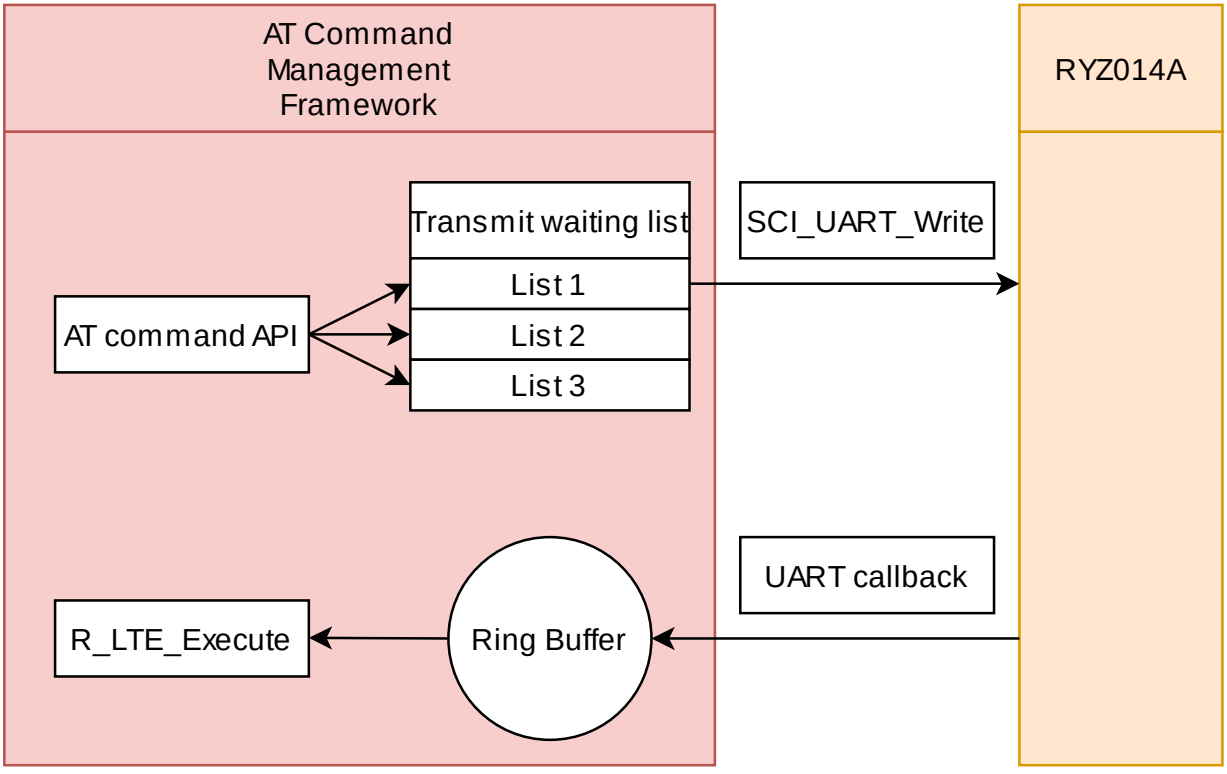

図 3.9 SCI UART モジュールの利用

3.5.2　AGT Timer モジュール

AT コマンドマネジメントフレームワークでは AGT Timer モジュールを使用してタイムアウト機能を実装しています。AT コマンドを送信した後レスポンスを受け取るまでに 60 秒経過するとタイムアウトが発生するように実装しています。

AT コマンドを送信するタイミングでタイマーを開始します。このタイマーは comp_msg に指定したレスポンスを受信したとき、もしくはエラーレスポンスを受信したときに停止します。AT コマンド送信後、一定時間レスポンスを受け取れない場合、タイムアウトが発生したとしてタイマーのコールバック関数がフレームワーク内で呼び出されます。コールバック関数が呼び出された後、R_LTE_Execute 関数でタイムアウトが発生したことをアプリケーションに通知するためにユーザのコールバック関数を呼び出します。

本サンプルアプリケーションでは AT コマンド送信から 60 秒経過した後にタイムアウトが発生するように設定しています。タイムアウトが発生するまでの時間は RA コンフィグレータで設定しています。タイムアウト発生までの時間を変更する場合は RA コンフィグレータから変更してください。

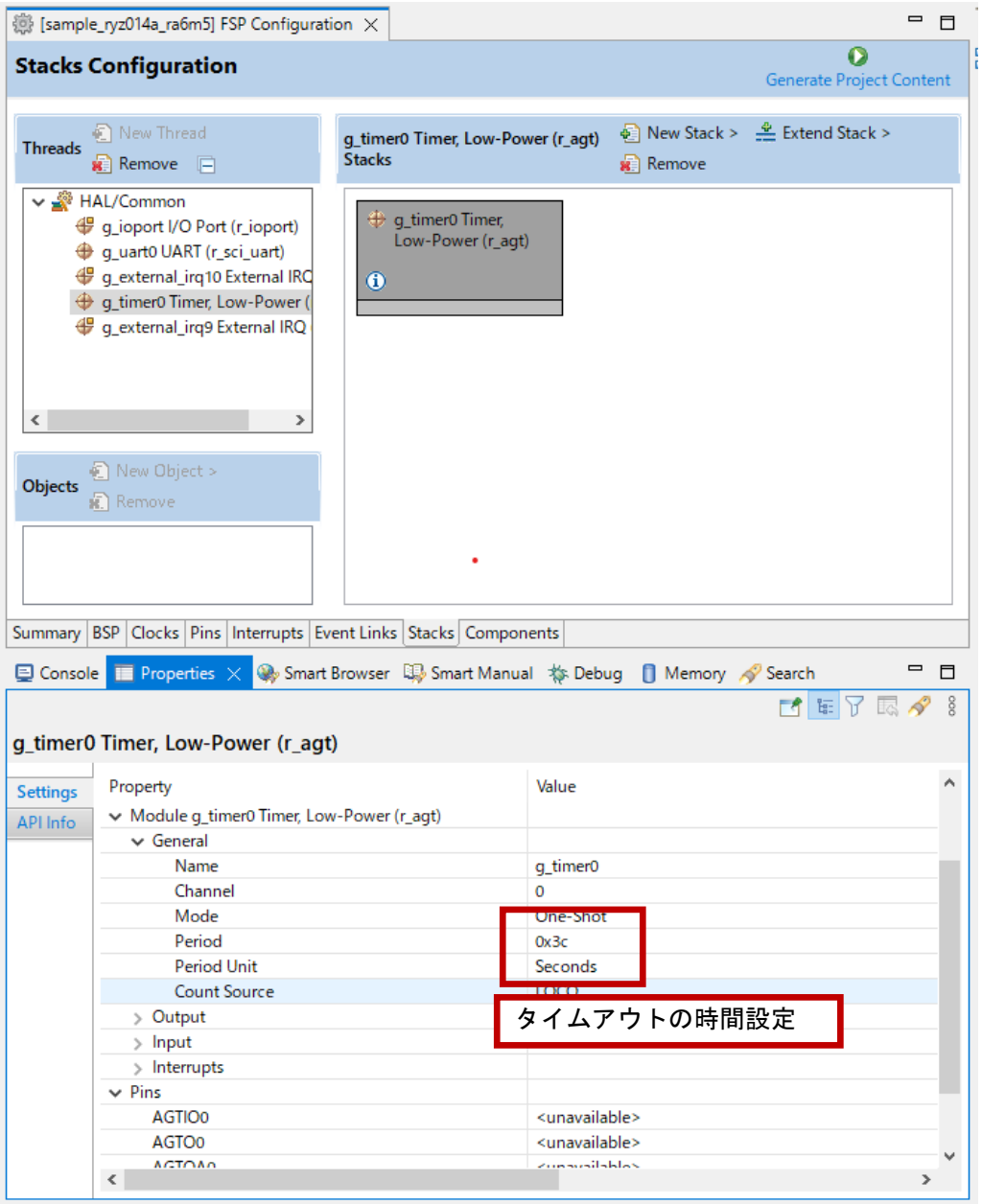

図 3.10 AGT Timer モジュールの設定

4. ATコマンドマネジメントフレームワークを利用したアプリケーション開発

ATコマンドマネジメントフレームワークはユーザのアプリケーション開発のベースとして利用されることを想定しています。ATコマンドマネジメントフレームワークを使用することでRYZ014AとホストMCUの通信を効率的に実装することができます。ここでは本サンプルアプリケーションを例にアプリケーションを開発する方法について説明します。

4.1 アプリケーション開発の概要

ATコマンドマネジメントフレームワークはフレームワーク内にAPIを効率的に追加実装できる仕様になっています。フレームワーク内に実装したAPIをアプリケーションプログラムで呼び出し、ユーザの求める動作を実現します。本サンプルアプリケーションでは以下のファイルで動作を実現しています。

- アプリケーションプログラム：
  - hal_entry.c
- フレームワークベースプログラム：
  - r_lte_ryz014a.c
  - r_lte_ryz014a.h
  - r_lte_user_config.h

フレームワークベースプログラムに実装されているAPIはマネジメントAPIとATコマンドAPIの2種類に分類されます。

マネジメントAPI

マネジメントAPIはRYZ014Aとのやり取りを管理するためのAPIです。アプリケーションプログラムの適切な場所に実装する必要があります。また、ユーザはアプリケーション開発の際に変更する必要はありません。

マネジメントAPIには以下の2つのAPIが実装されています。

- R_LTE_Init
  フレームワークベースプログラムの初期化を行う関数です。本関数ではFSPモジュールの初期化やRYZ014Aのハードウェアリセットなどが実行されます。RYZ014Aは初期化が完了し、ATコマンドを受け付けることが可能になったときに”＋SYSSTART”のURCを送信します。本関数では”+SYSSTART”の後、詳細なエラーレスポンスを受け取るためのATコマンド”AT+CMEE=1”を送信します。すべてのATコマンドの実行が完了した後、引数に指定したコールバック関数にAPI_ID = ”LTE_API_INIT”のイベントが通知されます。この関数はフレームワークに実装されたAPIの中で最初に実行して下さい。
- R_LTE_Execute
  RYZ014Aから受信したデータを保持・解析し、データに合わせてコールバック関数を呼び出したり、ATコマンドを送信したりする関数です。本関数は呼び出す度にリングバッファに格納された1文字ずつ処理を行うのでメインループ内で繰り返し呼び出す必要があります。

```
void hal_entry(void)
{
    SEGGER_RTT_printf(0, "PROGRAM START\n");

    /* SW interrupt driver open */
    R_ICU_Exter                                    l, g_external_irq10.p_cfg);
    R_ICU_Exter                                    , g_external_irq9.p_cfg);

    /* Initialize frame ork-based program and register callback function */
    R_LTE_Init(lte_user_cb);

    while(1)
   {
        /* Execute variable process in framework-based program */
       R_LTE_Execute();

/* 省略 */
   }
 }
```

R_LTE_Init を最初に呼び出す

R_LTE_Execute をメインループで繰り返しに呼び出す

図 4.1 マネジメント API の実装 (hal_entry.c)

AT コマンド API

AT コマンド API を呼び出すことで実行したい動作に必要な一連の AT コマンドが送信待機リストに追加されます。登録された AT コマンドは RYZ014A からのレスポンスに対応して順次送信されます。一連の AT コマンドの実行結果はコールバック関数でアプリケーションに通知されます。ユーザは望む順番で AT コマンド API を呼び出したり、コールバック関数に対応した処理を実装したりすることでアプリケーションを開発します。またユーザは、自身で新たな AT コマンド API を追加し、本サンプルアプリケーションでは利用していない AT コマンドを利用することが可能です。

```
void lte_user_cb(uint16_t event_type, uint16_t api_id, uint16_t data_len, uint8_t *
p_data)
{
    if(LTE_EVENT_API_COMPLETE == event_type)
    {
        switch (api_id)
        {
            case LTE_API_INIT:
            {
                /* Configure operation mo    after initiation complete */
                SEGGER_RTT_printf(0,    INIT COMP\n");
                R_LTE_OM_Config(str_PDP_type, str_PDP_APN, str_LTE_bandlist);
            } break;

            case LTE_API_OM_CONFIG:
            {
                /* Connect to network after configuration of operation mode
complete */
                SEGGER_RTT_printf(0, "OM COMFIG COMP\n");
                R_LTE_NWK_Connect(1);
            } break;

/* 省略 */
```

1. AT コマンド API を呼び出す

2. コールバック関数に結果が通知される

3. 次の AT コマンド API を呼び出す

図 4.2 AT コマンド API の実装 (hal_entry.c)

4.2 AT コマンド API の追加

本フレームワークではユーザのアプリケーションに合わせて AT コマンド API を追加することを想定しています。ここでは本サンプルアプリケーションに実装されている AT コマンド API の解説と、新しい AT コマンド API の実装方法について説明します。

AT コマンド API を追加するときは以下の手順で追加します。

1. API ID と関数のプロトタイプ宣言の追加

   追加した AT コマンド API がコールバック関数で識別できるように API ID を追加します。また AT コマンド API がアプリケーションプログラムから実行できるようにヘッダファイル(r_lte_ryz014a.h)にプロトタイプ宣言を追加します。

```
typedef enum
{
    LTE_API_NO_CURRENT_API = 0,
    LTE_API_OM_CONFIG,
    LTE_API_NWK_CONNECT,
    LTE_API_NWK_DISCONNECT,
    LTE_API_MQTT_CONNECT,
    LTE_API_MQTT_DISCONNECT,
    LTE_API_MQTT_SUBSCRIBE,
    LTE_API_MQTT_PUBLISH,
    LTE_API_MQTT_RCVMESSAGE,
    LTE_API_SEC_CERTIFICATEADD,
    LTE_API_SEC_CERTIFICATEREMOVE,
    LTE_API_SEC_PRIVATEKEYADD,
    LTE_API_SEC_PRIVATEKEYREMOVE,
    LTE_API_NWK_CONNECTIONCONFIG,

    LTE_API_INIT = 0xff,
} e_lte_api_id_t;
```

図 4.3 本サンプルアプリケーションの API ID (r_lte_ryz014a.h)

2. AT コマンド API の実装

AT コマンド API の実態をソースファイル(r_lte_ryz014a.c)に実装します。本サンプルアプリケーションの AT コマンド API は以下の構成で実装されています。

引数の確認・実行中の AT コマンド API の確認

引数にポインタがある場合、NULL を指定していないことを確認します。また”gs_process_api”を確認して他の AT コマンド API が実行中でないことを確認します。実行中の場合、AT コマンド送信待機リストを変更すると、実行中の AT コマンド API が正常に動作することができないため何も実行せずにエラー”LTE_ERR_IN_PROCESS”を返却します。その後、本 AT コマンド API が実行中であることを示すため、”gs_process_api”に API_ID を登録します。

```
e_lte_err_t R_LTE_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{
    /* Check Argument and current state */
    if((NULL == p_pdp_type) || (NULL == p_pdp_apn) || (NULL == p_bandlist))
    {
        return LTE_ERR_POINTER_NULL;
    }

    if(LTE_API_NO_CURRENT_API != gs_process_api)
    {
        return LTE_ERR_IN_PROCESS;
    }

    /* Clear ATC list and set processing API ID */
    ryz014a_clear_atc_list();
    gs_process_api = LTE_API_OM_CONFIG;

/* 省略 */
```

引数の確認

実行中の AT コマンド API の確認

実行中の AT コマンド API の登録

図 4.4 R_LTE_OM_Config の引数確認・実行中の AT コマンド API の確認 (r_lte_ryz014a.c)

送信待機リストへ AT コマンドの登録

送信待機リスト”gs_atc_list”に AT コマンドを文字列として登録します。送信待機リスト”gs_atc_list”には 1 つの AT コマンドに対して以下を登録する必要があります。

- atcommand：
  実行したい AT コマンドの文字列を登録します。文字列の長さは”atcommand_size”に登録してください。登録できる文字列の最大長は 256 文字です。さらに大きい AT コマンド文字列を使用する場合はユーザ設定ファイル(r_lte_user_config.h)の”LTE_ATC_STR_SIZE”を変更してください。
- data：
  AT コマンドによって処理したいデータ文字列のアドレスを登録するポインタです。データを送信する AT コマンドなどで必要になります。ここに登録されたデータ文字列は”> ”のレスポンスに対応して送信されます。文字列の長さは”data_size”に登録してください。本ポインタに登録する文字列の実体はアプリケーションに実装するすることを想定しています。
- comp_msg：
  実行したい AT コマンドが完了したと見なせるレスポンス文字列を登録します。”OK”もしくは URC を指定して下さい。文字列の長さは”comp_msg_size”に登録してください。comp_msg で指定した文字列ト前方一致するレスポンスを受信した後、送信待機リストの次に登録されている AT コマンドが送信されます。”OK”と URC が連続で送信される場合、最後に送信されるレスポンスを登録してください。また送信待機リストに登録する一連の AT コマンドの最後の comp_msg によってコールバック関数で通知されるデータが変化します。詳しくは「3.3 コールバック関数」を参照してください。
- data_exist_flag：
  送信したい AT コマンドが設定されていることを示すフラグです。R_LTE_Execute 関数ではこの値を確認することで AT コマンドが登録されていることを確認します。AT コマンドを設定した場合は”1”を設定してください。

送信待機リスト”gs_atc_list”は固定長の配列によって文字列データを保持します。そのため、登録する文字列データが送信待機リストに登録できる最大長を超過するとバッファオーバーフローによるエラーが発生する可能性があります。入力されるデータサイズが登録できる文字列の最大長を超過することが予想される場合、データサイズを確認するための処理を追加してください。

```
e_lte_err_t R_LTE_MQTT_Publish(uint8_t* p_topic, uint16_t length, uint8_t* p_message)
{
/* 省略 */

    /* Set AT command to ATC list */
    gs_atc_list[0].atcommand_size = (uint16_t)snprintf((char*)gs_atc_list[0].atcommand,
LTE_ATC_STR_SIZE, "AT+SQNSMQTTPUBLISH=%s,\"%s\",%s,%d\r", "0",p_topic,"0",length);
    gs_atc_list[0].comp_msg_size = (uint16_t)snprintf((char*)gs_atc_list[0].comp_msg,
LTE_ATC_STR_SIZE, "%s", "OK");
    gs_atc_list[0].data_exist_flag = 1;


    gs_atc_list[0].data = p_message;
    gs_atc_list[0].data_size = length;




    if(gs_atc_list[0].atcommand_size > LTE_ATC_STR_SIZE)
    {
        ryz014a_clear_atc_list();
        return LTE_ERR_DATASIZE_OVERFLOW;
    }
```

送信待機リストの先頭に以下を追加

atcommand =
“AT+SQNSMQTTPUBLISH=0,"[p_topic]“,0,[length] ”
comp_msg = “OK”
data_exist_flag = 1

data には送信したいデータのアドレスを設定

引数を送信待機リストに登録するため、
データサイズの確認

図 4.5 R_LTE_MQTT_Publish の AT コマンドの登録 (r_lte_ryz014a.c)

先頭の AT コマンドの送信

登録した送信待機リストの先頭の AT コマンドを送信します。以降の AT コマンドの送信はレスポンスに対応して R_LTE_Execute 関数で行われます。

```
e_lte_err_t R_LTE_OM_Config(uint8_t* p_pdp_type, uint8_t* p_pdp_apn, uint8_t*
p_bandlist)
{

/* 省略 */

    /* Send first AT command from ATC list */
    ryz014a_transmit_atc_list(LTE_TRANSMIT_ATCOMMAND);

      return LTE_SUCCESS;
```

送信待機リストの先頭の AT コマンドを送信

図 4.6 R_LTE_OM_Config の先頭の AT コマンドの送信 (r_lte_ryz014a.c)

4.2.1 データ受信を伴う AT コマンド API

本サンプルアプリケーションに実装されている R_LTE_MQTT_RcvMessage 関数のように AT コマンド送信後に任意のデータ受信を行う AT コマンド API を実装するにはフレームワーク内のグローバル変数の書き換えが必要になります。

データ受信を行う場合、グローバル変数”gs_ryz014a_receive_size”と”gs_ryz014a_receive_flag”を変更する必要があります。”gs_ryz014a_receive_size” には受信したいデータのサイズを、”gs_ryz014a_receive_flag”はマクロ”LTE_RCV_DATA_FLAG_ON”を設定します。

```
e_lte_err_t R_LTE_MQTT_RcvMessage(uint8_t* p_topic, uint8_t message_id, uint16_t
message_size)
{
                                                    受信サイズとフラグを設定
/* 省略 */

    /* Set receive flag and size for data receive operation */
    gs_ryz014a_receive_size = message_size;
    gs_ryz014a_receive_flag = LTE_RCV_DATA_FLAG_ON;

/* 省略 */
```

図 4.7 R_LTE_MQTT_RcvMessage のグローバル変数設定 (r_lte_ryz014a.c)

この時に受信したデータは、アプリケーションへコールバック関数で通知されます。コールバック関数はデータの後に RYZ014A から送信される「OK」の応答を受け取ったタイミングで呼び出されます。

※注意：受信データ内の「\r」または「\n」は RYZ014A 内で「\r\n」に変換されてホスト MCU に送信されます。そのため、受信データの内容やサイズの一部が変更される場合があります。

4.3 エラー発生処理のガイドライン

通信制御のシステムでは、通信コントローラの制御やネットワーク動作における様々なエラー発生を想定してアプリケーションを開発する必要があります。ここではその検出と処理について、本 AT コマンドマネジメントフレームワークを使用してアプリケーション開発を行う場合のガイドラインを示します。実際には応用製品への要求事項によって処理は変わりますので、参考情報の扱いでお願いします。

UART 通信及び RYZ014A の動作エラー

不具合の状況	フレームワークの挙動	アプリケーションの対応
意図せず RYZ014A が再起動される	“+SYSSTART”を受信する。意図しない”+SYSSTART”を受信するため、アプリケーションにコールバック関数で通知する。	イベント ”LTE_EVENT_FATAL_ERROR” でコールバック関数が呼び出される。本イベントに R_LTE_Init 関数を使用して初期化することが推奨される。
RYZ014A からホスト MCU への UART 通信でビットエラーあるいは文字の受信ミスが起きる	文字列が comp_msg で指定した文字列と合致しない場合もしくは文字列が”\n”で終了しない場合、タイムアウトが発生し、コールバック関数でアプリケーションに通知する。	イベント ”LTE_EVENT_TIMEOUT_ERROR”でコールバック関数が呼び出される。本イベントに対しては R_LTE_Init 関数を使用して初期化することが推奨される。
	受信文字列が comp_msg で指定した URC と前方一致する場合、コールバック関数にてアプリケーションに通知される。	イベント ”LTE_EVENT_RCVURC”でコールバック関数が呼び出される。P_data に受信した文字列が登録されているため、データを確認する。
ホスト MCU から RYZ014A への UART 通信でビットエラーあるいは文字の受信ミスが起きる	送信した文字列に対するレスポンスがない場合、タイムアウトが発生し、コールバック関数でアプリケーションに通知する。	イベント ”LTE_EVENT_TIMEOUT_ERROR”でコールバック関数が呼び出される。本イベントに対しては R_LTE_Init 関数を使用して初期化することが推奨される。
	送信した AT コマンドの一部が間違っている場合、エラーのレスポンスを受信する。エラーレスポンスを受信後、コールバック関数でアプリケーションに通知する。	イベント”LTE_EVENT_ERROR” でコールバック関数が呼び出される。エラーコード” LTE_CME_ERR_OPERATION_NOT_SUPPORTED”(0x04)が p_data にて通知されるので対応した処理を追加する。
MCU の送信と RYZ014A の送信タイミングが重なり、RYZ014A が期待した動作をしない	動作が停止することでタイムアウトが発生する。タイムアウト発生時にコールバック関数でアプリケーションに通知する。	イベント ”LTE_EVENT_TIMEOUT_ERROR”でコールバック関数が呼び出される。本イベントに対しては R_LTE_Init 関数を使用して初期化することが推奨される。
RYZ014A からの CTS が長時間イネーブルにならない	長時間 RYZ014A からレスポンスを受け取ることができず、タイムアウトが発生する。タイムアウト発生時にコールバック関数でアプリケーションに通知する。	イベント ”LTE_EVENT_TIMEOUT_ERROR”でコールバック関数が呼び出される。本イベントに対しては R_LTE_Init 関数を使用して初期化することが推奨される。

ネットワーク通信エラー

不具合の状況	フレームワークの挙動	アプリケーションの対応
電波状況の悪化などでネットワークが切断される	“+CEREG”の URC を受信する。コールバック関数でアプリケーションに通知される。	イベント ”LTE_EVENT_RCVURC”でコールバック関数が呼び出される。URC のパラメータを確認し、対応した処理を実行する。URC のパラメータは AT コマンドマニュアルを確認する。
ネットワークに接続しようとしたが APN の間違いなどによって接続できなかった	“+CEREG”の URC を受信する。コールバック関数でアプリケーションに通知される。	イベント ”LTE_EVENT_RCVURC”でコールバック関数が呼び出される。URC のパラメータを確認し、対応した処理を実行する。URC のパラメータは AT コマンドマニュアルを確認する。
その他、何らかの理由でコネクションが切断される。	“+SQNSH”の URC を受信する。コールバック関数でアプリケーションに通知される。	イベント ”LTE_EVENT_RCVURC”でコールバック関数が呼び出される。URC のパラメータを確認し、対応した処理を実行する。URC のパラメータは AT コマンドマニュアルを確認する。

通信状況は URC”+CEREG”などによって通知されます。通信の切断に対応したアプリケーション実装の一例として、ここでは本サンプルアプリケーションで実装されている MQTT 通信に復帰する処理について説明します。

- URC”+CEREG: 80”または”+CEREG: 4”を受信：
  - 一時的にネットワークから切断されたときに通知される URC です。再度ネットワークに接続しようとしているため、電波の状況などが改善すれば、AT コマンド API を実行することなくネットワークに再接続することができます。この時、RYZ014A 内で MQTT 通信は保持されているため、ネットワークに再接続した際に R_LTE_MQTT_Connect 関数を実行することなく MQTT 通信を再開することができます。
- URC”+CEREG: 0”を受信：
  - ネットワークから切断されたときに通知される URC です。再度ネットワークに接続しようとしているため、電波の状況などが改善すれば、AT コマンド API を実行することなくネットワークに再接続することができます。再度ネットワークに接続したときに URC” +SQNSMQTTONCONNECT: 0,0”が通知されます。R_LTE_MQTT_Connect 関数を実行することなく MQTT 通信を再開することができますが、MQTT サーバへの接続がリセットされているため、R_LTE_MQTT_Subscribe 関数などは再度実行する必要があります。
- URC” +SQNSMQTTONCONNECT: 0,-7”を受信
  - ネットワークから切断されて一定時間たった後 MQTT 通信も切断したときに通知される URC です。ネットワークに再接続しても MQTT 通信は保持されていないので再度 R_LTE_MQTT_Connect 関数を実行する必要があります。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022/8/31	-	新規発行

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。