R01AN7184JJ0100
Rev.1.00
2023.12.08

RX ファミリ

RYZ014A Cellular モジュール制御モジュール

Firmware Integration Technology

ポーティングガイド

要旨

本アプリケーションノートでは、Firmware Integration Technology (FIT) に準拠した RYZ014A Cellular モジュール制御 FIT モジュール (Rev.1.11、ドキュメント番号：R01AN6324) をベースに、新規に AT コマンドを追加する方法、RYZ014A Cellular モジュール以外の通信モジュールを制御する方法、およびサポート対象外の RX ファミリマイコンで使用する方法について説明します。

以降、RYZ014A Cellular モジュール制御 FIT モジュールのソフトウェアを総じて“RYZ014A Cellular FIT モジュール”または“Cellular FIT モジュール”と称します。

RYZ014A Cellular FIT モジュールの詳細につきましては、関連ドキュメントをご確認ください。

関連ドキュメント

[1] Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
[2] RX ファミリ ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
[3] e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)
[4] CS+に組み込む方法 Firmware Integration Technology (R01AN1826)
[5] Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
[6] RX ファミリ RYZ014A Cellular モジュール制御モジュール Firmware Integration Technology (R01AN6324)
[7] RX ファミリ SCI モジュール Firmware Integration Technology (R01AN1815)
[8] RX ファミリ BYTEQ モジュール Firmware Integration Technology (R01AN1683)
[9] RX ファミリ IRQ モジュール Firmware Integration Technology (R01AN1668)
[10] RYZ014 Modules User's Manual: AT Command (R11UZ0110)
[11] RYZ014 Module System Integration Guide (R19AN0074)
[12] FreeRTOS Kernel API reference (リンク)
[13] FreeRTOS Cellular Interface Library (リンク)

目次

1. 概要

1.1 RYZ014A Cellular FIT モジュールとは

RYZ014A Cellular FIT モジュールは、Cellular モジュールとの UART 通信をサポートします。RYZ014A Cellular FIT モジュールの詳細については、関連ドキュメント[6]を参照してください。

2. 使用されている各資源について

2.1 ハードウェア

ご使用になるマイコンが、以下の機能をサポートしている必要があります。

- シリアル通信
- I/O ポート
- IRQ
- 割り込み要因として設定できる 1 つ、または複数の IRQ 入力端子

2.2 ソフトウェア

RYZ014A Cellular FIT モジュールは、以下のソフトウェアおよびリアルタイム OS に依存しています。

- ボードサポートパッケージモジュール FIT　(以降、r_bsp と記載します)
- SCI モジュール FIT　(以降、r_sci_rx と記載します)
- バイト型キューバッファ (BYTEQ) モジュール FIT　(以降、r_byteq と記載します) ※1
- IRQ モジュール FIT　(以降、r_irq_rx と記載します)
- FreeRTOS

※1. r_sci_rx で使用。RYZ014A Cellular FIT モジュールでは直接使用していない。

2.2.1 r_bsp

RYZ014A Cellular FIT モジュールは、以下の r_bsp の API を使用しています。r_bsp は、マイコンのクロックや割り込みの設定などの機能をサポートするモジュールです。

r_bsp を使用しない場合は、適切な処置に置き換えてください。API の詳細については、関連ドキュメント[2]を参照してください。RYZ014A Cellular FIT モジュール内での用途については、7.5 節を参照してください。

- R_BSP_NOP()
- R_BSP_RegisterProtectDisable()
- R_BSP_RegisterProtectEnable()

2.2.2 r_sci_rx

RYZ014A Cellular FIT モジュールは、r_sci_rx の以下の API を使用しています。r_sci_rx は、RYZ014A Cellular モジュールと UART 通信するためのモジュールです。RYZ014A およびマイコン間で AT コマンドやデータの送受信をするために使用しています。

r_sci_rx を使用しない場合は、適切な処置に置き換えてください。API の詳細については、関連ドキュメント[7]を参照してください。RYZ014A Cellular FIT モジュール内での用途については、7.6 節を参照してください。

- R_SCI_Open()
- R_SCI_Close()
- R_SCI_Send()
- R_SCI_Receive()
- R_SCI_Control()

r_sci_rx では、r_byteq を使用しています。r_byteq は r_sci_rx の送受信データをバッファリングするために使用しています。r_sci_rx を使用しない場合、r_byteq の API を使用する処理の置き換えも必要となります。

2.2.3 r_irq_rx

RYZ014A Cellular FIT モジュールは、r_irq_rx の以下の API を使用しています。r_irq_rx は、割り込み要求 (IRQ) を使用して、外部端子割り込みからのイベントを処理します。r_irq_rx は、RYZ014A モジュールの RING 端子からの出力信号による通知を、割り込みとして処理するために使用しています。RING 信号による通知は、Cellular モジュールの PSM モード時のデータ受信要求をマイコンが検出するために使用します。

r_irq_rx を使用しない場合は、適切な処置に置き換えてください。r_irq_rx の詳細については、関連ドキュメント[9]を参照してください。RYZ014A Cellular FIT モジュール内での用途については、7.7 節を参照してください。

- R_IRQ_Open()
- R_IRQ_Close()

2.2.4 FreeRTOS

RYZ014A Cellular FIT モジュールは、FreeRTOS 上でのみ動作し、以下の FreeRTOS の API を使用しています。

FreeRTOS を使用しない場合、適切な処置に置き換えてください。FreeRTOS の詳細については、関連ドキュメント[12]を参照してください。RYZ014A Cellular FIT モジュール内での用途については、7.8 節を参照してください。

- xTaskCreate()
- vTaskDelay()
- xTaskGetTickCount()
- vTaskSuspend()
- vTaskDelete()
- xEventGroupCreate()
- xEventGroupWaitBits()
- xEventGroupSetBitsFromISR()
- xEventGroupSync()
- vEventGroupDelete()
- xSemaphoreCreateMutex()
- xSemaphoreTake()
- xSemaphoreGive()
- vSemaphoreDelete()
- pvPortMalloc()
- vPortFree()
- taskENTER_CRITICAL()
- taskEXIT_CRITICAL()

3. RYZ014A Cellular FIT モジュール仕様

3.1 RYZ014A Cellular FIT モジュールの API 動作概要

RYZ014A Cellular FIT モジュールは、RYZ014A Cellular モジュールと UART 通信により AT コマンドの送信、データの受信を実施します。各 API は引数パラメータチェック、AT コマンド発行のためのセマフォの取得、AT コマンドの生成、AT コマンドの実行 (送信)、モジュールからの応答の受信を行っています。

RYZ014A Cellular FIT モジュールの API 動作概要を図 1 に示します。

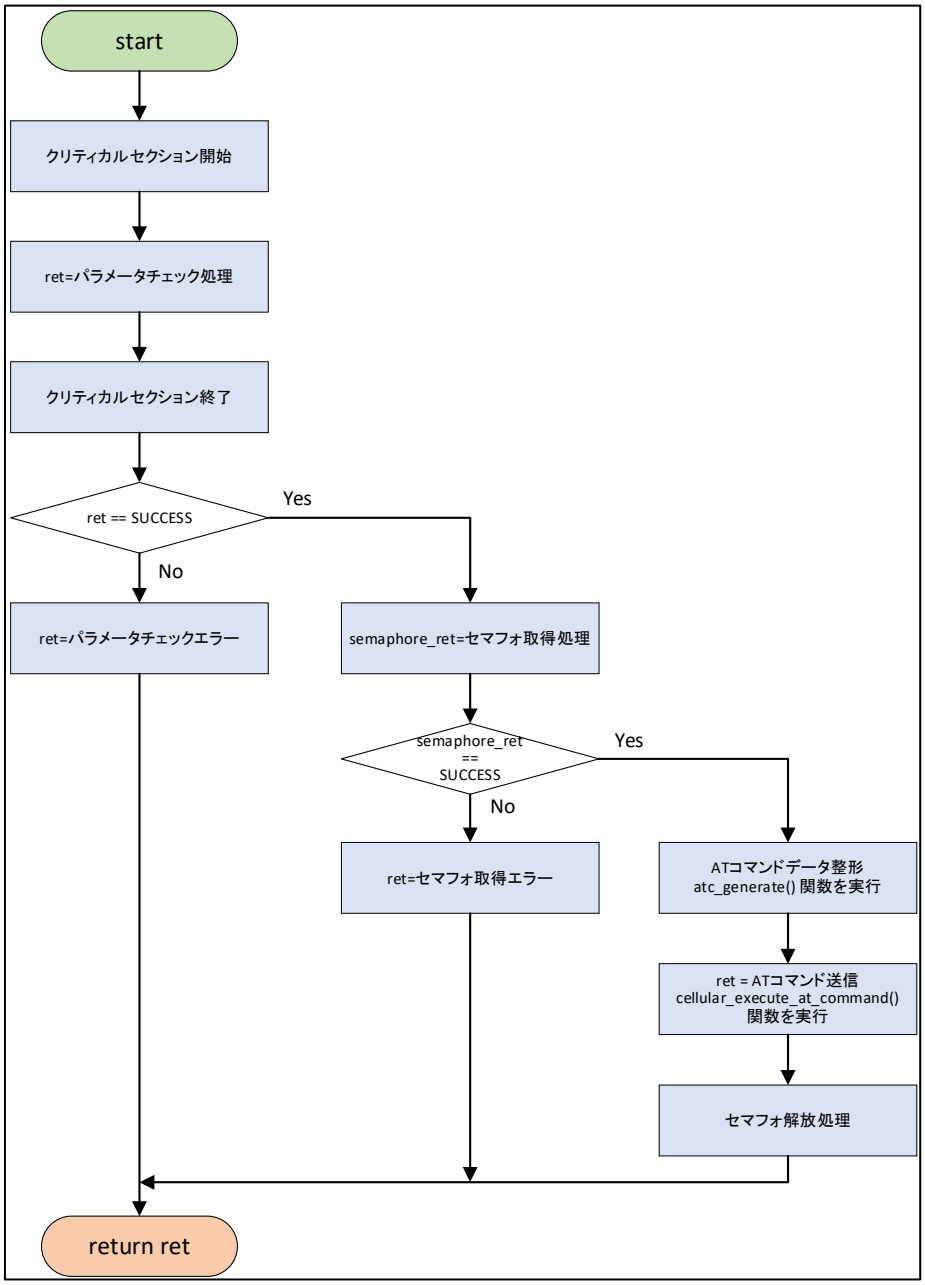

図 1　RYZ014A Cellular FIT モジュールの API 動作概要図

3.1.1 AT コマンド送信関数 cellular_execute_at_command() の動作概要

RYZ014A Cellular FIT モジュールの内部関数 cellular_execute_at_command() の動作概要を図 2 に示します。cellular_execute_at_command() 関数は、Cellular モジュールへ UART 通信で AT コマンドを送信する関数です。送信した AT コマンドに対して応答がない場合に、タイムアウトを検出する処理も実装されています。

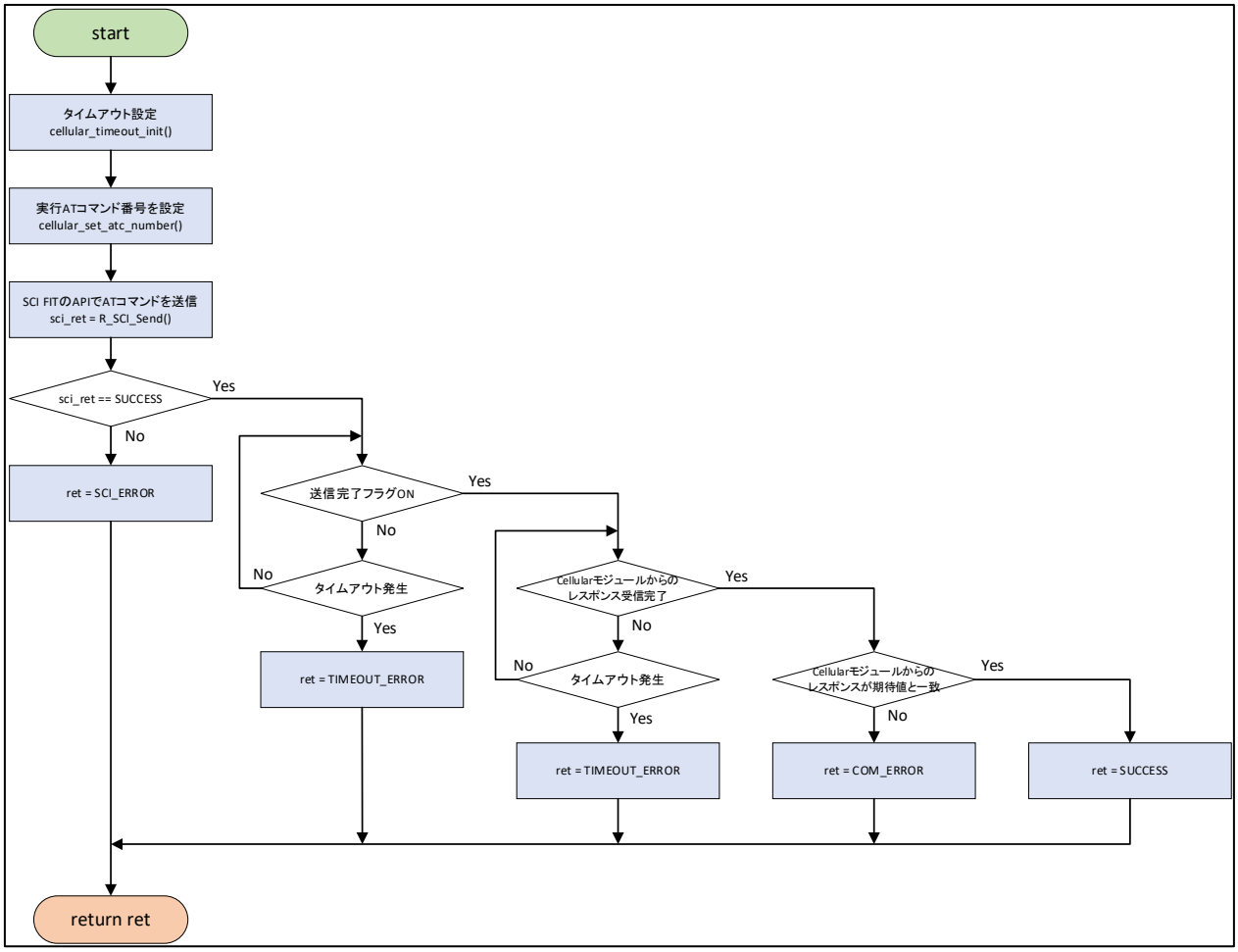

図 2　cellular_execute_at_command()の動作概要図

3.2 AT コマンドの処理について

3.2.1 atc_generate() 関数

RYZ014A Cellular FIT モジュールは、送信する AT コマンドを atc_generate() 関数によって生成します。

AT コマンドを新規追加する場合は、追加する AT コマンドに対応する AT コマンド番号を AT コマンドテーブルへ追加する必要があります。AT コマンドの新規追加方法の詳細については、「4. AT コマンドの新規追加方法について」を参照してください。

atc_generate() 関数の仕様は、以下の通りです。

```
atc_generate(
    uint8_t * const p_command_buff,
    const uint8_t * p_command,
    const uint8_t ** pp_command_arg
)
```

- 第 1 引数 … 送信する AT コマンド文字列を格納するバッファを設定します。
  RYZ014A Cellular FIT モジュールでは、管理構造体 st_cellular_ctrl_t で宣言している p_ctrl->sci_ctrl.atc_buff を使用しています。
- 第 2 引数 … 実行する AT コマンド文字列書式を、AT コマンドテーブルを参照して設定します。
  実行する AT コマンドに対応する AT コマンド番号を使用して、AT コマンドテーブル gp_at_command[] を参照します。例：gp_at_command[ATC_GET_LR_SVN]。
  ryz014_private.h で定義されている、AT コマンドテーブル gp_at_command[] および AT コマンド番号リスト e_atc_list_t を図 3 に示します。ryz014_private.h の詳細は、4.2.1 節を参照してください。
- 第 3 引数 … AT コマンド引数を設定します。
  実行する AT コマンドに合わせて設定します。

```
const uint8_t * const gp_at_command[ATC_LIST_MAX] =
{
    g_ryz014_echo_off,
    g_ryz014_function_level_check,
    g_ryz014_function_level,
    g_ryz014_pin_lock_check,
    g_ryz014_pin_lock_release,
              |
            中略
              |
    g_ryz014_atc_get_lr_svn,
    g_ryz014_write_certificate,
    g_ryz014_erase_certificate,
    g_ryz014_get_certificate,
    g_ryz014_config_ssl_profile,
#if (CELLULAR_IMPLEMENT_TYPE == 'B')
    g_ryz014_config_ssl_socket,
#endif
    g_ryz014_no_command,
};
```

```
typedef enum
{
    ATC_ECHO_OFF          = 0,
    ATC_FUNCTION_LEVEL_CHECK,
    ATC_FUNCTION_LEVEL,
    ATC_PIN_LOCK_CHECK,
    ATC_PIN_LOCK_RELEASE,
                |
              中略
                |
    ATC_GET_LR_SVN,
    ATC_WRITE_CERTIFICATE,
    ATC_ERASE_CERTIFICATE,
    ATC_GET_CERTIFICATE,
    ATC_CONFIG_SSL_PROFILE,
#if (CELLULAR_IMPLEMENT_TYPE == 'B')
    ATC_CONFIG_SSL_SOCKET,
#endif
    ATC_SQNSSENDEXT_END,
    ATC_LIST_MAX
} e_atc_list_t;
```

図 3　第 2 引数で使用する配列 gp_at_command[]および列挙体 e_atc_list_t の定義 (ryz014_private.h)

3.2.2 受信文字列の処理方法

RYZ014A Cellular FIT モジュールでは、Cellular モジュールからレスポンスや URC として送信される文字列の受信処理を cellular_recv_task() 関数で実行しています。

cellular_recv_task() 関数は、Cellular モジュールから送信される文字の受信処理を 1 文字単位で実行します。Cellular モジュールから 1 文字受信する毎に、cellular_recv_task() 関数で受信済みの文字を含む受信文字列を処理します。cellular_recv_task() 関数は、逐次受信文字列に応じた処理番号 e_atc_return_code_t への遷移、および処理番号を使用して関数ポインタテーブル p_cellular_recvtask_api[] の関数を逐次呼び出すことで受信文字列の処理を行います。

関数ポインタテーブル p_cellular_recvtask_api[] の仕様は、以下の通りです。

```
static void (* p_cellular_recvtask_api[]) (
    st_cellular_ctrl_t * p_ctrl,
    st_cellular_receive_t * cellular_receive
)
```

p_cellular_recvtask_api[] のインデックスへ、受信した文字に応じた処理番号 e_atc_return_code_t を設定します。

- 第 1 引数　…　Cellular FIT モジュールの管理構造体ポインタを設定します。
- 第 2 引数　…　Cellular FIT モジュールの受信タスク cellular_recv_task() の管理構造体ポインタを設定します。

```
static void(* p_cellular_recvtask_api[])(st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive) =
{
    cellular_data_send_command,
    cellular_memclear,
    cellular_memclear,
    cellular_memclear,
    cellular_memclear,
    cellular_exit,
          |
        中略
          |
    cellular_get_certificate,
    cellular_response_skip,
    cellular_store_data,
    cellular_response_check,
    cellular_job_check
};
```

図 4　p_cellular_recvtask_api[] の定義 (r_cellular_receive_task.c)

```
typedef enum
{
    CELLULAR_RES_GO_SEND = 0,           // Request for Data Transmission
    CELLULAR_RES_OK,                    // Response is OK
    CELLULAR_RES_ERROR,                 // Response is ERROR
    CELLULAR_RES_NO_CARRIER,            // Response is NO CARRIER
    CELLULAR_RES_CONNECT,               // Response is CONNECT
    CELLULAR_RES_EXIT,                  // Exit error detected (module is automatically restarted)
              |
            中略
              |
    CELLULAR_GET_CERTIFICATE,           // Get Certificate
    CELLULAR_RES_MAX,                   // End of analysis result processing

    CELLULAR_RES_PUT_CHAR,              // Received data storage process
    CELLULAR_RES_CHECK,                 // Receipt confirmation process
    CELLULAR_RES_NONE,                  // No process
} e_atc_return_code_t;
```

図 5　処理番号 e_atc_return_code_t の定義　(cellular_receive_task.h)

3.3 RYZ014A Cellular FIT モジュールのタイムアウト処理

RYZ014A Cellular FIT モジュールは、Cellular モジュールの応答を正常に受信できなかった場合などに処理がデッドロックしないようにするため、タイムアウト処理を持ちます。タイムアウト処理を実行する関数を、表 3.1 に示します。タイムアウト時間は、モジュールにより推奨時間が異なります。RYZ014A の場合は、ネットワークに通信を行わない AT コマンドは 60 秒が推奨値とガイドされています。使用するモジュールに合わせて設定してください。

表 3.1　タイムアウト処理を実行する Cellular FIT モジュールの関数

関数名	種別	タイムアウト時間
R_CELLULAR_SendSocket()	ユーザ向け API 関数	ユーザが API の引数で設定可能
R_CELLULAR_ReceiveSocket()	ユーザ向け API 関数	ユーザが API の引数で設定可能
cellular_take_semaphore()	内部関数	ユーザがスマート・コンフィグレータ(GUI)から設定可能
cellular_execute_at_command()	内部関数	ドライバ内部のマクロ定義で設定可能
cellular_synchro_event_group()	内部関数	ドライバ内部のマクロ定義で設定可能
cellular_pin_reset()	内部関数	ドライバ内部のマクロ定義で設定可能
cellular_power_down()	内部関数	ドライバ内部のマクロ定義で設定可能

3.3.1 R_CELLULAR_SendSocket() 関数のタイムアウト処理

R_CELLULAR_SendSocket() 関数は、ネットワークへ接続中のソケットでデータ送信を実行する API 関数です。

タイムアウト時間は、ユーザが第 5 引数へ設定します。

※ R_CELLULAR_SendSocket() 関数の実行終了までのタイムアウト時間を設定します。
R_CELLULAR_SendSocket() 関数の処理中の AT コマンド送信処理 (cellular_execute_at_command() 関数) には適用されません。

タイムアウト発生時は、返り値で CELLULAR_ERR_MODULE_TIMEOUT を返します。

3.3.2 R_CELLULAR_ReceiveSocket() 関数のタイムアウト処理

R_CELLULAR_ReceiveSocket() 関数は、ネットワークへ接続中のソケットで受信したデータを、マイコンが Cellular モジュールから受信する API です。

タイムアウト時間は、ユーザが第 5 引数へ設定します。

※ R_CELLULAR_ReceiveSocket() 関数の実行終了までのタイムアウト時間を設定します。
R_CELLULAR_ReceiveSocket() 関数の処理中の AT コマンド送信処理 (cellular_execute_at_command() 関数) には適用されません。

タイムアウト発生時は、返り値でタイムアウト発生時点までに受信済みのデータサイズ (単位：byte) を返します。

3.3.3 cellular_take_semaphore() 関数のタイムアウト処理

cellular_take_semaphore() 関数は、FreeRTOS のセマフォ取得時に実行される内部関数です。

タイムアウト時間は、r_cellular_config.h で定義されている CELLULAR_CFG_SEMAPHORE_BLOCK_TIME の値を使用しています。

タイムアウト発生時は、返り値で CELLULAR_SEMAPHORE_ERR_TAKE を返します。

※ タイムアウト時間のデフォルト値は、15000ms です。

3.3.4 cellular_execute_at_command() 関数のタイムアウト処理

cellular_execute_at_command() 関数は、Cellular モジュールへ AT コマンドを送信する内部関数です。

タイムアウト時間は、第 2 引数の値を使用します。

タイムアウト発生時は、返り値で CELLULAR_ERR_MODULE_TIMEOUT を返します。

※ R_CELLULAR_Open() 関数実行時に p_ctrl->sci_ctrl.atc_timeout にタイムアウト時間を格納し、以降はこの値を使用します。タイムアウト時間は、ryz014_private.h で定義されている CELLULAR_COMMAND_TIMEOUT の値を使用します。
※ タイムアウト時間のデフォルト値は、60000ms です。

3.3.5 cellular_synchro_event_group() 関数のタイムアウト処理

cellular_synchro_event_group() 関数はデータ受信処理を行う、cellular_recv_task() 関数と同期処理を行うための内部関数です。

タイムアウト発生時は、返り値でタイムアウト発生時点でのフラグ状態を返します。

※ タイムアウト時間は、r_cellular_private.h で定義されている CELLULAR_TIME_WAIT_TASK_START の値を使用します。
※ タイムアウト時間のデフォルト値は、10000ms です。

3.3.6 cellular_pin_reset() 関数のタイムアウト処理

cellular_pin_reset() 関数は、Cellular モジュールのハードウェアリセットを実行する内部関数です。

タイムアウト時間は、cellular_module_reset.c で定義されている CELLULAR_RESTART_LIMIT の値を使用します。

タイムアウト発生時は、返り値で CELLULAR_ERR_RECV_TASK を返します。

※ タイムアウト時間のデフォルト値は、100s (CELLULAR_RESTART_LIMIT のデフォルト値 100*1000ms) です。

3.3.7 cellular_power_down() 関数のタイムアウト処理

cellular_power_down() 関数は、Cellular モジュールのシャットダウンを実行する内部関数です。

タイムアウト時間は、cellular_power_down.c で定義されている CELLULAR_SHUTDOWN_LIMIT の値を使用します。

タイムアウト発生時は、返り値で CELLULAR_ERR_MODULE_COM を返します。

※ タイムアウト時間のデフォルト値は、50s (CELLULAR_SHUTDOWN_LIMIT のデフォルト値 50*1000ms) です。

3.4 RYZ014A Cellular FIT モジュールの排他制御処理

RYZ014A Cellular FIT モジュールで、排他制御が必要な処理を表 3.2 に示します。

表 3.2 排他制御処理が必要な Cellular FIT モジュールの処理

処理内容	排他制御開始位置	排他制御終了位置
AT コマンド発行	AT コマンド実行用の内部関数実行前	AT コマンド実行用の内部関数実行完了後
RTS 端子制御	RTS 端子制御開始前	RTS 端子制御終了後
ソケットデータ受信処理	受信処理関数実行前	受信処理関数実行完了後
API 関数実行	API 開始直後	API 終了直前

3.4.1 AT コマンド実行処理の排他制御

RYZ014A Cellular モジュールは AT コマンドを逐次処理するため、AT コマンド実行中に他の AT コマンドが実行されないように排他制御を行う必要があります。

Cellular FIT モジュールでは、AT コマンド実行前 (atc_***() 関数実行時) にセマフォの取得処理、および AT コマンド発行完了後にセマフォの開放処理が必要です。以下の処理で、AT コマンド実行処理の排他制御に使用するセマフォの取得および解放を実行できます。

- セマフォ取得：cellular_take_semaphore(p_ctrl->at_semaphore)
- セマフォ解放：cellular_give_semaphore(p_ctrl->at_semaphore)

3.4.2 RTS 端子制御処理の排他制御

RYZ014A Cellular モジュールは、モジュールを PSM 動作モードに遷移させるために RTS 端子を制御する必要があります。RTS 端子制御中に他の API により RTS 端子制御が実行されないように排他制御が必要になります。

Cellular FIT モジュールでは、RTS 端子制御前にセマフォの取得処理、および RTS 端子制御完了後にセマフォの開放処理が必要です。以下の処理で、RTS 制御処理の排他制御に使用するセマフォの取得および解放を実行できます。

- セマフォ取得：cellular_take_semaphore(p_ctrl->ring_ctrl.rts_semaphore)
- セマフォ解放：cellular_give_semaphore(p_ctrl->ring_ctrl.rts_semaphore)

3.4.3 データ受信処理の排他制御

Cellular FIT モジュールでは、ソケットからのデータ受信処理開始前にセマフォの取得処理、およびソケットデータ受信処理完了後にセマフォの開放処理が必要です。以下の処理で、ソケット毎にセマフォ取得および解放処理を実行できます。

- セマフォ取得：cellular_take_semaphore
  (p_ctrl->p_socket_ctrl[ソケット番号 – CELLULAR_START_SOCKET_NUMBER].rx_semaphore)
- セマフォ解放：cellular_give_semaphore
  (p_ctrl->p_socket_ctrl[ソケット番号 – CELLULAR_START_SOCKET_NUMBER].rx_semaphore)

※ CELLULAR_START_SOCKET_NUMBER は、Cellular モジュールのソケット開始番号です。使用する Cellular モジュールの仕様に合わせて、適切な値を設定してください。

3.4.4 API 関数実行時の排他制御 (スレッドセーフ機能)

Cellular FIT モジュールでは、API 関数毎にスレッドセーフ機能を所持しています。新規に API を追加する場合、以下の処理を実行することで API にスレッドセーフ機能を付加することができます。

本スレッドセーフは、Cellular FIT モジュールの API 実行中に R_CELLULAR_Close() が実行された場合に、R_CELLULAR_Close() により Cellular FIT モジュールが使用しているリソースが解放され動作不具合が発生する問題を回避することができます。

① クリティカルセクションを設定：preemption = cellular_interrupt_disable()
② 他の API 実行状況の確認処理を追加：

```
if (0 != (p_ctrl->running_api_count % 2))
{
   ret = CELLULAR_ERR_OTHER_API_RUNNING;
}
else
{
   p_ctrl->running_api_count += 1 or 2;    // 1 = 他の API 関数との同時動作が不可の場合
                                           // 2 = 他の API 関数との同時動作が可能な場合
}
```

③ クリティカルセクション無効化：cellular_interrupt_enable(preemption)
④ p_ctrl->running_api_count – 1 or 2;　　// ②で加算した値を減算 (②加算を実行した場合のみ)

3.5 コードサイズ

RYZ014A Cellular FIT モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを表 3.3 に示します。

表 3.3 に示す値は下記条件で確認しています。

FIT モジュールリビジョン: r_cellular rev1.11

コンパイラバージョン: Renesas Electronics C/C++ Compiler for RX Family V3.04.00
(統合開発環境のデフォルト設定に“-lang = c99“オプションを追加)

コンフィグレーションオプション: デフォルト設定

表 3.3 コードサイズ

ROM、RAM およびスタックのコードサイズ			
デバイス	**分類**	**使用メモリ**	**備考**
RX65N RX72N	ROM	約 36k バイト	-
	RAM	約 600 バイト	-
	最大使用スタックサイズ	約 700 バイト	-

4. AT コマンドの新規追加方法について

4.1 AT コマンド新規追加時の作業ステップ

AT コマンドを新規追加する全体の作業ステップを、図 6 に示します。

追加する AT コマンドの種類 (レスポンスの違い) により、実施する作業が異なります。

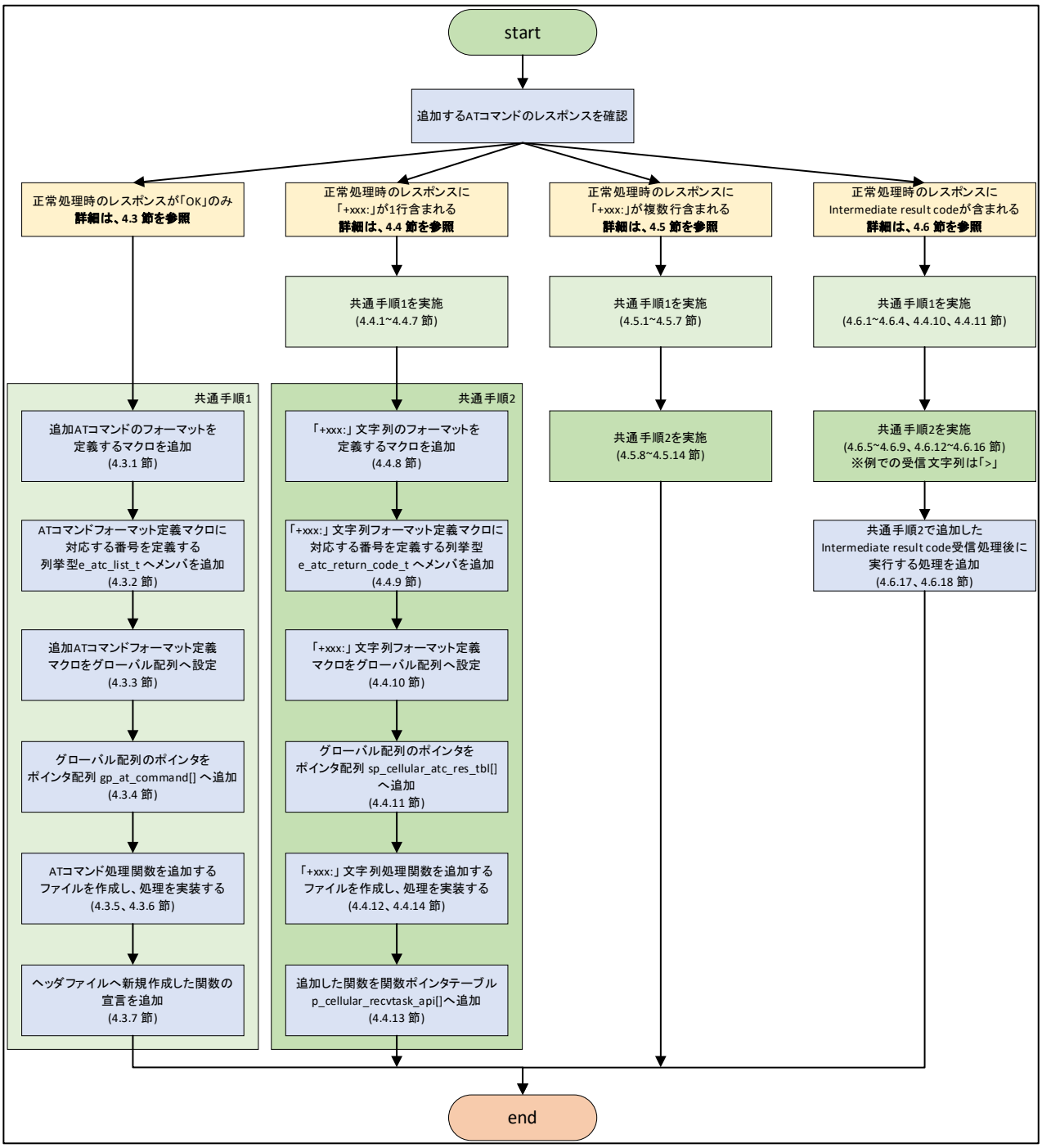

図 6　AT コマンドの新規追加フロー図

4.2　修正が必要なファイル

RYZ014A Cellular FIT モジュールのフォルダ構成を、図 7 に示します。

ユーザが AT コマンドを新規追加する場合は、RYZ014A Cellular FIT モジュールの仕様 (詳細は 3.2 節を参照) により、図 7 に示すファイルの修正が必要となります。

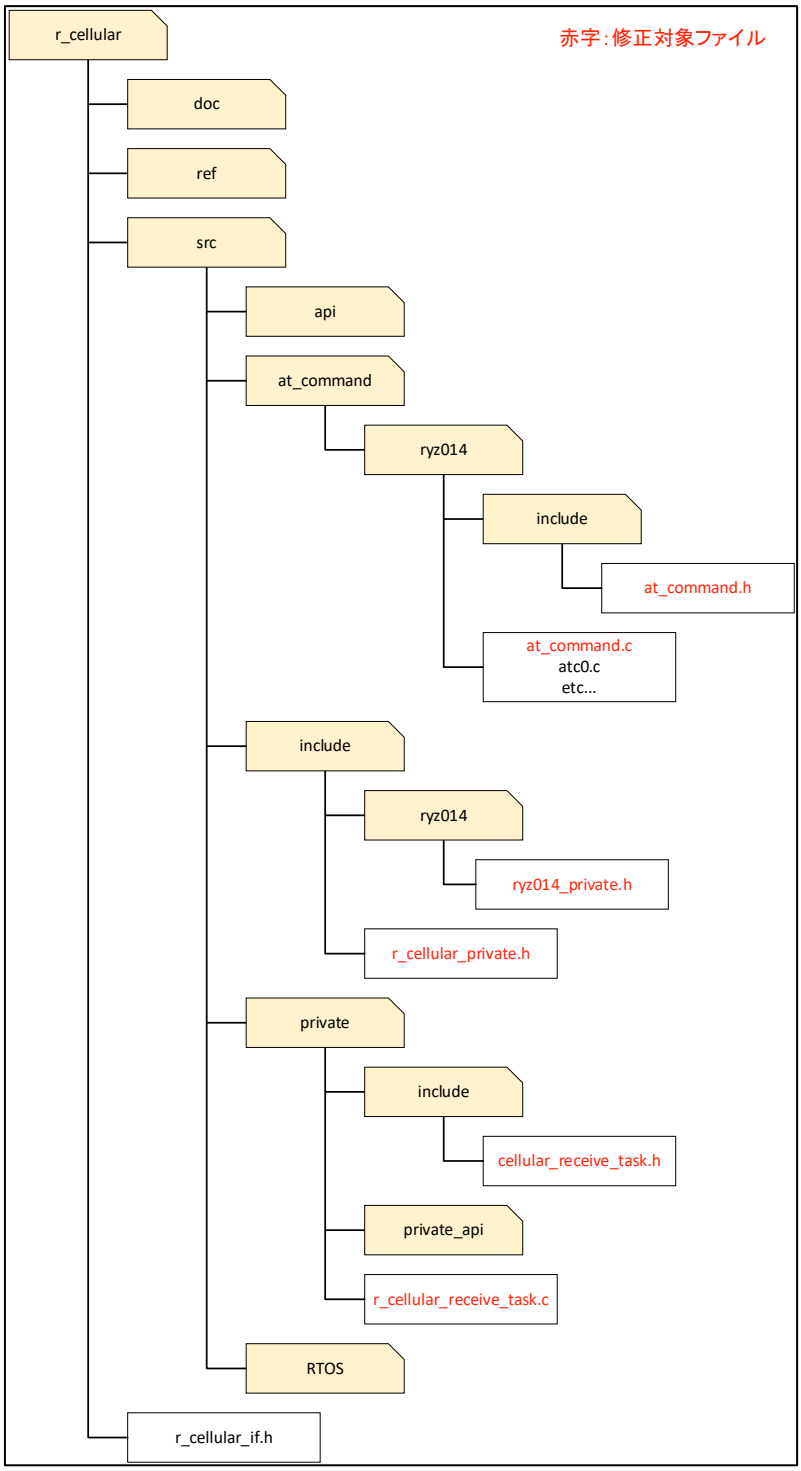

図 7　修正対象ファイル一覧

4.2.1 AT コマンドフォーマットを定義するファイル

Cellular FIT モジュールは、サポートする AT コマンドのフォーマットを ryz014_private.h で定義しています。AT コマンドを新規追加する場合は、ryz014_private.h を編集してください。

AT コマンドは、マクロ定義された AT コマンドのひな型 (書式) を設定して sprintf 関数を実行することで生成します。AT コマンド書式は、終端文字 (”¥r”) まで含めて定義してください。

ryz014_private.h は、AT コマンドのフォーマットおよびそれらに対応する番号の列挙型 e_atc_list_t を定義します。AT コマンドのフォーマット定義を図 8 に、列挙型 e_atc_list_t の各要素の対応関係を図 9 に示します。

AT コマンドを新規追加する場合は、追加する AT コマンドのフォーマット定義を ryz014_private.h へ追加してください。また、追加する AT コマンドに対応する番号の定義を列挙型 e_atc_list_t へ追加してください。追加する定義は、ATC_LIST_MAX より上 (先頭側) へ挿入してください。

```
#define RYZ014_ATC_ECHO_OFF                 "ATE0\r"
#define RYZ014_ATC_FUNCTION_LEVEL_CHECK     "AT+CFUN?\r"
#define RYZ014_ATC_FUNCTION_LEVEL           "AT+CFUN=%s\r"
#define RYZ014_ATC_PIN_LOCK_CHECK           "AT+CPIN?\r"
#define RYZ014_ATC_PIN_LOCK_RELEASE         "AT+CPIN=\"%s\"\r"
                          |
                        中略
                          |
#define RYZ014_ATC_WRITE_CERTIFICATE        "AT+SQNSNVW=\"%s\",%s,%s\r"
#define RYZ014_ATC_ERASE_CERTIFICATE        "AT+SQNSNVW=\"%s\",%s,0\r"
#define RYZ014_ATC_GET_CERTIFICATE          "AT+SQNSNVR=\"%s\",%s\r"
#define RYZ014_ATC_CONFIG_SSL_PROFILE       "AT+SQNSPCFG=%s,2,,%s,%s,%s,%s,\"\"\r"
#if (CELLULAR_IMPLEMENT_TYPE == 'B')
#define RYZ014_ATC_CONFIG_SSL_SOCKET        "AT+SQNSSCFG=%s,%s,%s\r"
#endif
#define RYZ014_NO_COMMAND                   "\r"


typedef enum
{
    ATC_ECHO_OFF         = 0,
    ATC_FUNCTION_LEVEL_CHECK,
    ATC_FUNCTION_LEVEL,
    ATC_PIN_LOCK_CHECK,
    ATC_PIN_LOCK_RELEASE,
             |
           中略
             |
    ATC_ERASE_CERTIFICATE,
    ATC_GET_CERTIFICATE,
    ATC_CONFIG_SSL_PROFILE,
#if (CELLULAR_IMPLEMENT_TYPE == 'B')
    ATC_CONFIG_SSL_SOCKET,
#endif
    ATC_SQNSSENDEXT_END,
    ATC_LIST_MAX
} e_atc_list_t;
```

図 8　ryz014_private.h 定義

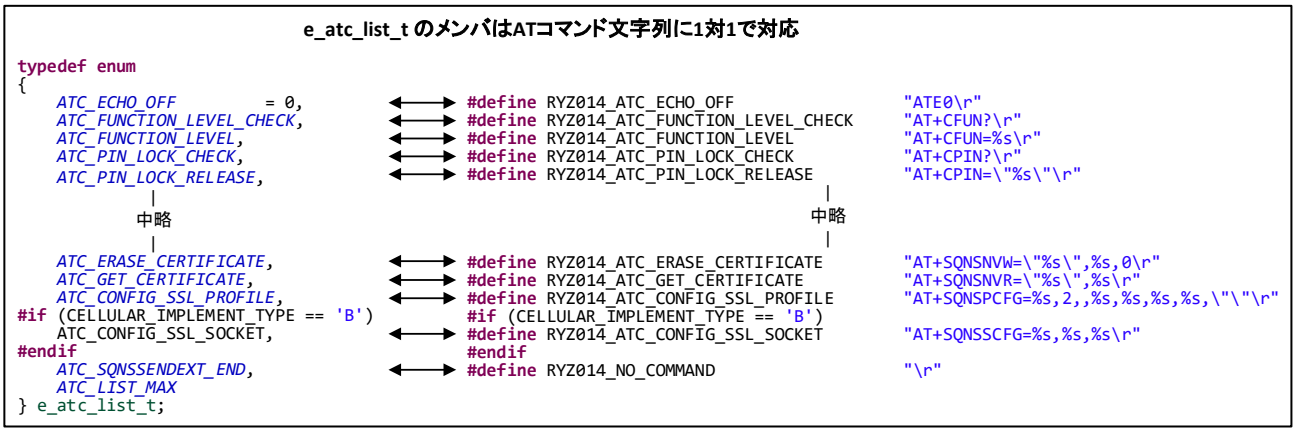

図 9　列挙型 e_atc_list_t と AT コマンド定義の対応

4.2.2 AT コマンドフォーマットのテーブルを定義するファイル

Cellular FIT モジュールは、AT コマンドフォーマットをテーブルデータとして管理します。AT コマンドフォーマットのテーブルは、at_command.c で定義しています。AT コマンドを新規追加する場合は、at_command.c を編集してください。

at_command.c は、ryz014_private.h で定義される AT コマンドフォーマットの文字列定数、およびそれらを参照するポインタテーブル gp_at_command[] を定義します。AT コマンドフォーマット定義、AT コマンドフォーマット文字列定数および gp_at_command[] の各要素の対応関係を、図 10 に示します。

AT コマンドを新規追加する場合は、ryz014_private.h へ追加した AT コマンドフォーマットの文字列定数の定義を at_command.c へ追加してください。また、追加した文字列定数へのポインタを gp_at_command[] へ追加してください。

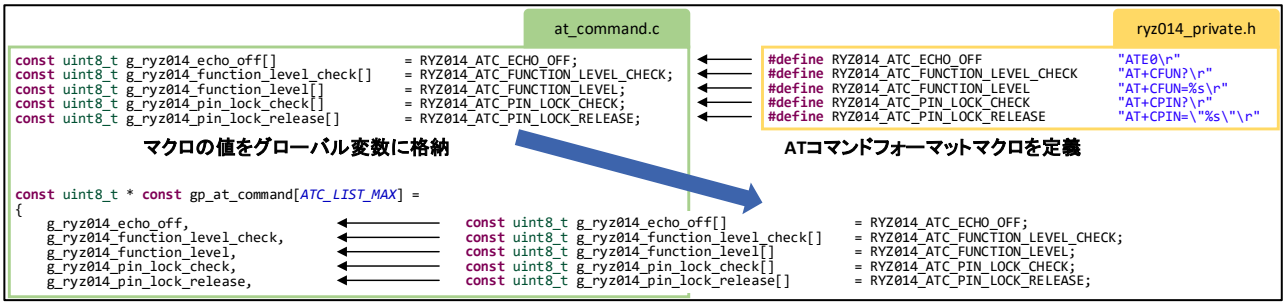

図 10　AT コマンドフォーマット定義、AT コマンドフォーマット文字列定数および gp_at_command[] の各要素の対応

4.2.3 レスポンス文字列を定義するファイル

Cellular FIT モジュールは、Cellular モジュールに対して AT コマンドを実行後に返されるレスポンス文字列を cellular_receive_task.h で定義しています。レスポンス文字列を新規追加する場合は、cellular_receive_task.h を編集してください。

cellular_receive_task.h は、レスポンス文字列およびそれらに対応する番号を列挙型 e_atc_return_code_t で定義します。また、cellular_receive_task.c では e_atc_return_code_t の各メンバに対応する関数ポインタテーブル p_cellular_recvtask_api[]を定義します。レスポンス文字列定義と e_atc_return_code_t の各メンバの対応関係、および e_atc_return_code_t と p_cellular_recvtask_api[]の各メンバの対応関係を、図 11 に示します。

レスポンス文字列を新規追加する場合は、レスポンス文字列の定義を cellular_receive_task.h へ追加、および追加したレスポンスに対応する番号の定義を e_atc_return_code_t へ追加してください。追加する定義は、CELLULAR_RES_MAX より上 (先頭側) に挿入してください。

e_atc_return_code_t の各メンバは、関数ポインタテーブル p_cellular_recvtask_api[] の各メンバと 1 対 1 で対応しています。e_atc_retur_code_t へ番号を追加した場合、それに対応する関数のポインタを cellular_receive_task.c で定義される p_cellular_recvtask_api[] へ追加してください。追加する関数は、ユーザが実装する必要があります。実装すべき処理がない場合は、cellular_memclear() の関数ポインタを p_cellular_recvtask_api[] へ追加してください。

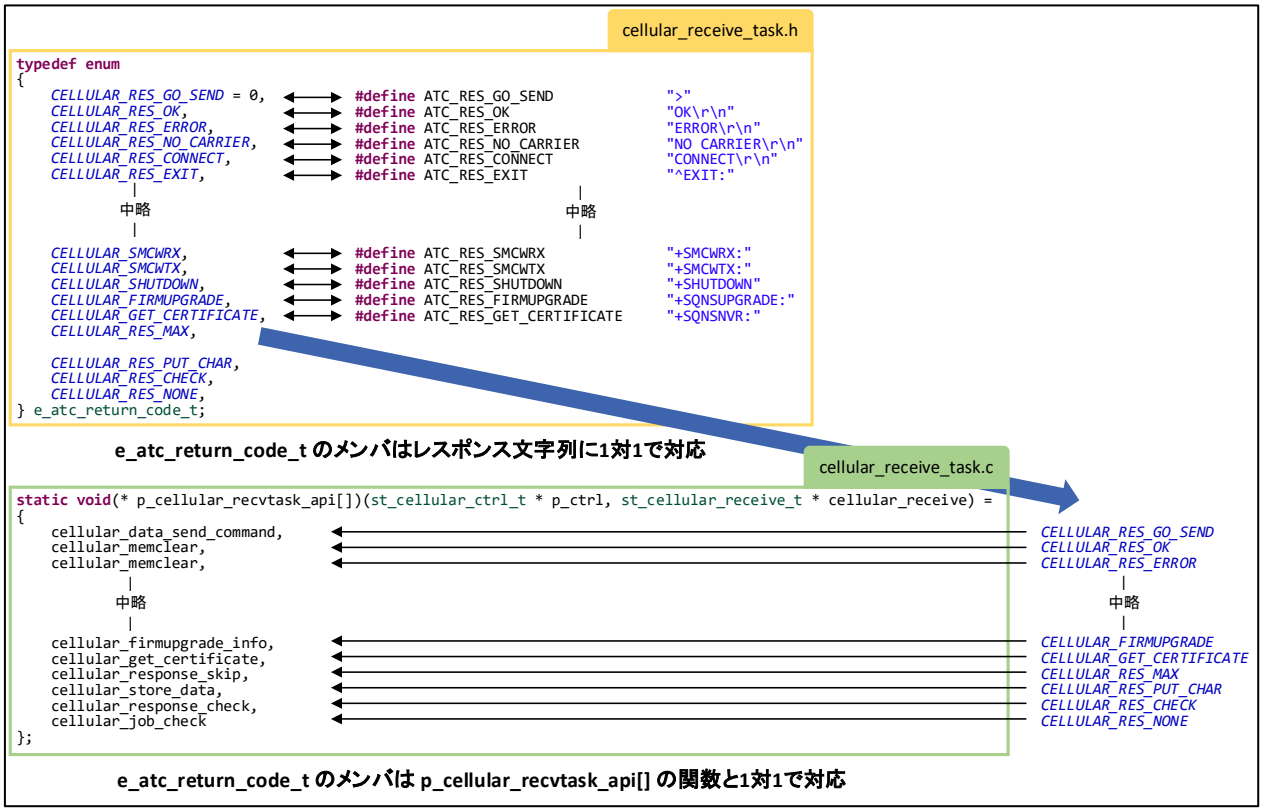

図 11　レスポンス文字列定義と e_atc_return_code_t の各メンバの関係

4.2.4 レスポンス文字列テーブルを定義するファイル

Cellular FIT モジュールは、レスポンス文字列テーブルを r_cellular_receive_task.c で定義します。レスポンス文字列を新規追加する場合は、cellular_receive_task.c を編集してください。

cellular_receive_task.c では、レスポンス文字列定数およびそのポインタのテーブル sp_cellular_atc_res_tbl[] を定義します。レスポンス文字列定義、レスポンス文字列定数および sp_cellular_atc_res_tbl[] の対応関係を、図 12 に示します。

レスポンス文字列を新規追加する場合は、追加したレスポンス文字列定数のポインタを sp_cellular_atc_res_tbl[] へ追加してください。レスポンス文字列定数の追加については、4.2.3 節を参照してください。

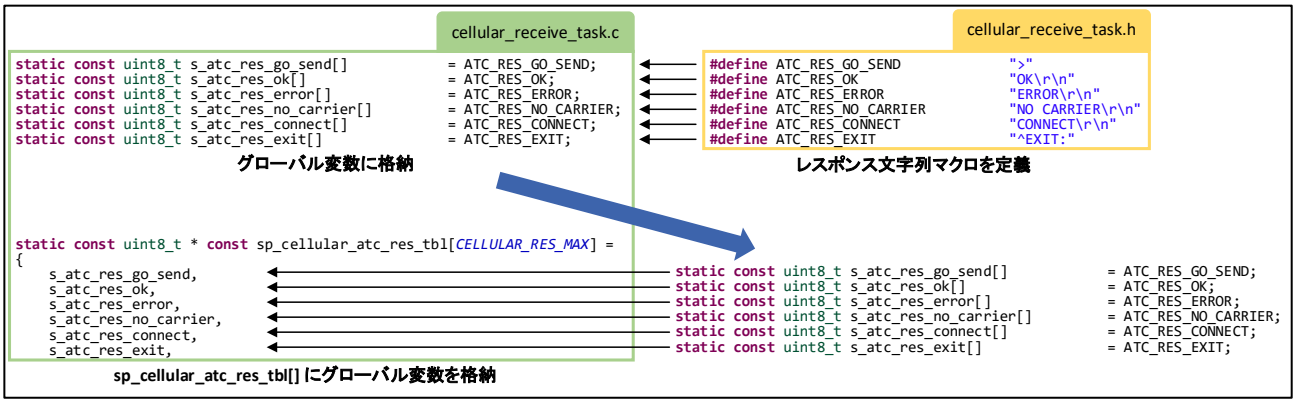

図 12　レスポンス文字列マクロ格納図

4.3　正常処理時のレスポンスが「OK」のみのコマンド

正常処理時のレスポンスが「OK」のみ返される AT コマンドを追加する場合のソースコード修正手順を説明します。

ここでは、AT+CTZU コマンドを新規追加する場合を例として説明します。AT コマンドの仕様は、関連ドキュメント[10]を参照してください。

```
AT+CTZU=1

OK
```

7.17 Automatic Time Zone Update: AT+CTZU

Note: This command is described in 3GPP TS 27.007. See Section References.

7.17.1 Syntax

Command	Possible Response(s)
AT+CTZU=<onoff>	+CME ERROR: <err>

図 13　新規追加する AT コマンド「AT+CTZU」

onoff

Integer. 0 or 1. Boolean switch.

Table 146.　onoff

Value	Description
0	Disable automatic time zone update via NITZ.
1	Enable automatic time zone update via NITZ.

図 14　「AT+CTZU」コマンドの<onoff>の仕様

4.3.1 マクロを追加

ryz014a_private.h へ、追加する AT コマンドのフォーマット (書式) を定義するマクロを追加します。

図 14 の AT コマンド仕様を参照すると、<onoff>には 0 or 1 を指定可能であるため、AT コマンド文字列の<onoff>に対応する部分は「%s」とします。

```
158 #define RYZ014_ATC_CONFIG_SSL_PROFILE        "AT+SQNSPCFG=%s,2,,%s,%s,%s,%s,\"\"\r"
159 #define RYZ014_ATC_TIMEZONE_UPDATE           "AT+CTZU=%s\r"
160 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 15　新規追加コマンド「AT+CTZU」を追加した状態

4.3.2 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_atc_list_t へ、新規メンバを追加します。

図 16 に示す位置に追加した場合、ATC_CONFIG_TIMEZONE_UPDATE の値は 70 となります。

```
239     ATC_CONFIG_SSL_PROFILE,
240     ATC_CONFIG_TIMEZONE_UPDATE,
241 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 16　列挙型 e_atc_list_t に新規メンバを追加した状態

4.3.3 定数の追加および文字列の格納

at_command.c へ文字列定数の定義を追加し、4.3.1 節で追加した文字列を格納します。

```
110 const uint8_t g_ryz014_config_ssl_profile[]      = RYZ014_ATC_CONFIG_SSL_PROFILE;
111 const uint8_t g_ryz014_config_timezone_update[]  = RYZ014_ATC_TIMEZONE_UPDATE;
112 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 17　文字列定数の定義を追加した状態

4.3.4 文字列のアドレスを追加

at_command.c で定義されているポインタ配列 gp_at_command[] へ、4.3.3 節で追加した文字列のアドレスを追加します。

必ず、図 16 で追加したメンバに合わせた位置に追加してください。ATC_CONFIG_TIMEZONE_UPDATE の値が 70 のため、図 18 では配列 gp_at_command[] の要素番号 70 の位置、すなわち 71 番目にメンバを追加しています。

```
189     g_ryz014_config_ssl_profile,
190     g_ryz014_config_timezone_update,
191 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 18　配列 gp_at_command[] に新規追加した状態

4.3.5 新規ファイルの作成

「r_cellular/src/at_command/ryz014」フォルダ内に、新規ファイルを作成します。

既存のファイル (cfun.c 等の、引数が 1 つのタイプ) をコピー＆リネームすることを推奨します。

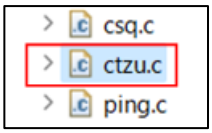

図 19　新規ファイル「ctzu.c」を追加した状態

4.3.6 AT コマンド実行関数を作成

4.3.5 節で作成した.c ファイルに、「AT+CTZU」コマンドを実行する関数を作成します。

Examples

```
#include "at_command.h"
#include "cellular_private_api.h"

e_cellular_err_t atc_ctzu(st_cellular_ctrl_t * const p_ctrl,
                          const uint8_t              onoff) ①
{
   uint8_t           str[2]                                  = {0}; ②
   const uint8_t *   p_command_arg[CELLULAR_MAX_ARG_COUNT]   = {0}; ③
   e_cellular_err_t ret                                      = CELLULAR_SUCCESS;

   sprintf((char *)str, "%d", onoff); // (uint8_t *)->(char *)  ④

   p_command_arg[0] = str; ⑤

   atc_generate(p_ctrl->sci_ctrl.atc_buff,
                gp_at_command[ATC_CONFIG_TIMEZONE_UPDATE],      ⑥
                p_command_arg);

   ret = cellular_execute_at_command(p_ctrl,
                                     p_ctrl->sci_ctrl.atc_timeout,  ⑦
                                     ATC_RETURN_OK,
                                     ATC_CONFIG_TIMEZONE_UPDATE);

   return ret;
}
```

① 図 14 よりパラメータの値は 0 or 1 であるため、引数は uint8_t 型とします。
第 1 引数は、管理構造体のポインタ「st_cellular_ctrl_t * const p_ctrl」とします。

② AT コマンドのパラメータを格納するための配列を宣言します。

③ atc_generate() 関数の第 2 引数へ設定するポインタ配列を宣言します。

④ sprintf() 関数を使用し、②で宣言した配列に本関数の第 2 引数の値を文字列へ変換して格納します。

⑤ ポインタ配列に、④で変換した文字列の先頭アドレスを格納します。

⑥ atc_generate() 関数を使用し、Cellular モジュールへ送信する AT コマンド文字列を生成します。

- 第 1 引数 … 「p_ctrl->sci_ctrl.atc_buff」を設定
- 第 2 引数 … 「gp_at_command [図 16 で追加したメンバを指定]」を設定
- 第 3 引数 … ③で宣言したポインタ配列を設定

⑦ cellular_execute_at_command() 関数を使用し、Cellular モジュールへ AT コマンドを送信します。

- 第 1 引数 … 「p_ctrl」を設定
- 第 2 引数 … Cellular モジュールからレスポンスが返るまでのタイムアウト時間を設定
- 第 3 引数 … Cellular モジュールから送られてくるレスポンスの期待値を設定
  一部のコマンド以外は、「ATC_RETURN_OK」を設定
- 第 4 引数 … 送信する AT コマンド番号 (図 16 で追加したメンバ) を設定

4.3.7 新規作成した関数の宣言を追加

at_command.h へ、新規作成した関数「atc_ctzu()」の宣言を追加します。

```
/***********************************************************************************************
 * Function Name  @fn            atc_ctzu
 * Description    @details       Execute the AT command (AT+CTZU).
 * Arguments      @param[in/out] p_ctrl -
 *                                  Pointer to managed structure.
 *                @param[in]     onoff -
 *                                  Enable(1)/Disable(0) automatic time zone update via NITZ.
 * Return Value   @retval        CELLULAR_SUCCESS -
 *                                  Successfully executed AT command.
 *                @retval        CELLULAR_ERR_MODULE_COM -
 *                                  Communication with module failed.
 ***********************************************************************************************/
e_cellular_err_t atc_ctzu (st_cellular_ctrl_t * const p_ctrl, const uint8_t onoff);
```

図 20　「atc_ctzu()」の宣言を追加した状態

4.4　正常処理時のレスポンスに「+xxx:」の文字列が 1 行含まれるコマンド

正常処理時のレスポンスに「+xxx:」の文字列が 1 行含まれる AT コマンドを追加する場合のソースコード修正手順を説明します。

4.3 節に引き続き、ここでは、AT+CPAS コマンドを新規追加する場合を例として説明します。AT コマンドの仕様は、関連ドキュメント[10]を参照してください。

7.11 Phone Activity Status: AT+CPAS

Note: This command is described in 3GPP TS 27.007. See Section References.

7.11.1 Syntax

Command	Possible Response(s)
AT+CPAS	+CPAS: <pas> +CME ERROR: <err>

図 21　新規追加する AT コマンド「AT+CPAS」

7.11.3 Defined Values

pas

Integer.

Caution: Only 0, 4 and 5 values are currently implemented. All other values are reserved.

Table 136.　pas

Value	Description
0	Ready (MT allows commands from TA/TE)
1	Unavailable (MT does not allow commands from TA/TE)
2	Unknown (MT is not guaranteed to respond to instructions)
3	Ringing (MT is ready for commands from TA/TE, but the ringer is active)
4	Call in progress (MT is ready for commands from TA/TE, but a call is in progress)
5	Asleep (MT is unable to process commands from TA/TE because it is in a low functionality state)
6..128	Reserved

図 22　+CPAS:に付随する＜pas＞の仕様

4.4.1 マクロを追加

ryz014a_private.h へ、追加する AT コマンドのフォーマットを定義するマクロを追加します。

```
159 #define RYZ014_ATC_TIMEZONE_UPDATE          "AT+CTZU=%s\r"
160 #define RYZ014_ATC_GET_MT_STATE             "AT+CPAS\r"
161 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 23　新規追加コマンド「AT+CPAS」を追加した状態

4.4.2 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_atc_list_t へ、新規メンバを追加します。

図 24 に示す位置に追加した場合、ATC_GET_MT_STATE の値は 71 となります。

```
240     ATC_CONFIG_TIMEZONE_UPDATE,
241     ATC_GET_MT_STATE,
242 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 24　列挙型 e_atc_list_t に新規メンバを追加した状態

4.4.3 定数の追加および文字列の格納

at_command.c へ文字列定数を追加し、4.4.1 節で追加した文字列を格納します。

```
111 const uint8_t g_ryz014_config_timezone_update[]  = RYZ014_ATC_TIMEZONE_UPDATE;
112 const uint8_t g_ryz014_get_mt_state[]            = RYZ014_ATC_GET_MT_STATE;
113 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 25　文字列定数を追加した状態

4.4.4 文字列のアドレスを追加

at_command.c で定義されているポインタ配列 gp_at_command[] に、4.4.3 節で追加した文字列のアドレスを追加します。

必ず、図 24 で追加したメンバに合わせた位置に追加してください。ATC_GET_MT_STATE の値が 71 のため、図 26 では配列 gp_at_command[] の要素番号 71 の位置、すなわち 72 番目にメンバを追加しています。

```
190     g_ryz014_config_timezone_update,
191     g_ryz014_get_mt_state,
192 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 26　配列 gp_at_command[] に新規追加した状態

4.4.5 新規ファイルの作成

「r_cellular/src/at_command/ryz014」フォルダ内に、新規ファイルを作成します。

既存のファイル（ceer.c 等の、引数のないタイプ）をコピー＆リネームすることを推奨します。

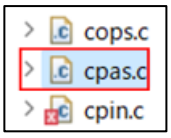

図 27　新規ファイル「cpas.c」を追加した状態

4.4.6 AT コマンド実行関数を作成

4.4.5 節で作成した.c ファイルに、「AT+CPAS」コマンドを実行する関数を作成します。

Examples

```
#include "at_command.h"
#include "cellular_private_api.h"

e_cellular_err_t atc_cpas(st_cellular_ctrl_t * const p_ctrl)
{
    e_cellular_err_t ret = CELLULAR_SUCCESS;

    atc_generate(p_ctrl->sci_ctrl.atc_buff,
                 gp_at_command[ATC_GET_MT_STATE], ①
                 NULL);

    ret = cellular_execute_at_command(p_ctrl,
                                      p_ctrl->sci_ctrl.atc_timeout,
                                      ATC_RETURN_OK,
                                      ATC_GET_MT_STATE);  ②

    return ret;
}
```

① atc_generate() 関数を使用し、Cellular モジュールへ送信する AT コマンド文字列を生成します。

- 第 1 引数 … 「p_ctrl->sci_ctrl.atc_buff」を設定
- 第 2 引数 … 「gp_at_command [図 24 で追加したメンバを指定]」を設定
- 第 3 引数 … AT コマンドに引数が存在しないため、NULL を設定

② cellular_execute_at_command() 関数を使用し、Cellular モジュールへ AT コマンドを送信します。

- 第 1 引数 … 「p_ctrl」を設定
- 第 2 引数 … Cellular モジュールからレスポンスが返るまでのタイムアウト時間を設定
- 第 3 引数 … Cellular モジュールから送られてくるレスポンスの期待値を設定
  一部のコマンド以外は、「ATC_RETURN_OK」を設定
- 第 4 引数 … 送信する AT コマンド番号（図 24 で追加したメンバ）を設定

4.4.7 新規作成した関数の宣言を追加

at_command.h へ、新規作成した関数「atc_cpas()」の宣言を追加します。

```
/**********************************************************************
 * Function Name  @fn             atc_cpas
 * Description    @details        Execute the AT command (AT+CPAS).
 * Arguments      @param[in/out] p_ctrl -
 *                                  Pointer to managed structure.
 * Return Value   @retval        CELLULAR_SUCCESS -
 *                                  Successfully executed AT command.
 *                @retval        CELLULAR_ERR_MODULE_COM -
 *                                  Communication with module failed.
 **********************************************************************/
e_cellular_err_t atc_cpas (st_cellular_ctrl_t * const p_ctrl);
```

図 28　「atc_cpas()」の宣言を追加した状態

4.4.8 マクロを追加

cellular_receive_task.h へ、レスポンス文字列「+CPAS:」のマクロ定義を追加します。

```
#define ATC_RES_GET_CERTIFICATE     "+SQNSNVR:"
#define ATC_RES_GET_MT_STATE        "+CPAS:"
```

図 29　「+CPAS:」のマクロ定義を追加した状態

4.4.9 列挙型へ新規メンバを追加

cellular_receive_task.h で定義されている列挙型 e_atc_return_code_t へ、新規メンバを追加します。

図 30 に示す位置に追加した場合、CELLULAR_GET_MT_STATE の値は 41 となります。

```
    CELLULAR_GET_CERTIFICATE,           // Get Certificate
    CELLULAR_GET_MT_STATE,              // Get Phone Activity Status
    CELLULAR_RES_MAX,                   // End of analysis result processing
```

図 30　列挙型 e_atc_return_code_t に新規メンバを追加した状態

4.4.10 定数の追加および文字列の格納

r_cellular_receive_task.c へ文字列定数を追加し、4.4.8 節で追加した文字列を格納します。

```
static const uint8_t s_atc_res_get_certificate[]    = ATC_RES_GET_CERTIFICATE;
static const uint8_t s_atc_res_get_mt_state[]       = ATC_RES_GET_MT_STATE;
```

図 31　文字列定数を追加した状態

4.4.11 文字列のアドレスを追加

r_cellular_receive_task.c で定義されているポインタ配列 sp_cellular_atc_res_tbl[] へ、4.4.10 節で追加した文字列のアドレスを追加します。

必ず、図 30 で追加したメンバに合わせた位置に追加してください。CELLULAR_GET_MT_STATE の値が 41 のため、図 32 では配列 sp_cellular_atc_res_tbl[] の要素番号 41、すなわち 42 番目にメンバを追加しています。

```
141     s_atc_res_get_certificate,
142     s_atc_res_get_mt_state,
143 };
```

図 32　配列 sp_cellular_atc_res_tbl[] に新規追加した状態

4.4.12 関数の宣言を追加

r_cellular_receive_task.c へ、レスポンス「+CPAS:」受信時の処理を実装する関数の宣言を追加します。

引数は、他の関数と同じにしてください。

```
186 static void cellular_get_certificate (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
187 static void cellular_get_mt_state (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
188 static void cellular_get_revision (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
```

図 33　「+CPAS:」受信時に処理を行う関数を宣言した状態

4.4.13 関数ポインタテーブルへ関数を登録

r_cellular_receive_task.c で定義されている関数ポインタテーブル p_cellular_recvtask_api[] へ、4.4.12 節で宣言した関数を追加します。

必ず、図 30 で追加したメンバに合わせた位置に追加してください。CELLULAR_GET_MT_STATE の値が 41 のため、図 34 では配列 p_cellular_recvtask_api[] の要素番号 41、すなわち 42 番目にメンバを追加しています。

```
243     cellular_get_certificate,
244     cellular_get_mt_state,
245     cellular_response_skip,
```

図 34 新規に宣言した関数を追加した状態

4.4.14 レスポンス処理関数を追加

r_cellular_receive_task.c へ、4.4.12 節で宣言した関数を追加します。

Examples

```
static void cellular_get_mt_state(st_cellular_ctrl_t     * p_ctrl,
                                  st_cellular_receive_t * cellular_receive)
{
    st_cellular_receive_t * p_cellular_receive = cellular_receive;
    uint8_t               * p_state            = p_ctrl->recv_data;  ①

    if (CHAR_CHECK_4 == p_cellular_receive->data)  ②
    {
        if (NULL != p_state)
        {
            sscanf((char *)&p_ctrl->sci_ctrl.receive_buff[p_cellular_receive-
>tmp_recvcnt],
                   " %hhd", (char *)p_state);  ③
        }
        cellular_cleardata(p_ctrl, p_cellular_receive);
    }

    return;
}
```

① 値受け渡し用のバッファアドレスをローカル変数へコピーします。

② 終端文字(¥n) を確認し、処理を開始します。

③ ①で p_ctrl->recv_data にアドレスが格納されていた場合のみ処理を実行します。
受信文字列「+CPAS:」に続く文字を、p_state へ格納します。
ここでは sscanf() 関数の第 1 引数に「¥r¥n+CPAS: <pas>¥r¥n」が格納されているため、半角スペース以降の文字列が p_state へ格納されます。

4.4.15 補足事項

4.4.14 節の関数のように p_ctrl->recv_data を通じてデータを受け取る場合、関数（この例では atc_cpas() 関数）の実行前に p_ctrl->recv_data へデータ書き込み先のアドレスを格納する必要があります。

データ書き込み完了後、必ず p_ctrl->recv_data へ NULL を設定してください。

Examples

```
/* 関数内で閉じて処理する場合 */
void Examples_API1(st_cellular_ctrl_t * const p_ctrl)
{
    e_cellular_err_t           ret;
    e_cellular_err_semaphore_t semahore_ret;
    uint8_t                    state = 0;

    semahore_ret = cellular_take_semaphore(p_ctrl->at_semaphore);
    if (CELLULAR_SEMAPHORE_SUCCESS == semahore_ret)
    {
        p_ctrl->recv_data = (void *) &state;
        ret = atc_cpas(p_ctrl);
        p_ctrl->recv_data = NULL;
        cellular_give_semaphore(p_ctrl->at_semaphore);
    }

    return;
}

/* ユーザへデータを返す場合 */
void Examples_API2(st_cellular_ctrl_t * const p_ctrl,
                   uint8_t             * p_state)
{
    e_cellular_err_t           ret;
    e_cellular_err_semaphore_t semahore_ret;

    semahore_ret = cellular_take_semaphore(p_ctrl->at_semaphore);
    if (CELLULAR_SEMAPHORE_SUCCESS == semahore_ret)
    {
        p_ctrl->recv_data = (void *) p_state;
        ret = atc_cpas(p_ctrl);
        p_ctrl->recv_data = NULL;
        cellular_give_semaphore(p_ctrl->at_semaphore);
    }

    return;
}
```

4.5　正常処理時のレスポンスに「+xxx:」の文字列が複数行含まれるコマンド

正常処理時のレスポンスに「+xxx:」の文字列が複数行含まれる AT コマンドを追加する場合のソースコード修正手順を説明します。

4.4 節に引き続き、ここでは、AT+COPN コマンドを新規追加する場合を例として説明します。AT コマンドの仕様は、関連ドキュメント[10]を参照してください。

```
AT+COPN

+COPN: "20205","vodafone GR"
+COPN: "20404","vodafone NL"
+COPN: "20408","NL KPN"
```

～中略～

```
+COPN: "74801","Antel"
+COPN: "90112","Telenor Maritime"
+COPN: "90171","Tampnet"

OK
```

7.9 Read Operator Names: AT+COPN

Note: This command is described in 3GPP TS 27.007. See Section References.

7.9.1 Syntax

Command	Possible Response(s)
AT+COPN	+COPN:<numeric1>,<alpha1>[<S3><S4>+COPN:<numeric2>,<alpha2>[...]] +CME ERROR: <err>

図 35　新規追加する AT コマンド「AT+COPN」

4.5.1 マクロを追加

ryz014a_private.h へ、追加する AT コマンドのフォーマットを定義するマクロを追加します。

```
160 #define RYZ014_ATC_GET_MT_STATE                 "AT+CPAS\r"
161 #define RYZ014_ATC_GET_OPERATOR_LIST            "AT+COPN\r"
162 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 36　新規追加コマンド「AT+COPN」を追加した状態

4.5.2 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_atc_list_t へ、新規メンバを追加します。

図 37 に示す位置に追加した場合、ATC_GET_OPERATOR_LIST の値は 72 となります。

```
244     ATC_GET_MT_STATE,
245     ATC_GET_OPERATOR_LIST,
246 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 37　列挙型 e_atc_list_t に新規メンバを追加した状態

4.5.3 定数の追加および文字列の格納

at_command.c へ文字列定数を追加し、4.5.1 節で追加した文字列を格納します。

```
112 const uint8_t g_ryz014_get_mt_state[]            = RYZ014_ATC_GET_MT_STATE;
113 const uint8_t g_ryz014_get_operator_list[]       = RYZ014_ATC_GET_OPERATOR_LIST;
114 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 38　文字列定数を追加した状態

4.5.4 文字列のアドレスを追加

at_command.c で定義されているポインタ配列 gp_at_command[] へ、4.5.3 節で追加した文字列のアドレスを追加します。

必ず、図 37 で追加したメンバに合わせた位置に追加してください。ATC_GET_OPERATOR_LIST の値が 72 のため、図 39 では配列 gp_at_command[] の要素番号 72 の位置、すなわち 73 番目にメンバを追加しています。

```
190     g_ryz014_get_mt_state,
191     g_ryz014_get_operator_list,
192 #if (CELLULAR_IMPLEMENT_TYPE == 'B')
```

図 39　配列 gp_at_command に新規追加した状態

4.5.5 新規ファイルの作成

「r_cellular/src/at_command/ryz014」フォルダ内に、新規ファイルを作成します。

既存のファイル（ceer.c 等の、引数のないタイプ）をコピー＆リネームすることを推奨します。

図 40　新規ファイル「copn.c」を追加した状態

4.5.6 AT コマンド実行関数を作成

4.5.5 節で追加した.c ファイルに、「AT+COPN」コマンドを実行する関数を作成します。

Examples

```
#include "at_command.h"
#include "cellular_private_api.h"

e_cellular_err_t atc_copn(st_cellular_ctrl_t * const p_ctrl)
{
    e_cellular_err_t ret = CELLULAR_SUCCESS;

    atc_generate(p_ctrl->sci_ctrl.atc_buff,
                 gp_at_command[ATC_GET_OPERATOR_LIST],①
                 NULL);

    ret = cellular_execute_at_command(p_ctrl,
                                      p_ctrl->sci_ctrl.atc_timeout,
                                      ATC_RETURN_OK,
                                      ATC_GET_OPERATOR_LIST);②

    return ret;
}
```

① atc_generate() 関数を使用し、Cellular モジュールへ送信する AT コマンドを生成します。

- 第 1 引数 … 「p_ctrl->sci_ctrl.atc_buff」を設定
- 第 2 引数 … 「gp_at_command [図 37 で追加したメンバを指定]」を設定
- 第 3 引数 … AT コマンドに引数が存在しないため、NULL を設定

② cellular_execute_at_command() 関数を使用し、Cellular モジュールへ AT コマンドを送信します。

- 第 1 引数 … 「p_ctrl」を設定
- 第 2 引数 … Cellular モジュールからレスポンスが返るまでのタイムアウト時間を設定
- 第 3 引数 … Cellular モジュールから送られてくるレスポンスの期待値を設定
  一部のコマンド以外は、ATC_RETURN_OK を設定
- 第 4 引数 … 送信する AT コマンド番号（図 37 で追加したメンバ）を設定

4.5.7 新規作成した関数の宣言を追加

at_command.h へ、新規作成した関数「atc_copn()」の宣言を追加します。

```
912 /***********************************************************************************************
913  * Function Name  @fn            atc_copn
914  * Description    @details       Execute the AT command (COPN). / Obtains operator names.
915  * Arguments      @param[in/out] p_ctrl -
916  *                                  Pointer to managed structure.
917  * Return Value   @retval        CELLULAR_SUCCESS -
918  *                                  Successfully executed AT command.
919  *                @retval        CELLULAR_ERR_MODULE_COM -
920  *                                  Communication with module failed.
921  ***********************************************************************************************/
922 e_cellular_err_t atc_copn (st_cellular_ctrl_t * const p_ctrl);
```

図 41　「atc_copn()」の宣言を追加した状態

4.5.8 マクロを追加

cellular_receive_task.h へ、レスポンス文字列「+COPN:」のマクロ定義を追加します。

```
80 #define ATC_RES_GET_MT_STATE        "+CPAS:"
81 #define ATC_RES_GET_OPERATOR_LIST   "+COPN:"
```

図 42　「+COPN:」のマクロ定義を追加した状態

4.5.9 列挙型へ新規メンバを追加

cellular_receive_task.h で定義されている列挙型 e_atc_return_code_t へ、新規メンバを追加します。

図 43 の位置に追加した場合、CELLULAR_GET_MT_STATE の値は 42 となります。

```
129     CELLULAR_GET_MT_STATE,              // Get Phone Activity Status
130     CELLULAR_GET_OPERATOR_LIST,         // Get Operator names
131     CELLULAR_RES_MAX,                   // End of analysis result processing
```

図 43　列挙型「e_atc_return_code_t」に新規メンバを追加した状態

4.5.10 定数の追加および文字列の格納

r_cellular_receive_task.c へ文字列定数を追加し、4.5.8 節で追加した文字列を格納します。

```
97 static const uint8_t s_atc_res_get_mt_state[]       = ATC_RES_GET_MT_STATE;
98 static const uint8_t s_atc_res_get_operator_list[]  = ATC_RES_GET_OPERATOR_LIST;
```

図 44　文字列定数を追加した状態

4.5.11 文字列のアドレスを追加

r_cellular_receive_task.c で定義されているポインタ配列 sp_cellular_atc_res_tbl[] へ、4.5.10 節で追加した文字列のアドレスを追加します。

必ず、図 43 で追加したメンバに合わせた位置に追加します。CELLULAR_GET_MT_STATE の値が 42 のため、図 45 では配列 sp_cellular_atc_res_tbl[] の要素番号 42、すなわち 43 番目にメンバを追加しています。

```
143     s_atc_res_get_mt_state,
144     s_atc_res_get_operator_list,
145 };
```

図 45　配列 sp_cellular_atc_res_tbl[] に新規追加した状態

4.5.12 関数の宣言を追加

r_cellular_receive_task.c へ、レスポンス「+COPN:」受信時の処理を実装する関数の宣言を追加します。

引数は他の関数と同じにしてください。

```
186 static void cellular_get_mt_state (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
187 static void cellular_get_operator_list (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
188 static void cellular_get_revision (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
```

図 46　「+COPN:」受信時に処理を行う関数を宣言した状態

4.5.13 関数ポインタテーブルへ関数を登録

r_cellular_receive_task.c で定義されている関数ポインタテーブル p_cellular_recvtask_api[] へ、4.5.12 節で宣言した関数を登録します。

必ず、図 43 で追加したメンバに合わせた位置に追加します。CELLULAR_GET_MT_STATE の値が 42 のため、図 47 では配列 p_cellular_recvtask_api [] の要素番号 42、すなわち 43 番目にメンバを追加しています。

```
247     cellular_get_mt_state,
248     cellular_get_operator_list,
249     cellular_response_skip,
```

図 47　新規に宣言した関数を追加した状態

4.5.14 関数を追加

r_cellular_receive_task.c へ、4.5.12 節で宣言した関数を追加します。

Examples

```
static void cellular_get_operator_list(st_cellular_ctrl_t     * p_ctrl,
                                       st_cellular_receive_t * cellular_receive)
{
    st_cellular_receive_t * p_cellular_receive = cellular_receive;
    sci_err_t               sci_ret;

    if (CHAR_CHECK_4 == p_cellular_receive->data) ①
    {
        cellular_cleardata(p_ctrl, p_cellular_receive); ②

        do
        {
            sci_ret = R_SCI_Receive(p_ctrl->sci_ctrl.sci_hdl,
                                    &p_cellular_receive->data, 1);
            cellular_delay_task(1);
        } while (SCI_SUCCESS != sci_ret);                          ③

        p_ctrl->sci_ctrl.receive_buff[0] = p_cellular_receive->data;
        p_cellular_receive->recv_count++;
        if (CHAR_CHECK_1 == p_cellular_receive->data)
        {
            p_cellular_receive->job_no      = CELLULAR_GET_OPERATOR_LIST;
            p_cellular_receive->tmp_recvcnt = 6;
        }                                                          ④
    }

    return;
}
```

① 終端文字 (¥n) を確認し、処理を開始します。

② 受信した文字列を削除します。

③ 連続して+COPN:が送られてきていることを確認するため、1 文字受信するまでループします。
図 35 にある通り、+COPN:~<s3><s4>+COPN:と連続して送られてきます。
<s3> =’ ¥r’、<s4> = ‘¥n’であり、①の終端文字’¥n’がここでの<s4>にあたります。
そのため、連続して+COPN:が送られてきている場合、次に’+’を受信することになります。

④ ③で受信した 1 文字を recv_task の受信バッファへ格納し、受信文字数をインクリメントします。
その後、受信した文字の内容を確認します。
受信した文字が’+’であった場合、連続して+COPN:が送られてきたということになるので、job_no に
図 43 で追加したメンバを代入し、Examples の関数が呼び出されるようにします。

4.6　正常処理時のレスポンスで Intermediate result code が返却されるコマンド

正常処理時のレスポンスで Intermediate result code が返される AT コマンドを追加する場合のソースコード修正手順を説明します。

ここでは、実装済みの AT+SQNSSENDEXT コマンドを例として説明します。

```
AT+SQNSSENDEXT=1,10

> 0123456789
OK
```

2.2.6 Extended Send Data In Command Mode: AT+SQNSSENDEXT
2.2.6.1 Syntax

Command	Possible Response(s)
AT+SQNSSENDEXT=<connId>,<bytesToSend>[,<RAI>]	Intermediate result code: > OK ERROR NO CARRIER +CME ERROR:<err>

図 48　Intermediate result code が返却される AT コマンド「AT+SQNSSENDEXT」

4.6.1 マクロを追加

ryz014a_private.h へ、追加するＡＴコマンドのフォーマットを定義するマクロを追加します。

```
95  #define RYZ014_ATC_CLOSE_SOCKET             "AT+SQNSH=%s\r"
96  #define RYZ014_ATC_SEND_SCOKET              "AT+SQNSSENDEXT=%s,%s\r"
97  #define RYZ014_ATC_RECV_SCOKET              "AT+SQNSRECV=%s,%s\r"
```

図 49　「AT+SQNSSENDEXT」を追加した状態

4.6.2 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_atc_list_t へ、新規メンバを追加します。

図 50 の位置に追加した場合、ATC_SEND_SOCKET の値は 7 となります。

```
178     ATC_CLOSE_SOCKET,
179     ATC_SEND_SOCKET,
180     ATC_RECV_SOCKET,
```

図 50　列挙型 e_atc_list_t に新規メンバを追加した状態

4.6.3 定数の追加および文字列の格納

at_command.c に文字列定数を追加し、4.6.1 節で追加した文字列を格納します。

```
47  const uint8_t g_ryz014_close_socket[]           = RYZ014_ATC_CLOSE_SOCKET;
48  const uint8_t g_ryz014_send_socket[]            = RYZ014_ATC_SEND_SCOKET;
49  const uint8_t g_ryz014_recv_socket[]            = RYZ014_ATC_RECV_SCOKET;
```

図 51　文字列定数を追加した状態

4.6.4 文字列のアドレスを追加

at_command.c で定義済みのポインタ配列 gp_at_command[] へ、4.6.3 節で追加した文字列のアドレスを追加します。

必ず、図 50 で追加したメンバに合わせた位置に追加します。ATC_SEND_SOCKET の値が 7 のため、図 52 では配列 gp_at_command[] の要素番号 7 の位置、すなわち 8 番目にメンバを追加しています。

```
127     g_ryz014_close_socket,
128     g_ryz014_send_socket,
129     g_ryz014_recv_socket,
```

図 52　配列 gp_at_command[] に新規追加した状態

4.6.5 マクロを追加

cellular_receive_task.h へ、Intermediate result code 文字列のマクロ定義を追加します。

```
39  #define ATC_RES_GO_SEND                 ">"
40  #define ATC_RES_OK                      "OK\r\n"
```

図 53　Intermediate result code のマクロ定義を追加した状態

4.6.6 列挙型へ新規メンバを追加

cellular_receive_task.h で定義されている列挙型 e_atc_return_code_t へ、新規メンバを追加します。

```
88      CELLULAR_RES_GO_SEND = 0,           // Request for Data Transmission
89      CELLULAR_RES_OK,                    // Response is OK
```

図 54　列挙型 e_atc_return_code_t に新規メンバを追加した状態

4.6.7 定数を追加および文字列の格納

r_cellular_receive_task.c に文字列定数を追加し、4.6.5 節で追加した文字列を格納します。

```
56  static const uint8_t s_atc_res_go_send[]            = ATC_RES_GO_SEND;
57  static const uint8_t s_atc_res_ok[]                 = ATC_RES_OK;
```

図 55　文字列定数を追加した状態

4.6.8 文字列のアドレスを追加

r_cellular_receive_task.c で定義されているポインタ配列 sp_cellular_atc_res_tbl[] へ、4.6.7 節で追加した文字列のアドレスを追加します。

必ず、図 54 で追加したメンバに合わせた位置に追加します。CELLULAR_RES_GO_SEND の値が 0 のため、図 56 では配列 sp_cellular_atc_res_tbl[] の要素番号 0、すなわち 1 番目にメンバを追加しています。

```
102     s_atc_res_go_send,
103     s_atc_res_ok,
```

図 56　配列 sp_cellular_atc_res_tbl[] に新規追加した状態

4.6.9 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_cellular_atc_return_t へ、新規メンバを追加します。

追加する位置は、ATC_RETURN_ENUM_MAX より上 (先頭側) の位置にしてください。

```
252 typedef enum
253 {
254     ATC_RETURN_NONE = 0,          // No response from the module
255     ATC_RETURN_OK,                // Module response is "OK"
256     ATC_RETURN_ERROR,             // Module response is "ERROR"
257     ATC_RETURN_OK_GO_SEND,        // Module response is ">"
258     ATC_RETURN_SEND_NO_CARRIER, // Module response is "NO CARRIER"
259     ATC_RETURN_AP_CONNECTING,     // Module response is "CONNECT"
260     ATC_RETURN_ENUM_MAX,          // Maximum enumeration value
261 } e_cellular_atc_return_t;
```

図 57　列挙型 e_cellular_atc_return_t に新規メンバを追加した状態

4.6.10 新規ファイルの作成

「r_cellular/src/at_command/ryz014」フォルダ内に、新規ファイルを作成します。

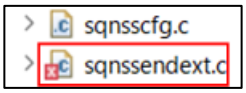

図 58　新規ファイル「sqnssendext.c」を追加した状態

4.6.11 AT コマンド実行用の定義を作成

4.6.10 節で作成した.c ファイルに、「AT+SQNSSENDEXT」コマンドを実行する関数を作成します。

Examples

```
#include "at_command.h"
#include "cellular_private_api.h"

e_cellular_err_t atc_sqnssendext(st_cellular_ctrl_t * const p_ctrl,
                                 const uint8_t socket_no,
                                 const uint16_t length)
{
    uint8_t          str[2][10]                                  = {0};  ①
    const uint8_t * p_command_arg[CELLULAR_MAX_ARG_COUNT] = {0};
    e_cellular_err_t ret                                          = CELLULAR_SUCCESS;

    sprintf((char *)str[0], "%d", socket_no);   // (uint8_t *)->(char *)
    sprintf((char *)str[1], "%d", length);      // (uint8_t *)->(char *)
                                                                          ②
    p_command_arg[0] = str[0];
    p_command_arg[1] = str[1];

    atc_generate(p_ctrl->sci_ctrl.atc_buff,
                 gp_at_command[ATC_SEND_SOCKET],                          ③
                 p_command_arg);

    ret = cellular_execute_at_command(p_ctrl,
                                      p_ctrl->sci_ctrl.atc_timeout,       ④
                                      ATC_RETURN_OK_GO_SEND,
                                      ATC_SEND_SOCKET);

    return ret;
}
```

① AT コマンドへ与える引数用の変数です。

② AT コマンドへ与える引数を変数「str」に格納し、さらに「str」に格納した文字列の先頭アドレスを変数「p_command_arg」に格納します。

③ atc_generate() 関数を使用し、Cellular モジュールへ送信する AT コマンドを生成します。

- 第 1 引数 … p_ctrl->sci_ctrl.atc_buff を設定
- 第 2 引数 … 「gp_at_command [図 50 で追加したメンバを指定]」を設定
- 第 3 引数 … ②で用意した変数「p_command_arg」を設定

④ cellular_execute_at_command() 関数を使用し、Cellular モジュールへ AT コマンドを送信します。

- 第 1 引数 … 「p_ctrl」を設定
- 第 2 引数 … Cellular モジュールからレスポンスが返るまでのタイムアウト時間を指定可能
- 第 3 引数 … Cellular モジュールから送られてくるレスポンスの期待値
  図 57 で追加したメンバを設定する
- 第 4 引数 … 送信する AT コマンド番号 (図 50 で追加したメンバを指定)

4.6.12 マクロを追加

r_cellular_receive_task.c へ、Intermediate result code のマクロ定義を追加します。Intermediate result code を受信したことを判定するために使用します。

```
36  #define CHAR_CHECK_1     ('+')
37  #define CHAR_CHECK_2     ('>')
```

図 59　Intermediate result code のマクロ定義を追加した状態

4.6.13 case 文の処理を修正

r_cellular_receive_task.c の cellular_job_check() 関数内へ、図 60 に示す処理を追加します。

if 文で使用するマクロは 4.6.12 節で追加したマクロを指定してください。また、「p_cellular_receive->job_no」には、4.6.6 節で追加したメンバを設定してください。

```
347         case JOB_STATUS_FIRST_CHAR_CHECK:
348         {
349             if ((char)p_cellular_receive->data == (CHAR_CHECK_1))
350             {
351                 p_cellular_receive->job_status = JOB_STATUS_COLON_CHECK;
352             }
353             else if ((char)p_cellular_receive->data == (CHAR_CHECK_2))
354             {
355                 p_cellular_receive->job_no = CELLULAR_RES_GO_SEND;
356             }
357             else
358             {
359                 p_cellular_receive->job_no = CELLULAR_RES_CHECK;
360             }
361             break;
362         }
```

図 60　cellular_job_check() 関数に処理を追加した状態

4.6.14 関数の宣言を追加

r_cellular_receive_task.c へ、Intermediate result code 受信時の処理を実装する関数の宣言を追加します。

引数は他の関数と同じにしてください。

```
150 static void cellular_response_check (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
151 static void cellular_data_send_command (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
152 static void cellular_get_data_reception (st_cellular_ctrl_t * p_ctrl, st_cellular_receive_t * cellular_receive);
```

図 61　Intermediate result code 受信時に処理を行う関数を宣言した状態

4.6.15 関数ポインタテーブルへ関数を登録

r_cellular_receive_task.c で定義されている関数ポインタテーブル p_cellular_recvtask_api[] へ、4.6.14 節で宣言した関数を登録します。

必ず、図 54 で追加したメンバに合わせた位置に追加します。CELLULAR_RES_GO_SEND の値が 0 のため、図 62 では配列 p_cellular_recvtask_api[] の要素番号 0、すなわち 1 番目にメンバを追加しています。

```
206     cellular_data_send_command,
207     cellular_memclear,
```

図 62　新規に宣言した関数を追加した状態

4.6.16 関数を追加

r_cellular_receive_task.c へ、4.6.14 節で宣言した関数を追加します。

Examples

```
static void cellular_data_send_command(st_cellular_ctrl_t * p_ctrl,
                                       st_cellular_receive_t * cellular_receive)
{
    st_cellular_receive_t * p_cellular_receive = cellular_receive;

    if (CHAR_CHECK_6 == p_cellular_receive->data)  ①
    {
        cellular_set_atc_response(p_ctrl, ATC_RETURN_OK_GO_SEND); ②
        cellular_memclear(p_ctrl, p_cellular_receive); ③
    }

    return;
}
```

① Intermediate result code の次に送られてくる文字列が正常であることを確認しています。
RYZ014A Cellular モジュールは「>」の後に半角スペースが送られてくるため、半角スペース確認用のマクロを定義しています。

```
41  #define CHAR_CHECK_6    (' ')
```

② ①の判定文で「> 」の受信が確認できたため、Intermediate result code の受信が正常に完了した結果を cellular_set_atc_response() 関数によって通知します。
第 2 引数には、4.6.11 節の④で呼び出している cellular_execute_at_command() 関数の第 3 引数と同じ値を設定してください。

③ AT コマンドの処理が Intermediate result code の処理まで完了したので、cellular_memclear() 関数で情報をクリアします。

4.6.17 列挙型へ新規メンバを追加

ryz014a_private.h で定義されている列挙型 e_atc_list_t に新規メンバを追加します。

Intermediate result code には、4.6.1 節で定義しているような対応する AT コマンド文字列マクロが存在しないため、必ず ATC_LIST_MAX の上に追加してください。

```
248     ATC_SQNSSENDEXT_END,
249     ATC_LIST_MAX
250 } e_atc_list_t;
```

図 63　列挙型 e_atc_list_t に新規メンバを追加した状態

4.6.18 後処理を作成

Intermediate result code 受信処理後の処理を作成します。

RYZ014A Cellular ドライバでは、Intermediate result code「>」を受信後、続けて送信データを Cellular モジュールへ送信します。

Examples

r_cellular_sendsocket.c の cellular_send_data() 関数の処理を一部抜粋。

```
ret = atc_sqnssendext(p_ctrl, socket_no, send_size); ①
if (CELLULAR_SUCCESS != ret)
{
    break;
}
p_ctrl->sci_ctrl.tx_end_flg = CELLULAR_TX_END_FLAG_OFF; ②

sci_ret = R_SCI_Send(p_ctrl->sci_ctrl.sci_hdl,
                     (uint8_t *)p_data + complete_length, send_size); ③

if (SCI_SUCCESS != sci_ret)
{
    ret = CELLULAR_ERR_MODULE_COM; ④
    break;
}

cellular_set_atc_number(p_ctrl, ATC_SQNSSENDEXT_END); ⑤

timeout = cellular_tx_flag_check(p_ctrl, socket_no); ⑥
if (CELLULAR_TIMEOUT != timeout)
{
    timeout = cellular_atc_response_check(p_ctrl, socket_no); ⑦
}

if (CELLULAR_TIMEOUT == timeout)
{
    ret = CELLULAR_ERR_MODULE_TIMEOUT; ⑧
    break;
}
```

① 4.6.11 節で作成した関数を実行します。

② ①が正常終了したため、続けて送信データを Cellular モジュールへ送信する準備を行います。
送信データが正常に完了したことを確認するフラグを初期化します。

③ 送信データを Cellular モジュールへ送信するため、SCI FIT の R_SCI_Send() 関数を実行します。

④ ③の結果を確認します。
異常終了していた場合はデータ送信処理を強制的に終了します。

⑤ R_SCI_Send() 関数が正常に完了したため、Cellular モジュールからのレスポンス文字列を確認します。
cellular_set_atc_number() 関数を実行し、レスポンス文字列の処理準備を行います。
cellular_set_atc_number() 関数の第 2 引数には 4.6.17 節で追加したメンバを指定してください。

⑥ Cellular モジュールへのデータ送信が完了したことを確認します。

⑦ Cellular モジュールからレスポンス文字列「OK」が返ってくることを確認します。

⑧ ⑦でレスポンス文字列「OK」を確認できなかった場合、データ送信処理を強制的に終了します。

5. 他の Cellular モジュールで流用可能な処理

RYZ014A Cellular FIT モジュールでは、3GPP 規格準拠の AT コマンドと、RYZ014A Cellular モジュール専用の AT コマンド (コマンドの先頭に SQN が付与されるコマンド) を使用しています。3GPP 規格準拠の AT コマンドは、他の Cellular モジュールを使用する場合でも流用が可能ですが、RYZ014A Cellular モジュール専用の AT コマンドを実行する処理およびそれらの実行結果の処理は、使用する Cellular モジュールに応じて適切な処理に置き換える必要があります。

RYZ014A Cellular FIT モジュールで実装済みの 3GPP 規格準拠の AT コマンドを流用する場合の処理例は、以下の通りです。

① AT コマンド用セマフォを取得

② AT コマンド実行用の関数を実行

③ AT コマンド用セマフォを解放

Cellular FIT モジュールで実装済みの AT コマンド実行関数を使用する場合は、R_CELLULAR_Open() 関数が実行済みである必要があります。

Examples

```
e_cellular_err_t           ret           = CELLULAR_SUCCESS;
e_cellular_err_semaphore_t semahore_ret = CELLULAR_SEMAPHORE_SUCCESS;

semahore_ret = cellular_take_semaphore(p_ctrl->at_semaphore); ①
if (CELLULAR_SEMAPHORE_SUCCESS == semahore_ret)
{
    atc_cfun(p_ctrl, CELLULAR_MODULE_OPERATING_LEVEL4); ②
    cellular_give_semaphore(p_ctrl->at_semaphore); ③
}
else
{
    ret = CELLULAR_ERR_OTHER_ATCOMMAND_RUNNING;
}
```

① AT コマンド用 API を実行する前に、AT コマンド用セマフォを取得します。
p_ctrl は、R_CELLULAR_Open() 関数実行時に第 1 引数へ設定した st_cellular_ctrl_t のポインタです。

② 3GPP 規格準拠の AT コマンド「AT+CFUN」コマンドを送信する関数を実行します。
第 2 引数へ、設定値を指定します。

③ AT コマンド用セマフォを解放します。

5.1　R_CELLULAR_Open()

R_CELLULAR_Open()の処理の流れは、以下の通りです。

① 引数確認およびスレッドセーフ有効化
② グローバル変数「gp_cellular_ctrl」へ引数「p_ctrl」を格納
③ 引数「p_cfg」に格納されたコンフィグ設定値を適用(「p_cfg」が NULL の場合はデフォルト値を使用)
④ シリアル通信機能を有効化
⑤ OS の機能を使用し、セマフォの取得、およびイベントグループおよびデータ受信タスクを作成
⑥ データ受信タスクと同期処理
⑦ ハードウェアリセット実行 (145 行目：cellular_module_reset() 関数を実行)
⑧ 「ATE0」コマンド発行
⑨ 「AT+SQNSIMST=0」コマンド発行
⑩ 「AT+CEREG＝x」コマンド発行 (x = CELLULAR_CFG_NETWORK_NOTIFY_LEVEL)
⑪ 「AT+CFUN=4」コマンド発行
⑫ 「AT+CPIN?」コマンド発行
PIN ロックが設定された SIM の場合は「AT+CPIN=x」コマンド発行
(x = CELLULAR_CFG_PIN_CODE または p_cfg->sim_pin_code)
⑬ スレッドセーフ無効化

5.1.1 流用可能な処理

①～⑥および⑬の処理は Cellular モジュールに関係なく流用可能です。ただし③で適用されるデフォルト値に関してはモジュール毎に値が異なる可能性があるため、「ryz014_private.h」で適切な値を設定してください。

5.1.2 一部置き換えが必要な処理

⑧、⑩～⑫の処理はモジュールに対して主要な AT コマンドを発行可能な状態とするための処理となります。モジュールによっては別途 AT コマンドの発行が必要になる可能性があるため、必要に応じて処理を追加してください。

⑦のハードウェアリセット実行用 private 関数内では、RYZ014A Cellular モジュール専用のコマンドが実行されるため、別モジュールでは使用不可となります。cellular_module_reset.c の 115 行目を削除するか、別の処理に置き換えてください。
atc_sqnautoconnect_chek() 関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1) に自動で設定されるか否かを確認します。

```
115             ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 64　cellular_module_reset.c の 115 行目

⑨のコマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_open.c の 289 行目を削除するか、別の処理に置き換えてください。

```
289         ret = atc_sqnsimst(p_ctrl);
```

図 65　r_cellular_open.c の 289 行目

5.2 R_CELLULAR_Close()

R_CELLULAR_Close()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② PSM (Power Saving Mode) 無効化
③ OS の機能を使用し、イベントグループ・セマフォ・タスクを削除 (PSM 制御関連)
④ ハードウェアリセット実行
⑤ モジュールをシャットダウン (cellular_power_down() 関数を実行)
「AT+SQNSSHDN」コマンドを実行
⑥ OS の機能を使用し、イベントグループ・セマフォ・タスクを削除 (AT コマンド制御関連)
⑦ OS の機能を使用し、セマフォを削除およびメモリを解放 (ソケット関連)
⑧ シリアル通信を無効化
⑨ スレッドセーフ無効化

5.2.1 流用可能な処理

①～④および⑥～⑨の処理は、Cellular モジュールに関係なく流用可能です。

5.2.2 置き換えが必要な処理⑤

⑤の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_power_down.c の 62 行目を適切な処理に置き換えてください。

```
62          ret                 = atc_sqnsshdn(p_ctrl);
```

図 66 cellular_power_down.c の 62 行目

5.3 R_CELLULAR_ APConnect()

R_CELLULAR_APConnect()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ AP 接続設定を実行 (cellular_apconnect_config() 関数を実行)
④ AP 接続を実行 (cellular_apconnect() 関数を実行)
⑤ AT コマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.3.1 流用可能な処理

①～⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.4　R_CELLULAR_IsConnected()

R_CELLULAR_IsConnected()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② 管理構造体より AP 接続状態を取得
③ スレッドセーフ無効化

5.4.1 流用可能な処理

①～③の処理は、Cellular モジュールに関係なく流用可能です。

5.5　R_CELLULAR_Disconnect()

R_CELLULAR_Disconnect()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケット切断処理を実行（cellular_shutdownsocket() 関数を実行）
「AT+SQNSH」コマンドを実行
③ ソケットクローズ処理を実行（cellular_closesocket() 関数を実行）
④ アクセスポイント切断処理を実行（cellular_disconnect() 関数を実行）
「AT+CFUN=4」コマンドを実行
⑤ スレッドセーフ無効化

5.5.1 流用可能な処理

①および③～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.5.2 置き換えが必要な処理

②の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_shutdownsocket.c の 76 行目を適切な処理に置き換えてください。

```
76              ret = atc_sqnsh(p_ctrl, socket_no);
```

図 67　cellular_shutdownsocket.c の 76 行目

5.6　R_CELLULAR_CreateSocket()

R_CELLULAR_CreateSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 未使用ソケット番号検索
④ ソケットコンフィグを実行 (cellular_socket_cfg() 関数を実行)
「AT+SQNSCFG=%s,1,%s,%s,%s,%s」、「AT+SQNSCFGEXT=%s,1,0,0」コマンドを実行
⑤ AT コマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.6.1 流用可能な処理

①～③、⑤および⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.6.2 置き換えが必要な処理

④の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。R_cellular_createsocket.c の 154、157 行目を適切な処理に置き換えてください。

```
154     atc_ret = atc_sqnscfg(p_ctrl, (uint8_t)(socket_num + start_num));
```

図 68　r_cellular_createsocket.c の 154 行目

```
157         atc_ret = atc_sqnscfgext(p_ctrl, (uint8_t)(socket_num + start_num));
```

図 69　r_cellular_createsocket.c の 157 行目

5.7　R_CELLULAR_ConnectSocket()

R_CELLULAR_ConnectSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ATコマンド用セマフォ取得
③ ソケット接続
「AT+SQNSD=%s,%s,%s,”%s”,0,%s,1」コマンドを実行
④ ソケット管理構造体のステータスを更新
⑤ ATコマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.7.1 流用可能な処理

①、②、⑤および⑥の処理は、Cellularモジュールに関係なく流用可能です。

5.7.2 置き換えが必要な処理

③のATコマンドは、RYZ014A Cellularモジュール専用のコマンドであるため、別モジュールでは使用不可となります。また、④の処理も③のATコマンドの実行結果の処理 (+から始まる受信文字列を処理) であるため、同様に別モジュールでは使用不可となります。r_cellular_connectsocket.cの114～131行目を適切な処理に置き換えてください。

```
114             ret = atc_sqnsd(p_ctrl, socket_no, p_ip_addr, port);
115             if (CELLULAR_SUCCESS == ret)
116             {
117                 p_ctrl->p_socket_ctrl[socket_no - CELLULAR_START_SOCKET_NUMBER].socket_status
118                     = CELLULAR_SOCKET_STATUS_CONNECTED;
119                 if (CELLULAR_PROTOCOL_IPV4 ==
120                         p_ctrl->p_socket_ctrl[socket_no - CELLULAR_START_SOCKET_NUMBER].ipversion)
121                 {
122                     strncpy((char *)p_ctrl->p_socket_ctrl[socket_no - CELLULAR_START_SOCKET_NUMBER].ip_addr.ipv4,
123                             (char *)p_ip_addr, CELLULAR_IPV4_ADDR_LENGTH); //(uint8_t *)->(char *)
124                 }
125                 else
126                 {
127                     strncpy((char *)p_ctrl->p_socket_ctrl[socket_no - CELLULAR_START_SOCKET_NUMBER].ip_addr.ipv6,
128                             (char *)p_ip_addr, CELLULAR_IPV6_ADDR_LENGTH); //(uint8_t *)->(char *)
129                 }
130                 p_ctrl->p_socket_ctrl[socket_no - CELLULAR_START_SOCKET_NUMBER].port = port;
131             }
```

図 70　r_cellular_connectsocket.c の 114～131 行目

5.8 R_CELLULAR_ShutdownSocket()

R_CELLULAR_ShutdownSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケット切断処理を実行 (cellular_shutdownsocket() 関数を実行)
「AT+SQNSH=%s」コマンドを実行
③ スレッドセーフ無効化

5.8.1 流用可能な処理

①および③の処理は、Cellular モジュールに関係なく流用可能です。

5.8.2 置き換えが必要な処理

②の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_shutdownsocket.c の 76 行目を適切な処理に置き換えてください。

```
76              ret = atc_sqnsh(p_ctrl, socket_no);
```

図 71 cellular_shutdownsocket.c の 76 行目

5.9 R_CELLULAR_CloseSocket()

R_CELLULAR_CloseSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケット切断処理を実行 (cellular_shutdownsocket() 関数を実行)
「AT+SQNSH=%s」コマンドを実行
③ ソケットクローズ処理を実行 (cellular_closesocket()を実行)
④ スレッドセーフ無効化

5.9.1 流用可能な処理

①、③および④の処理は、Cellular モジュールに関係なく流用可能です。

5.9.2 置き換えが必要な処理

②の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_shutdownsocket.c の 76 行目を適切な処理に置き換えてください。

```
76              ret = atc_sqnsh(p_ctrl, socket_no);
```

図 72 cellular_shutdownsocket.c の 76 行目

5.10 R_CELLULAR_SendSocket()

R_CELLULAR_SendSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ ソケットデータ送信を実行 (cellular_send_data() 関数を実行)
「AT+SQNSSENDEXT=%s,%s」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.10.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.10.2 置き換えが必要な処理

③のソケットデータ送信処理は、RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_send_data()の 208 行目と処理内容を適切な処理に置き換えてください。
置き換え処理は 4.6 節を参考にしてください

```
208         ret = atc_sqnssendext(p_ctrl, socket_no, send_size);
```

図 73　r_cellular_sendsocket.c の 208 行目

5.11 R_CELLULAR_ReceiveSocket()

R_CELLULAR_ReceiveSocket()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケットデータ受信用セマフォ取得
③ ソケットデータ受信を実行 (cellular_receive_data() 関数を実行)
「AT+SQNSRECV=%s,%s」コマンドを実行
④ ソケットデータ受信用セマフォ開放
⑤ スレッドセーフ無効化

5.11.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.11.2 置き換えが必要な処理

③のソケットデータ受信処理は、RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_receive_data()の処理内容 (161~270 行目) を適切な処理に置き換えてください。

5.11.3 cellular_receive_data() 関数の処理内容

219 行目：Cellular モジュールからのデータ受信通知待ちをしています
→R_CELLULAR_CreateSocket()実行時に AT コマンドで「AT+SQNSCFGEXT=%s,1,0,0」を実行し、Cellular モジュールが接続先からデータを受信した場合、「+SQNSRING:<connId>,<recData>」を通知するように設定しています。この文字列を r_cellular_receive_task.c の 544~575 行目で処理することにより、どのソケットに何 byte のデータが蓄積されたかを確認しています。これにより、データ受信コマンド発行前にデータの蓄積状態が確認できるため、受信データが蓄積されていない場合は受信コマンドを発行しないという処理が可能となります。
※不要な場合は、削除しても問題ありません
230、231 行目：受信要求データサイズを決定しています。
→要求受信データサイズと実際に受信可能なデータサイズ、および 1 回の処理で受信可能なデータサイズ上限を比較し、適切なものを選択しています。Cellular モジュールの仕様次第では、1 回の処理で受信可能なデータサイズを指定しても問題ありません。Cellular モジュールの仕様に合わせて修正してください。
235 行目：データ受信要求コマンドを発行しています。
→RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。
r_cellular_receivesocket.c の 235 行目を適切な処理に置き換えてください。

```
235             ret = atc_sqnsrecv(p_ctrl, socket_no, receive_size);
```

図 74 r_cellular_receivesocket.c の 235 行目

「AT+SQNSRECV」コマンドのレスポンス文字列は「+SQNSRECV:<connId>,<maxByte><CR><LF><data>」であるため、r_cellular_receive_task.c の 580~618 行目の処理で<LF>までの文字列の受信処理を、<data>の受信処理を r_cellular_receive_task.c の 631~654 行目で行います。こちらも必要に応じて Cellular モジュールの仕様に合わせて修正してください。
240 行目：cellular_receive_data()実行中の合計受信データサイズを取得しています
→データ受信処理を r_cellular_receive_task.c の 631~654 行目で行っており、合計受信データサイズを p_ctrl->p_socket_ctrl[p_cellular_receive->socket_no].total_recv_count でカウントしています。

5.12 R_CELLULAR_DnsQuery()

R_CELLULAR_DnsQuery()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ DNS クエリを実行
「AT+SQNDNSLKUP="%s",%s」コマンドを実行
④ ③で取得した文字列を第 4 引数「p_addr」に格納
⑤ AT コマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.12.1 流用可能な処理

①、②、⑤および⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.12.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。また、④の処理も③の処理結果の反映であるため、別モジュールでは使用不可となります。r_cellular_dnsquery.c の 116 行目、120 行目を適切な処理に置き換えてください。

```
116                     ret = atc_sqndnslkup(p_ctrl, p_domain_name, ip_version);
```

図 75　r_cellular_dnsquery.c の 116 行目

```
120                       cellular_getip(p_ctrl, ip_version, p_addr);
```

図 76　r_cellular_dnsquery.c の 120 行目

5.13 R_CELLULAR_GetTime()

R_CELLULAR_GetTime()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 時刻取得コマンドを実行
「AT+CCLK?」コマンド発行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.13.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.14 R_CELLULAR_SetTime()

R_CELLULAR_SetTime()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 時刻設定コマンドを実行
「AT+CCLK="%s/%s/%s,%s:%s:%s%s"」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.14.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.15 R_CELLULAR_SetEDRX()

R_CELLULAR_SetEDRX()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ eDRX 設定、設定取得コマンドを実行
「AT+SQNEDRX=%s,4,"%s","%s"」コマンドを実行
「AT+SQNEDRX?」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.15.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.15.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_setedrx.c の 93 行目、97 行目を適切な処理に置き換えてください。

```
93              ret = atc_sqnedrx(p_ctrl, p_config);
```

図 77　r_cellular_setedrx.c の 93 行目

```
97                  ret                 = atc_sqnedrx_check(p_ctrl);
```

図 78　r_cellular_setedrx.c の 97 行目

5.16 R_CELLULAR_GetEDRX()

R_CELLULAR_GetEDRX()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ eDRX 設定取得コマンドを実行
「AT+SQNEDRX?」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.16.1 流用可能な処理

①、②、④および⑤の処理は Cellular モジュールに関係なく流用可能です。

5.16.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_getedrx.c の 84 行目を適切な処理に置き換えてください。

```
84              ret                 = atc_sqnedrx_check(p_ctrl);
```

図 79　r_cellular_getedrx.c の 84 行目

5.17 R_CELLULAR_SetPSM()

R_CELLULAR_SetPSM()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ PSM コンフィグ実行 (97 行目：cellular_psm_config() 関数を実行)
「AT+SQNRICFG=%s,3,%s」コマンドを実行
「AT+SQNIPSCFG=%s,%s」コマンドを実行
「AT+SQNPSCFG=%s」コマンドを実行
④ PSM 有効化/無効化、設定値取得コマンド実行
「AT+CPSMS=%s,,,"%s","%s”」コマンドを実行
「AT+CPSMS?」コマンドを実行
⑤ AT コマンド用セマフォ開放
⑥ ハードウェアリセット実行 (114 行目：cellular_module_reset() 関数を実行)
⑦ スレッドセーフ無効化

5.17.1 流用可能な処理

①、②、④、⑤および⑦の処理は、Cellular モジュールに関係なく流用可能です。

5.17.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_psm_config.c の 64、68、74、116、120、188 および 193 行目を適切な処理に置き換えてください。

```
64          ret = atc_sqnricfg(p_ctrl, CELLULAR_SQNRICFG_MODE);
116         ret = atc_sqnricfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
193         atc_sqnricfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
```

図 80　cellular_psm_config.c の 64、116、193 行目

```
68              ret         = atc_sqnipscfg(p_ctrl, CELLULAR_SQNIPSCFG_MODE);
120             ret = atc_sqnipscfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
188         atc_sqnipscfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
```

図 81　cellular_psm_config.c の 68、120、188 行目

```
74              ret         = atc_sqnpscfg(p_ctrl);
```

図 82　cellular_psm_config.c の 74 行目

⑥のハードウェアリセット実行用 private 関数内では、RYZ014A Cellular モジュール専用のコマンドが実行されるため、別モジュールでは使用不可となります。cellular_module_reset.c の 115 行目を削除するか、別の処理に置き換えてください。

atc_sqnautoconnect_chek() 関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1) に自動で設定されるか否かを確認します。

```
115                 ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 83　cellular_module_reset.c の 115 行目

5.18 R_CELLULAR_GetPSM()

R_CELLULAR_GetPSM()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ PSM 設定取得コマンドを実行
「AT+CPSMS?」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.18.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.19 R_CELLULAR_GetICCID()

R_CELLULAR_GetICCID()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ ICCID 取得コマンドを実行
「AT+SQNCCID?」コマンド発行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.19.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.19.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_geticcid.c の 84 行目を適切な処理に置き換えてください。

```
84                  ret                 = atc_sqnccid(p_ctrl);
```

図 84　r_cellular_geticcid.c の 84 行目

5.20 R_CELLULAR_GetIMEI()

R_CELLULAR_GetIMEI()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ IMEI 取得コマンドを実行
「AT+CGSN」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.20.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.21 R_CELLULAR_GetIMSI()

R_CELLULAR_GetIMSI()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ IMSI 取得コマンドを実行
「AT+CIMI」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.21.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.22 R_CELLULAR_GetPhonenum()

R_CELLULAR_GetPhonenum()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 電話番号取得コマンドを実行
「AT+CNUM」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.22.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.23 R_CELLULAR_GetRSSI()

R_CELLULAR_GetRSSI()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 電波品質取得コマンドを実行
「AT+CSQ」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.23.1 流用可能な処理

①～⑤の処理は Cellular モジュールに関係なく流用可能です。

5.24 R_CELLULAR_GetSVN()

R_CELLULAR_GetRSSI()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ SVN、リビジョン取得コマンドを実行
「AT+CGSN=3」コマンドを実行
「ATI1」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.24.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.25 R_CELLULAR_Ping()

R_CELLULAR_Ping()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ Ping コマンドを実行
「AT+PING="%s",%s,%s,%s,%s」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.25.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.26 R_CELLULAR_GetAPConnectState()

R_CELLULAR_GetAPConnectState()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ AP 接続状態通知レベル設定、AP 接続状態取得コマンドを実行
「AT+CEREG=%s」コマンドを実行
「AT+CEREG?」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.26.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.27 R_CELLULAR_GetCellInfo()

R_CELLULAR_GetCellInfo()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ ファンクションレベル取得、ファンクションレベル設定コマンドを実行
「AT+CFUN?」コマンド発行
「AT+CFUN=1」コマンド発行
④ セル情報取得コマンドを実行
「AT+SQNMONI=%s」コマンド発行
⑤ AT コマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.27.1 流用可能な処理

①～③、⑤および⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.27.2 置き換えが必要な処理

④の AT コマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_getcellinfo.c の 100 行目を適切な処理に置き換えてください。

```
100                             ret = atc_sqnmoni(p_ctrl, type);
```

図 85　r_cellular_getcellinfo.c の 100 行目

5.28 R_CELLULAR_AutoConnectConfig()

R_CELLULAR_AutoConnectConfig()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ AP 自動接続設定コマンドを実行
「AT+SQNAUTOCONNECT=%s」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.28.1 流用可能な処理

①、②、⑤および⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.28.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_autoconnectconfig.c の 83 行目を適切な処理に置き換えてください。

```
83              ret = atc_sqnautoconnect(p_ctrl, type);
```

図 86　r_cellular_autoconnectconfig.c の 83 行目

5.29 R_CELLULAR_SetOperator()

R_CELLULAR_SetOperator()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ オペレータ設定取得、オペレータ設定コマンドを実行 (cellular_set_operator() 関数を実行)
「AT+SQNCTM?」コマンドを実行
「AT+SQNCTM="%s"」コマンドを実行
④ ファンクションレベルチェックおよびファンクションレベル設定コマンドを実行
「AT+CFUN?」コマンドを実行
「AT+CFUN=4」コマンドを実行
⑤ AT コマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.29.1 流用可能な処理

①、②および④～⑥の処理は、Cellular モジュールに関係なく流用可能です。

5.29.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_setoperator.c の 127 行目、133 行目を適切な処理に置き換えてください。

```
127     ret                 = atc_sqnctm_check(p_ctrl);
133         ret                 = atc_sqnctm(p_ctrl, p_operator);
```

図 87　r_cellular_setoperator.c の 127 行目、133 行目

5.30 R_CELLULAR_SetBand()

R_CELLULAR_SetBand()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ オペレータ設定取得
「AT+SQNCTM?」コマンドを実行
④ バンド設定実行
「AT+SQNBANDSEL=0,"%s","%s"」コマンドを実行
⑤ ソフトリセット実行
「AT^RESET」コマンドを実行
⑥ ソケットクローズ処理を実行 (cellular_closesocket() 関数を実行)
⑦ 「+SYSSTART」検出処理を実行
⑧ Cellular モジュールリセット後のファンクションレベルチェックおよびファンクションレベル設定を実行
「AT+CFUN?」コマンドを実行
「AT+CFUN=4」コマンドを実行
⑨ AT コマンド用セマフォ開放
⑩ スレッドセーフ無効化

5.30.1 流用可能な処理

①、②および⑤～⑩の処理は、Cellular モジュールに関係なく流用可能です。

5.30.2 置き換えが必要な処理

③および④の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_setband.c の 93 行目、96 行目を適切な処理に置き換えてください。

```
93              ret                 = atc_sqnctm_check(p_ctrl);
96                  ret = atc_sqnbandsel(p_ctrl, ctm_name, p_band);
```

図 88　r_cellular_setband.c の 93 行目、96 行目

5.31 R_CELLULAR_GetPDPAddress()

R_CELLULAR_GetPDPAddress()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ PDP アドレス取得コマンドを実行
「AT+CGPADDR=1」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.31.1 流用可能な処理

①～⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.32 R_CELLULAR_FirmUpgrade()

R_CELLULAR_FirmUpgrade()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ ファームアップグレードコマンドを実行
「AT+SQNSUPGRADE="%s",%s,5,%s」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.32.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.32.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_firmupgrade.c の 90 行目を適切な処理に置き換えてください。

```
90              ret = atc_sqnsupgrade(p_ctrl, p_url, 0, command, spid);
```

図 89　r_cellular_firmupgrade.c の 90 行目

5.33 R_CELLULAR_FirmUpgradeBlocking()

R_CELLULAR_FirmUpgradeBlocking()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケット切断処理を実行 (cellular_shutdownsocket() 関数を実行)
「AT+SQNSH=%s」コマンドを実行
③ ソケットクローズ処理を実行 (cellular_closesocket() 関数を実行)
④ AT コマンド用セマフォ取得
⑤ ファームアップグレードコマンドを実行
「AT+SQNSUPGRADE="%s",%s,5,%s」コマンドを実行
⑥ ファンクションレベル取得、ファンクションレベル設定コマンドを実行
「AT+CFUN?」コマンドを実行
「AT+CFUN=4」コマンドを実行
⑦ AT コマンド用セマフォ開放
⑧ スレッドセーフ無効化

5.33.1 流用可能な処理

①、③、④および⑥～⑧の処理は、Cellular モジュールに関係なく流用可能です。

5.33.2 置き換えが必要な処理

②の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_shutdownsocket.c の 76 行目を適切な処理に置き換えてください。

```
76              ret = atc_sqnsh(p_ctrl, socket_no);
```

図 90　cellular_shutdownsocket.c の 76 行目

⑤の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_firmupgradeblocking.c の 141 行目を適切な処理に置き換えてください。

```
141     ret             = atc_sqnsupgrade(p_ctrl, p_url, 1, CELLULAR_FIRM_UPGRADE_BLOCKING, spid);
```

図 91　r_cellular_firmupgradeblocking.c の 141 行目

5.34 R_CELLULAR_GetUpgradeState()

R_CELLULAR_GetUpgradeState()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ATコマンド用セマフォ取得
③ ファームアップグレードコマンドを実行
「AT+SQNSUPGRADE?」コマンドを実行
④ ATコマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.34.1 流用可能な処理

①、②、④および⑤の処理は、Cellularモジュールに関係なく流用可能です。

5.34.2 置き換えが必要な処理

③のATコマンドは、RYZ014A Cellularモジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_getupgradestate.cの87行目を適切な処理に置き換えてください。

```
87                  ret                 = atc_sqnsupgrade_check(p_ctrl);
```

図 92　r_cellular_getupgradestate.c の 87 行目

5.35 R_CELLULAR_UnlockSIM()

R_CELLULAR_UnlockSIM()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ATコマンド用セマフォ取得
③ ファンクションレベル取得、ファンクションレベル設定コマンドを実行
「AT+CFUN?」コマンドを実行
「AT+CFUN=4」コマンドを実行
④ PINロック状態取得コマンド、PINロック解除コマンドを実行
「AT+CPIN?」コマンドを実行
「AT+CPIN="%s"」コマンドを実行
⑤ ATコマンド用セマフォ開放
⑥ スレッドセーフ無効化

5.35.1 流用可能な処理

①～⑥の処理は、Cellularモジュールに関係なく流用可能です。

5.36 R_CELLULAR_WriteCertificate()

R_CELLULAR_WriteCertificate()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 証明書書き込みコマンドを実行 (cellular_write_certificate() 関数を実行)
「AT+SQNSNVW=¥"%s¥",%s,%s」コマンド発行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.36.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.36.2 置き換えが必要な処理

④ の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_writecertificate.c の 181 行目と cellular_write_certificate() の処理内容を適切な処理に置き換えてください。置き換え処理は 4.6 節を参考にしてください。

```
181     ret = atc_sqnsnvw(p_ctrl, data_type, index, size);
```

図 93　r_cellular_writecertificate.c の 181 行目

5.37 R_CELLULAR_EraseCertificate()

R_CELLULAR_EraseCertificate()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ 証明書書き込みコマンドを実行 (0 で上書きして削除する)
「AT+SQNSNVW=¥"%s¥",%s,0」コマンドを実行
④ AT コマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.37.1 流用可能な処理

①、②、④および⑤の処理は、Cellular モジュールに関係なく流用可能です。

5.37.2 置き換えが必要な処理

③の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_erasecertificate.c の 85 行目を適切な処理に置き換えてください。

```
85              ret = atc_sqnsnvw_erase(p_ctrl, data_type, index);
```

図 94　r_cellular_erasecertificate.c の 85 行目

5.38 R_CELLULAR_GetCertificate()

R_CELLULAR_GetCertificate()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ATコマンド用セマフォ取得
③ 証明書読み出しコマンドを実行
「AT+SQNSNVR="%s",%s」コマンドを実行
④ ATコマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.38.1 流用可能な処理

①、②、④および⑤の処理は、Cellularモジュールに関係なく流用可能です。

5.38.2 置き換えが必要な処理

③のATコマンドは、RYZ014A Cellularモジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_getcertificate.cの87行目を適切な処理に置き換えてください。

```
87              ret                 = atc_sqnsnvr(p_ctrl, data_type, index);
```

図 95　r_cellular_getcertificate.c の 87 行目

5.39 R_CELLULAR_ConfigSSLProfile()

R_CELLULAR_ConfigSSLProfile()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ATコマンド用セマフォ取得
③ セキュリティプロファイル設定コマンドを実行
「AT+SQNSPCFG=%s,2,,%s,%s,%s,%s,¥"¥"¥r」コマンドを実行
④ ATコマンド用セマフォ開放
⑤ スレッドセーフ無効化

5.39.1 流用可能な処理

①、②、④および⑤の処理は、Cellularモジュールに関係なく流用可能です。

5.39.2 置き換えが必要な処理

③のATコマンドは、RYZ014A Cellularモジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_configsslprofile.cの92、93行目を適切な処理に置き換えてください。

```
92              ret = atc_sqnspcfg(p_ctrl, security_profile_id, cert_valid_level,
93                      ca_certificate_id, client_certificate_id, client_privatekey_id);
```

図 96　r_cellular_configsslprofile.c の 92、93 行目

5.40 R_CELLULAR_SoftwareReset()

R_CELLULAR_SoftwareReset()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ソケット切断処理を実行 (cellular_shutdownsocket() 関数を実行)
「AT+SQNSH=%s」コマンドを実行
③ ソケットクローズ処理を実行 (cellular_closesocket() 関数を実行)
④ AT コマンド用セマフォ取得
⑤ アクセスポイント自動接続設定取得コマンドを実行
「AT+SQNAUTOCONNECT?」コマンドを実行
⑥ アクセスポイント接続状態通知レベル設定コマンドを実行
「AT+CEREG=2」コマンドを実行
⑦ リセットマンドを実行
「AT^RESET」コマンドを実行
⑧ ファンクションレベル設定コマンドを実行
「AT+CFUN=4」コマンドを実行
⑨ AT コマンド用セマフォ開放
⑩ スレッドセーフ無効化

5.40.1 流用可能な処理

①、③、④および⑥～⑩の処理は、Cellular モジュールに関係なく流用可能です。

5.40.2 置き換えが必要な処理

②の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_shutdownsocket.c の 76 行目を適切な処理に置き換えてください。

```
76              ret = atc_sqnsh(p_ctrl, socket_no);
```

図 97　cellular_shutdownsocket.c の 76 行目

⑤の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_softwarereset.c の 133 行目を適切な処理に置き換え、または削除してください。
atc_sqnautoconnect_chek() 関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1)に自動で設定されるか否かを確認します。

```
133     ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 98　r_cellular_softwarereset.c の 133 行目

5.41 R_CELLULAR_HardwareReset()

R_CELLULAR_HardwareReset()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② ハードウェアリセット実行 (79 行目：cellular_module_reset() 関数を実行)
③ スレッドセーフ無効化

5.41.1 流用可能な処理

①、③の処理は、Cellular モジュールに関係なく流用可能です。

5.41.2 置き換えが必要な処理

②のハードウェアリセット実行用 private 関数内では、RYZ014A Cellular モジュール専用のコマンドが実行されるため、別モジュールでは使用不可となります。cellular_module_reset.c の 115 行目を削除するか、別の処理に置き換えてください。

atc_sqnautoconnect_chek() 関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1)に自動で設定されるか否かを確認します。

```
115             ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 99　cellular_module_reset.c の 115 行目

5.42 R_CELLULAR_FactoryReset()

R_CELLULAR_FactoryReset()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② AT コマンド用セマフォ取得
③ PDP コンテキスト使用状況を取得
「AT+CGDCONT?」コマンドを実行
④ ③で取得した情報から、未使用コンテキスト番号を指定してダミーデータを登録
「AT+CGDCONT=%s,"IPV4V6","%s"」コマンドを実行
⑤ ファクトリーリセットを実行
「AT+SQNSFACTORYRESET」コマンドを実行
⑥ ハードウェアリセット実行 (141 行目：cellular_module_reset() 関数を実行)
⑦ PDP コンテキスト使用状況を取得
「AT+CGDCONT?」コマンドを実行
⑧ ⑦で取得した PDP コンテキスト情報に、④で登録したダミーデータが残っていないことを確認
※ 「AT+SQNSFACTORYRESET」コマンドは復元ポイントが作成されていない場合、ファクトリーリセット処理が正常終了した場合でもレスポンスで「ERROR」が返却されます。そのため、ファクトリーリセット実行の直前に登録した不揮発情報が消去されたことを確認し、ファクトリーリセットが成功したことを確認する仕様となっています。
⑨ PSM 設定状態を確認
リセット前に PSM が有効化されていたが、リセット後に PSM が無効化された場合の処理
「AT+CPSMS?」コマンドを実行
⑩ ⑨で取得したステータスが PSM 無効状態であった場合、RING ライン通知や IRQ の無効化などの PSM 機能に関連する設定を実行 (223 行目：cellular_psm_config() 関数を実行)
「AT+SQNRICFG=%s,3,%s」コマンドを実行
「AT+SQNIPSCFG=%s,%s」コマンドを実行
⑪ AT コマンド用セマフォ開放
⑫ スレッドセーフ無効化

5.42.1 流用可能な処理

①～⑤、⑦、⑨、⑪および⑫の処理は、Cellular モジュールに関係なく流用可能です。

5.42.2 置き換えが必要な処理

⑥のハードウェアリセット実行用 private 関数内では、RYZ014A Cellular モジュール専用のコマンドが実行されるため、別モジュールでは使用不可となります。cellular_module_reset.c の 115 行目を削除するか、別の処理に置き換えてください。

atc_sqnautoconnect_chek() 関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1) に自動で設定されるか否かを確認します。

```
115             ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 100　cellular_module_reset.c の 115 行目

⑤の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。r_cellular_factoryreset.c の 139 行目を適切な処理に置き換えてください。

```
139                 atc_sqnsfactoryreset(p_ctrl);
```

図 101　r_cellular_factoryreset.c の 139 行目

⑩の AT コマンドは、RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。cellular_psm_config.c の 64 行目、68 行目、74 行目、116 行目および 120 行目を適切な処理に置き換えてください。

```
64          ret = atc_sqnricfg(p_ctrl, CELLULAR_SQNRICFG_MODE);
116         ret = atc_sqnricfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
```

図 102　cellular_psm_config.c の 64 行目、116 行目

```
68              ret         = atc_sqnipscfg(p_ctrl, CELLULAR_SQNIPSCFG_MODE);
120             ret = atc_sqnipscfg(p_ctrl, (uint8_t)CELLULAR_PSM_MODE_INVALID);
```

図 103　cellular_psm_config.c の 68 行目、120 行目

```
74              ret         = atc_sqnpscfg(p_ctrl);
```

図 104　cellular_psm_config.c の 74 行目

5.43 R_CELLULAR_RTS_Ctrl()

R_CELLULAR_RTS_Ctrl()の処理の流れは、以下の通りとなっています。

① 引数確認およびスレッドセーフ有効化
② RTS 端子の出力制御を実行
③ スレッドセーフ無効化

5.43.1 流用可能な処理

①～③の処理は、Cellular モジュールに関係なく流用可能です。

6. 各ソフトウェアの API

RYZ014A Cellular FIT モジュールを、RYZ014A Cellular モジュール以外の通信モジュール制御に使用する場合やサポート対象外の RX マイコンを使用する場合に必要となる対応について、関数毎の詳細を付録に記載します。以下のソフトウェア処理に関する情報は、7 章を参照してください。

1. RYZ014A Cellular FIT モジュール内部関数
2. r_bsp
3. r_sci_rx
4. r_irq_rx
5. FreeRTOS

7. 付録

7.1 動作確認環境

RYZ014A Cellular FIT モジュールの動作確認環境を表 7.1 に示します。

表 7.1 動作確認環境

項目		内容
統合開発環境		ルネサスエレクトロニクス製 e^2 studio Ver.2023-07
コンパイラ	CC-RX	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.05.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加-lang = c99
	GCC	-
エンディアン		リトルエンディアン
RYZ014A Cellular FIT モジュールのリビジョン		Rev1.11
使用ボード(RX)		Renesas CK-RX65N (型名：RTK5CK65N0SxxxxxBE)
		Renesas RX72N Envision Kit (型名：RTK5RX72N0C00000BJ)
使用ボード(RYZ014A)		PMOD Expansion Board for RYZ014A(型名：RTKYZ014A0B00000BE)
RTOS	FreeRTOS	10.4.3-rx-1.0.1
FIT	BSP FIT	Ver 7.30
	SCI FIT	Ver 4.80
	IRQ FIT	Ver 4.30

7.2 トラブルシューティング

(1) Q：RYZ014A Cellular FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e^2 studio を使用している場合
  アプリケーションノート RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、RYZ014A Cellular FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：RYZ014A Cellular FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「コンフィグ設定が間違っている場合のエラーメッセージ」エラーが発生します。

A：“r_aws_cellular_config.h”ファイルの設定値が間違っている可能性があります。
“r_aws_cellular_config.h”ファイルを確認して正しい値を設定してください。詳細は「[6]の 2.7 節」を参照してください。

7.3 復帰処理

RYZ014A Cellular FIT モジュールを使用時に本節に示す事象が発生した場合、ユーザは復帰処理を実行してください。

7.3.1 Cellular モジュールが^EXIT URC を通知した場合

Cellular モジュールは、^EXIT URC を通知して自己リセットする場合があります。ユーザは、^EXIT URC をコールバック関数で検出することを推奨します。^EXIT URC の通知を検出した場合、ユーザは以下の処理を実行してください。

(1) +SYSSTART URC を検出。
　^EXIT URC 後の+SYSSTART URC をコールバック関数で検出します。
(2) R_CELLULAR_Close() を実行。
(3) R_CELLULAR_Open() を実行。

以上で、アクセスポイントへ接続可能な状態 (関連ドキュメント[6]の図 1.3 の「Cellular モジュール・FIT モジュールの初期化完了」状態) へ復帰します。

7.3.2 Cellular モジュールが+SYSSTART URC を通知した場合

Cellular モジュールは、起動完了後に+SYSSTART URC を通知します。+SYSSTART URC は、R_CELLULAR_Open()を含む以下の API 実行中の再起動に伴って通知されます。

- R_CELLULAR_Open()
- R_CELLULAR_SetPSM()
- R_CELLULAR_SetOperator()
- R_CELLULAR_SetBand()
- R_CELLULAR_FirmUpgradeBlocking()
- R_CELLULAR_SoftwareReset()
- R_CELLULAR_HardwareReset()
- R_CELLULAR_FactoryReset()

Cellular モジュールが予期せず再起動した場合は、上記 API の実行中以外に+SYSSTART URC が通知されます。ユーザは、+SYSSTART URC をコールバック関数で検出することを推奨します。Cellular モジュールで予期しない再起動が発生した場合は、ユーザは以下の処理を実行してください。

(1) R_CELLULAR_Close() を実行。
(2) R_CELLULAR_Open() を実行。

以上で、アクセスポイントへ接続可能な状態 (関連ドキュメント[6]の図 1.3 の「Cellular モジュール・FIT モジュールの初期化完了」状態) へ復帰します。

7.3.3 API でタイムアウトが発生した場合

API でタイムアウトが発生した場合、戻り値で CELLULAR_ERR_MODULE_TIMEOUT を返してユーザへ通知します。その場合は、ユーザは以下の処理を実行してください。

(1) R_CELLULAR_HardwareReset() を実行。
　RYZ014A Cellular モジュールの RESETN 端子でリセットを実行します。
(2) R_CELLULAR_Close() を実行。
(3) R_CELLULAR_Open() を実行。

以上で、アクセスポイントへ接続可能な状態 ([6]の図 1.3 の「Cellular モジュール・FIT モジュールの初期化完了」状態) へ復帰します。

7.4 RYZ014A Cellular FIT モジュール内部関数の変更箇所

7.4.1 cellular_apconnect_config()

cellular_apconnect_config()は r_cellular_apconnect.c 内の static 関数で、アクセスポイント接続のコンフィグ設定を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.2 cellular_apconnect()

cellular_apconnect()は r_cellular_apconnect.c 内の static 関数で、アクセスポイントへの接続処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.3 cellular_sync_check()

cellular_sync_check()は r_cellular_apconnect.c 内の static 関数で、アクセスポイント接続完了後の情報取得 (PDP アドレス、ネットワークタイム) を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.4 cellular_socket_cfg()

cellular_socket_cfg()は r_cellular_createsocket.c 内の static 関数で、ソケットのコンフィグ設定を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_socket_cfg()の 154、157 行目を適切な処理に置き換えてください。

```
154     atc_ret = atc_sqnscfg(p_ctrl, (uint8_t)(socket_num + start_num));
```

図 105　cellular_socket_cfg()の 154 行目

```
157         atc_ret = atc_sqnscfgext(p_ctrl, (uint8_t)(socket_num + start_num));
```

図 106　cellular_socket_cfg()の 157 行目

7.4.5 cellular_getip()

cellular_getip()は r_cellular_dnsquery.c 内の static 関数で、取得した IP アドレスを第 3 引数に格納します。本関数は Cellular モジュールの IP アドレスの通知方法に合わせて、処理の修正が必要になります。

7.4.6 cellular_factoryreset()

cellular_factoryreset()は r_cellular_factoryreset.c 内の static 関数で、ファクトリーリセットを実行します。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_factoryreset()の 139 行目を適切な処理に置き換えてください。

```
139                 atc_sqnsfactoryreset(p_ctrl);
```

図 107　cellular_factoryreset()の 139 行目

7.4.7 cellular_psm_check()

cellular_psm_check()は r_cellular_dnsquery.c 内の static 関数で、PSM 設定状態の取得および PSM 設定の無効化を行います。本関数は Cellular モジュールに関係なく流用可能です。

※関数内で使用されている cellular_psm_config()については、修正が必要となります。

7.4.8 private_cgdcont()

private_cgdcont()は r_cellular_dnsquery.c 内の static 関数で、PDP コンテキストへダミーデータを登録します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.9 cellular_firmupgradeblocking()

cellular_firmupgradeblocking()は r_cellular_firmupgradeblocking.c 内の static 関数で、FOTA をブロッキングで実行します。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_firmupgradeblocking()の 141 行目を適切な処理に置き換えてください。

```
141     ret                 = atc_sqnsupgrade(p_ctrl, p_url, 1, CELLULAR_FIRM_UPGRADE_BLOCKING, spid);
```

図 108　cellular_firmupgradeblocking()の 141 行目

7.4.10 cellular_init()

cellular_init()は r_cellular_open.c 内の static 関数で、Cellular モジュールとの通信設定を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_init()の 289 行目を適切な処理に置き換え、または削除してください。

```
289         ret = atc_sqnsimst(p_ctrl);
```

図 109　cellular_init()の 289 行目

7.4.11 cellular_config_init()

cellular_config_init()は r_cellular_open.c 内の static 関数で、各種コンフィグ設定を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.12 cellular_open_fail()

cellular_open_fail()は r_cellular_open.c 内の static 関数で、R_CELLULAR_Open()が失敗した場合の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.13 cellular_receive_data()

cellular_receive_data()は r_cellular_receivesocket.c 内の static 関数で、ソケット通信のデータ受信処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。処理を修正する場合は、5.11.3 を参考にしてください。

7.4.14 cellular_receive_flag_check()

cellular_receive_flag_check()は r_cellular_receivesocket.c 内の static 関数で、受信可能なデータがあるかを確認する処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンド処理に関連した処理を行っている為、別モジュールでは使用不可となります。処理を修正する場合は、5.11.3 を参考にしてください。

7.4.15 cellular_recv_size_check()

cellular_recv_size_check()は r_cellular_receivesocket.c 内の static 関数で、Cellular モジュールへ要求する受信データサイズの決定処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンド処理に関連した処理を行っている為、別モジュールでは使用不可となります。処理を修正する場合は、5.11.3 を参考にしてください。※または 332~336 行目を削除してください。

7.4.16 cellular_send_data()

cellular_send_data()は r_cellular_sendsocket.c 内の static 関数で、ソケット通信のデータ送信処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。処理を修正する場合は、4.6 を参考にしてください。

7.4.17 cellular_send_size_check()

cellular_send_size_check()は r_cellular_sendsocket.c 内の static 関数で、Cellular モジュールへ送信する実際のデータサイズを決定します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.18 cellular_tx_flag_check()

cellular_tx_flag_check()は r_cellular_sendsocket.c 内の static 関数で、R_SCI_Send()関数によって Cellular モジュールへのデータ送信が完了したことを確認します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.19 cellular_atc_response_check()

cellular_atc_response_check()は r_cellular_sendsocket.c 内の static 関数で、AT コマンドを送信後に Cellular モジュールから Intermediate result code を受信し、その後 Cellular モジュールへ追加でデータを送信し、それに対するレスポンス文字列が期待値であることを確認します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.20 cellular_set_operator()

cellular_set_operator()は r_cellular_setoperator.c 内の static 関数で、オペレータ設定を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_set_operator()の 127 行目、133 行目を適切な処理に置き換えてください。

```
127     ret                 = atc_sqnctm_check(p_ctrl);
133         ret                 = atc_sqnctm(p_ctrl, p_operator);
```

図 110　「r_cellular_setoperator.c」の 127 行目、133 行目

7.4.21 cellular_softwarereset()

cellular_softwarereset()は r_cellular_softwarereset.c 内の static 関数で、ソフトリセットを実行します。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。cellular_softwarereset()の 133 行目を適切な処理に置き換え、または削除してください。

※atc_sqnautoconnect_chek()関数では、「AT+SQNAUTOCONNECT?」コマンドを実行し、Cellular モジュール起動時にファンクションレベルが 1 (CFUN=1) に自動で設定されるか否かを確認します。

```
133     ret                 = atc_sqnautoconnect_check(p_ctrl);
```

図 111　cellular_softwarereset()の 133 行目

7.4.22 cellular_unlocksim()

cellular_unlocksim()は r_cellular_unlocksim.c 内の static 関数で、SIM の PIN ロック解除コマンドを発行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.23 cellular_write_certificate()

cellular_write_certificate()は r_cellular_writecertficate.c 内の static 関数で、Cellular モジュールの不揮発メモリへ証明書/秘密鍵情報を書き込むためのコマンドを発行します。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。処理を修正する場合は、4.6 を参考にしてください。

7.4.24 cellular_send_size_check()

cellular_send_size_check()は r_cellular_unlocksim.c 内の static 関数で、R_SCI_Send()関数で Cellular モジュールへ送信するデータサイズの確認を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.25 atc_generate()

atc_generate()は at_command.c 内で定義されている関数で、Cellular モジュールへ送信する AT コマンドを生成します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.26 atc_ate0()

atc_ate0()は ate0.c 内で定義されている関数で、Cellular モジュールへ「ATE0」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.27 atc_ati1()

atc_ati1()は ati0.c 内で定義されている関数で、Cellular モジュールへ「ATI1」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.28 atc_cclk()

atc_cclk()は cclk.c 内で定義されている関数で、Cellular モジュールへ「AT+CCLK="%s/%s/%s,%s:%s:%s%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.29 atc_cclk_check()

atc_cclk_check()は cclk.c 内で定義されている関数で、Cellular モジュールへ「AT+CCLK?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.30 atc_ceer()

atc_ceer()は ceer.c 内で定義されている関数で、Cellular モジュールへ「AT+CEER」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.31 atc_cereg()

atc_cereg()は cereg.c 内で定義されている関数で、Cellular モジュールへ「AT+CEREG=%s」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.32 atc_cereg_check()

atc_cereg_check()は cereg.c 内で定義されている関数で、Cellular モジュールへ「AT+CEREG?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.33 atc_cfun()

atc_cfun()は cfun.c 内で定義されている関数で、Cellular モジュールへ「AT+CFUN=%s」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.34 atc_cfun_check()

atc_cfun_check()は cfun.c 内で定義されている関数で、Cellular モジュールへ「AT+CFUN?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.35 atc_cgact()

atc_cgact()は cgact.c 内で定義されている関数で、Cellular モジュールへ「AT+CGACT=1,%s」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.36 atc_cgact_check()

atc_cgact_check()は cgact.c 内で定義されている関数で、Cellular モジュールへ「AT+CGACT?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.37 atc_cgatt()

atc_cgatt()は cgatt.c 内で定義されている関数で、Cellular モジュールへ「AT+CGATT=%s」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.38 atc_cgatt_check()

atc_cgatt_check()は cgatt.c 内で定義されている関数で、Cellular モジュールへ「AT+CGATT?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.39 atc_cgauth()

atc_cgauth()は cgauth.c 内で定義されている関数で、Cellular モジュールへ「AT+CGAUTH=1,%s,"%s","%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.40 atc_cgauth_reset()

atc_cgauth_reset()は cgauth.c 内で定義されている関数で、Cellular モジュールへ「AT+CGAUTH=1,0」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.41 atc_cgdcont()

atc_cgdcont()は cgdcont.c 内で定義されている関数で、Cellular モジュールへ「AT+CGDCONT=%s,"IPV4V6","%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.42 atc_cgdcont_check()

atc_cgdcont_check()は cgdcont.c 内で定義されている関数で、Cellular モジュールへ「AT+CGDCONT?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.43 atc_cgmi()

atc_cgmi()は cgmi.c 内で定義されている関数で、Cellular モジュールへ「AT+CGMI」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.44 atc_cgmm()

atc_cgmm()は cgmm.c 内で定義されている関数で、Cellular モジュールへ「AT+CGMM」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.45 atc_cgmr()

atc_cgmr()は cgmr.c 内で定義されている関数で、Cellular モジュールへ「AT+CGMR」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.46 atc_cgpaddr()

atc_cgpaddr()は cgpaddr.c 内で定義されている関数で、Cellular モジュールへ「AT+CGPADDR=1」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.47 atc_cgpiaf()

atc_cgpiaf()は cgpiaf.c 内で定義されている関数で、Cellular モジュールへ「AT+CGPIAF=1,0,1,0」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.48 atc_cgsn()

atc_cgsn()は cgsn.c 内で定義されている関数で、Cellular モジュールへ「AT+CGSN」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.49 atc_cgsn3()

atc_cgsn3()は cgsn.c 内で定義されている関数で、Cellular モジュールへ「AT+CGSN=3」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.50 atc_cimi()

atc_cimi()は cimi.c 内で定義されている関数で、Cellular モジュールへ「AT+CIMI」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.51 atc_cmer()

atc_cmer()は cmer.c 内で定義されている関数で、Cellular モジュールへ「AT+CMER=3,0,0,%s,0,0,0」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.52 atc_cnum()

atc_cnum()は cnum.c 内で定義されている関数で、Cellular モジュールへ「AT+CNUM」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.53 atc_cops()

atc_cops()は cops.c 内で定義されている関数で、Cellular モジュールへ「AT+COPS=%s,2,"%s%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.54 atc_cops_check()

atc_cops_check()は cops.c 内で定義されている関数で、Cellular モジュールへ「AT+COPS?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.55 atc_cpin()

atc_cpin()は cpin.c 内で定義されている関数で、Cellular モジュールへ「AT+CPIN="%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.56 atc_cpin_check()

atc_cpin_check()は cpin.c 内で定義されている関数で、Cellular モジュールへ「AT+CPIN?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.57 atc_cpsms()

atc_cpsms()は cpsms.c 内で定義されている関数で、Cellular モジュールへ「AT+CPSMS=%s,,,"%s","%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.58 atc_cpsms_check()

atc_cpsms_check()は cops.c 内で定義されている関数で、Cellular モジュールへ「AT+CPSMS?」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.59 atc_crsm()

atc_crsm()は crsm.c 内で定義されている関数で、Cellular モジュールへ「AT+CRSM=%s,%s,%s,%s,%s,"%s","%s"」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.60 atc_csq()

atc_csq()は csq.c 内で定義されている関数で、Cellular モジュールへ「AT+CSQ」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.61 atc_ping()

atc_ping()は ping.c 内で定義されている関数で、Cellular モジュールへ「AT+PING="%s",%s,%s,%s,%s」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.62 atc_reset()

atc_reset()は reset.c 内で定義されている関数で、Cellular モジュールへ「AT^RESET」コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.63 atc_smcwrx()

atc_smcwrx()は smcwrx.c 内で定義されている関数で、Cellular モジュールへ「AT+SMCWRX=%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.64 atc_smcwtx()

atc_smcwtx()は smcwtx.c 内で定義されている関数で、Cellular モジュールへ「AT+SMCWTX=%s,%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.65 atc_sqnautoconnect()

atc_sqnautoconnect()は sqnautoconnect.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNAUTOCONNECT=%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.66 atc_sqnautoconnect_check()

atc_sqnautoconnect_check()は sqnautoconnect.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNAUTOCONNECT?」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.67 atc_sqnbandsel()

atc_sqnbandsel()は sqnbandsel.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNBANDSEL=0,"%s","%s"」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.68 atc_sqnccid()

atc_sqnccid()は sqnccid.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNCCID?」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.69 atc_sqnctm()

atc_sqnctm()は sqnctm.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNCTM="%s"」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.70 atc_sqnctm_check()

atc_sqnctm_check()は sqnctm.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNCTM?」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.71 atc_sqndnslkup()

atc_sqndnslkup()は sqndnslkup.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNDNSLKUP="%s",%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.72 atc_sqnedrx()

atc_sqnedrx()は sqnedrx.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNEDRX=%s,4,"%s","%s"」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.73 atc_sqnedrx_check()

atc_sqnedrx_check()は sqnedrx.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNEDRX?」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.74 atc_sqnipscfg()

atc_sqnipscfg()は sqnipscfg.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNIPSCFG=%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.75 atc_sqnmoni()

atc_sqnmoni()は sqnmoni.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNMONI=%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.76 atc_sqnpscfg()

atc_sqnpscfg()は sqnpscfg.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNPSCFG=%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.77 atc_sqnricfg()

atc_sqnricfg()は sqnricfg.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNRICFG=%s,3,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.78 atc_sqnscfg()

atc_sqnscfg()は sqnscfg.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSCFG=%s,1,%s,%s,%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.79 atc_sqnscfgext()

atc_sqnscfgext()は sqnscfgext.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSCFGEXT=%s,1,0,0」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.80 atc_sqnsd()

atc_sqnsd()は sqnsd.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSD=%s,%s,%s,"%s",0,%s,1」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.81 atc_sqnsfactoryreset()

atc_sqnsfactoryreset()は sqnsfactoryreset.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSFACTORYRESET」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.82 atc_sqnsh()

atc_sqnsh()は sqnsh.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSH=%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.83 atc_sqnsimst()

atc_sqnsimst()は sqnsimst.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSIMST=0」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.84 atc_sqnsl()

atc_sqnsl()は sqnsl.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSL=%s,%s,%s,0」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.85 atc_sqnsnvr()

atc_sqnsnvr()は sqnsnvr.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSNVR="%s",%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.86 atc_sqnsnvw()

atc_sqnsnvw()は sqnsnvw.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSNVW="%s",%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.87 atc_sqnsnvw_erase()

atc_sqnsnvw_erase()は sqnsnvw.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSNVW=¥"%s¥",%s,0」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.88 atc_sqnspcfg()

atc_sqnspcfg()は sqnspcfg.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSPCFG=%s,2,,%s,%s,%s,%s,""」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.89 atc_sqnsrecv()

atc_sqnsrecv()は sqnsrecv.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSRECV=%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.90 atc_sqnssendext()

atc_sqnssendext()は sqnssendext.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSSENDEXT=%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.91 atc_sqnsshdn()

atc_sqnsshdn()は sqnsshdn.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSSHDN」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.92 atc_sqnsupgrade()

atc_sqnsupgrade()は sqnsupgrade.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSUPGRADE="%s",%s,5,%s,%s」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.93 atc_sqnsupgrade_check()

atc_sqnsupgrade_check()は sqnsupgrade.c 内で定義されている関数で、Cellular モジュールへ「AT+SQNSUPGRADE?」コマンドを送信します。このコマンドは RYZ014A Cellular モジュール専用のコマンドであるため、別モジュールでは使用不可となります。

7.4.94 cellular_set_atc_number()

cellular_set_atc_number()は cellular_at_cmd_res_ctrl.c 内で定義されている関数で、AT コマンドを Cellular モジュールへ送信した場合の、レスポンスの受信準備を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.95 cellular_get_atc_response()

cellular_get_atc_response()は cellular_at_cmd_res_ctrl.c 内で定義されている関数で、Cellular モジュールからのレスポンス内容確認を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.96 cellular_closesocket()

cellular_closesocket()は cellular_closesocket.c 内で定義されている関数で、ソケット管理構造体の初期化を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.97 cellular_disconnect()

cellular_disconnect()は cellular_disconnect.c 内で定義されている関数で、アクセスポイントからの切断処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.98 cellular_execute_at_command()

cellular_execute_at_command()は cellular_execute_at_command.c 内で定義されている関数で、Cellular モジュールへ実際に AT コマンドを送信します。本関数は Cellular モジュールに関係なく流用可能です。

※同ファイル内の cellular_send_atc()および cellular_res_check()も同様に流用可能です。

7.4.99 cellular_getpdpaddr()

cellular_getpdpaddr()は cellular_getpdpaddr.c 内で定義されている関数で、取得した PDP アドレスを第 2 引数へ格納します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.100 cellular_irq_open()

cellular_irq_open()は cellular_irq_ctrl.c 内で定義されている関数で、R_IRQ_Open()を実行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.101 cellular_irq_close()

cellular_irq_close()は cellular_irq_ctrl.c 内で定義されている関数で、R_IRQ_Close()を実行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.102 cellular_ring_callbResponse()

cellular_ring_callbResponse()は cellular_irq_ctrl.c 内で定義されているコールバック関数で、R_IRQ_Open()実行時に登録します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.103 cellular_ring_task()

cellular_ring_task()は cellular_irq_ctrl.c 内で定義されているタスク関数で、RING ラインの監視を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.104 cellular_module_reset()

cellular_module_reset()は cellular_module_reset.c 内で定義されている関数で、ハードウェアリセットを行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

62 行目：cellular_shutdownsocket()の内容を修正

115 行目：atc_sqnautoconnect_check()をモジュールに適した AT コマンド実行関数に置き換え、
または削除

7.4.105 cellular_pin_reset()

cellular_pin_reset()は cellular_module_reset.c 内で定義されている static 関数で、ハードウェアリセットを実行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.106 cellular_power_down()

cellular_power_down()は cellular_power_down.c 内で定義されている関数で、ハードウェアリセットを行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

62 行目：atc_sqnsshdn()をモジュールに適した AT コマンド実行関数に置き換え

7.4.107 cellular_psm_config()

cellular_psm_config()は cellular_psm_config.c 内で定義されている関数で、PSM コンフィグを行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

64・116：atc_sqnricfg()をモジュールに適した AT コマンド実行関数に置き換え

68・120：atc_sqnipscfg()をモジュールに適した AT コマンド実行関数に置き換え

74 行目：atc_sqnpscfg()をモジュールに適した AT コマンド実行関数に置き換え

7.4.108 cellular_psm_config_fail()

cellular_psm_config_fail()は cellular_psm_config.c 内で定義されている static 関数で、PSM コンフィグが失敗した場合の処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

188 行目：atc_sqnipscfg()をモジュールに適した AT コマンド実行関数に置き換え

193 行目：atc_sqnricfg()をモジュールに適した AT コマンド実行関数に置き換え

7.4.109 cellular_rts_ctrl()

cellular_rts_ctrl()は cellular_rts_ctrl.c 内で定義されている関数で、RTS 端子を制御します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.110 cellular_rts_hw_flow_enable()

cellular_rts_hw_flow_enable()は cellular_rts_ctrl.c 内で定義されている関数で、RTS 端子の HW フロー制御を有効化します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.111 cellular_rts_hw_flow_disable()

cellular_rts_hw_flow_disable()は cellular_rts_ctrl.c 内で定義されている関数で、RTS 端子の HW フロー制御を無効化します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.112 cellular_serial_open()

cellular_serial_open()は cellular_sci_ctrl.c 内で定義されている関数で、R_SCI_Open()を実行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.113 cellular_serial_close()

cellular_serial_close()は cellular_sci_ctrl.c 内で定義されている関数で、R_SCI_Close()を実行します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.114 cellular_uart_callbResponse()

cellular_uart_callbResponse()は cellular_sci_ctrl.c 内で定義されているコールバック関数で、R_SCI_Open()実行時に登録します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.115 cellular_semaphore_init()

cellular_semaphore_init()は cellular_semaphore_ctrl.c 内で定義されている関数で、RYZ014A Cellular FIT モジュール内で使用されるセマフォの初期化を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.116 cellular_shutdownsocket()

cellular_shutdownsocket()は cellular_shutdownsocket.c 内で定義されている関数で、ソケット切断処理を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

76 行目：atc_sqnsh()をモジュールに適した AT コマンド実行関数に置き換え

7.4.117 cellular_smcwrx()

cellular_smcwrx()は cellular_smcwrx.c 内で定義されている関数で、電波受信強度を測定します。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

62 行目：atc_smcwrx()をモジュールに適した AT コマンド実行関数に置き換え

7.4.118 cellular_smcwtx()

cellular_smcwtx()は cellular_smcwtx.c 内で定義されている関数で、テスト送信を行います。本関数は RYZ014A Cellular モジュール専用のコマンドを使用しているため、別モジュールでは使用不可となります。以下の処理を適切な処理に置き換えてください。

63 行目：atc_smcwtx()をモジュールに適した AT コマンド実行関数に置き換え

7.4.119 cellular_start_recv_task()

cellular_start_recv_task()は cellular_task_ctrl.c 内で定義されている関数で、RYZ014A Cellular FIT モジュール内で使用されるデータ受信タスクの生成処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.120 cellular_start_ring_task()

cellular_start_ring_task()は cellular_task_ctrl.c 内で定義されている関数で、RYZ014A Cellular FIT モジュール内で使用される RING 端子制御タスクの生成処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.121 cellular_timeout_init()

cellular_timeout_init()は cellular_timeout_ctrl.c 内で定義されている関数で、タイムアウト時間設定処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.122 cellular_check_timeout()

cellular_check_timeout()は cellular_timeout_ctrl.c 内で定義されている関数で、タイムアウト処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.123 cellular_job_check()

cellular_job_check()は r_cellular_receive_task.c 内で定義されている static 関数で、レスポンス文字列タイプの判定を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.124 cellular_response_string_check()

cellular_response_string_check()は r_cellular_receive_task.c 内で定義されている static 関数で、レスポンス文字列の判定を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.125 cellular_response_check()

cellular_response_check()は r_cellular_receive_task.c 内で定義されている static 関数で、result code の判定を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.126 cellular_data_send_command()

cellular_data_send_command()は r_cellular_receive_task.c 内で定義されている static 関数で、Intermediate result code「>」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能ですが、Intermediate result code はモジュールに合わせて修正してください。詳細は 4.6 をご確認ください。

7.4.127 cellular_get_data_reception()

cellular_get_data_reception()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSRING」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.128 cellular_request_data()

cellular_request_data()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSRECV」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.129 cellular_store_data()

cellular_store_data()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSRECV」受信後に送られてくるデータの受信処理を行います。使用する Cellular モジュールのソケットデータ受信コマンドの仕様にあわせて、適切な処理に修正してください。

7.4.130 cellular_get_data_reception()

cellular_get_data_reception()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNDNSLKUP」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.131 cellular_get_ap_connect_status()

cellular_get_ap_connect_status()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CGATT」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.132 cellular_get_ap_connect_config()

cellular_get_ap_connect_config()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CGDCONT」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.133 cellular_station_info()

cellular_station_info()は r_cellular_receive_task.c 内で定義されている static 関数で、「+ CREG」または「+CEREG」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.134 cellular_control_level()

cellular_control_level()は r_cellular_receive_task.c 内で定義されている static 関数で、「+ CFUN」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.135 cellular_cpin_status()

cellular_cpin_status()は r_cellular_receive_task.c 内で定義されている static 関数で、「+ CPIN」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.136 cellular_get_time()

cellular_get_time()は r_cellular_receive_task.c 内で定義されている static 関数で、「+ CCLK」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.137 cellular_get_imei()

cellular_get_imei()は r_cellular_receive_task.c 内で定義されている static 関数で、「AT+CGSN」コマンド実行に対するレスポンス文字列の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.138 cellular_get_imsi()

cellular_get_imsi()は r_cellular_receive_task.c 内で定義されている static 関数で、「AT+CIMI」コマンド実行に対するレスポンス文字列の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.139 cellular_system_start()

cellular_system_start()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SYSSTART」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.140 cellular_disconnect_socket()

cellular_disconnect_socket ()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSH」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.141 cellular_get_timezone()

cellular_get_timezone()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CTZE」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.142 cellular_get_service_status()

cellular_get_service_status()は r_cellular_receive_task.c 内で定義されている static 関数で、「+COPS」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.143 cellular_get_service_status()

cellular_get_service_status()は r_cellular_receive_task.c 内で定義されている static 関数で、「+COPS」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.144 cellular_get_pdp_status()

cellular_get_pdp_status()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CGACT」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.145 cellular_get_pdp_addr()

cellular_get_pdp_addr()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CGPADDR」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.146 cellular_get_psms()

cellular_get_psms()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CPSMS」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.147 cellular_get_edrx()

cellular_get_edrx()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNEDRX」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.148 cellular_get_signal()

cellular_get_signal()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CSQ」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.149 cellular_res_command_send_sim()

cellular_res_command_send_sim()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CRSM」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.150 cellular_timezone_info()

cellular_timezone_info()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CTZV」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.151 cellular_ind_info()

cellular_ind_info()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CIEV」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.152 cellular_get_svn()

cellular_get_svn()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CGSN」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.153 cellular_get_lrsvn()

cellular_get_lrsvn()は r_cellular_receive_task.c 内で定義されている static 関数で、「ATI1」コマンド実行に対するレスポンス文字列の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.154 cellular_get_phone_number()

cellular_get_phone_number()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CNUM」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.155 cellular_get_iccid()

cellular_get_iccid()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNCCID」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.156 cellular_ping()

cellular_ping()は r_cellular_receive_task.c 内で定義されている static 関数で、「+PING」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.157 cellular_get_cellinfo()

cellular_get_cellinfo()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNMONI」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.158 cellular_get_autoconnect()

cellular_get_autoconnect()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNAUTOCONNECT」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.159 cellular_get_ctm()

cellular_get_ctm()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNCTM」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.160 cellular_set_smcwrx()

cellular_set_smcwrx()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SMCWRX」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.161 cellular_set_smcwtx()

cellular_set_smcwtx()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SMCWTX」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.162 cellular_shutdown_info()

cellular_shutdown_info()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SHUTDOWN」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.163 cellular_firmupgrade_info()

cellular_firmupgrade_info()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSUPGRADE」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.164 cellular_get_certificate()

cellular_get_certificate()は r_cellular_receive_task.c 内で定義されている static 関数で、「+SQNSNVR」受信時の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.165 cellular_get_revision()

cellular_get_revision()は r_cellular_receive_task.c 内で定義されている static 関数で、「AT+CGMR」コマンド実行に対するレスポンス文字列の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.166 cellular_response_skip()

cellular_response_skip()は r_cellular_receive_task.c 内で定義されている static 関数で、未登録のレスポンス文字列を受信した場合の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.167 cellular_memclear()

cellular_memclear()は r_cellular_receive_task.c 内で定義されている static 関数で、データ受信バッファの初期化処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.168 cellular_exit()

cellular_exit()は r_cellular_receive_task.c 内で定義されている static 関数で、「^EXIT」受信時の処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.169 cellular_system_state_change()

cellular_system_state_change()は r_cellular_receive_task.c 内で定義されている static 関数で、「+CREG」または「+CEREG」を受信した際にアクセスポイント接続ステータスを更新する処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.170 cellular_get_at_command()

cellular_get_at_command()は r_cellular_receive_task.c 内で定義されている static 関数で、実行中の AT コマンドを取得処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.171 cellular_set_atc_response()

cellular_set_atc_response()は r_cellular_receive_task.c 内で定義されている static 関数で、AT コマンドの実行結果格納処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.172 cellular_cleardata()

cellular_cleardata()は r_cellular_receive_task.c 内で定義されている static 関数で、レスポンス文字列をユーザが登録したコールバック関数への通知処理、ログ出力処理、受信バッファクリア処理を行います。本関数は Cellular モジュールに関係なく流用可能です。

7.4.173 cellular_charcheck()

cellular_charcheck()は r_cellular_receive_task.c 内で定義されている static 関数で、「AT+SQNSUPGRADE」コマンドをブロッキングモードで実行した際の処理を行います。RYZ014A Cellular モジュール専用のコマンドに対するレスポンスであるため、別モジュールでは使用不可となります。

7.4.174 binary_conversion()

binary_conversion()は r_cellular_receive_task.c 内で定義されている static 関数で、2 進数を 10 進数に変換します。本関数は Cellular モジュールに関係なく流用可能です。

7.4.175 RTOS フォルダ内の関数

RTOS フォルダ内に配置されているソースファイルに定義されている全ての関数は、Cellular モジュールに関係なく流用可能です。

7.5 r_bsp の変更箇所

7.5.1 R_BSP_NOP()

R_BSP_NOP()は nop 命令として使用しています。使用するマイコンに応じて、適切な処置に置き換えてください。

R_BSP_NOP()を使用している箇所は、以下の 2 種類が存在します。

- 送信処理時間軽減のために記述されている箇所
- コーディングルールに基づいて記述されている箇所

7.5.1.1 送信処理時間軽減のために記述されている箇所

RYZ014A Cellular FIT モジュールでは、CTS/RTS の SW/HW 制御切り替えが可能です。CTS 端子を SW 制御、RTS 端子を HW 制御とした場合は、Cellular モジュールに対して 1byte ずつデータ送信が行われます。この場合、CTS 端子を HW 制御、RTS 端子を SW 制御とした場合と同じタイムアウト処理 (vTaskDelay()を用いる方法) では 1byte 送信毎に 1ms (最小単位) のディレイが発生してしまい、データ送信処理に時間がかかりすぎてしまうため、R_BSP_NOP() を使用しています。

送信処理時間軽減のために R_BSP_NOP() が記述されている箇所は以下の通りです。

- r_cellular_sendsocket.c … 421 行目
- r_cellular_writecertificate.c … 396 行目

7.5.1.2 コーディングルールに基づいて記述されている箇所

7.5.1.1 節以外で使用されている R_BSP_NOP()は、すべてコーディングルールに基づいて記述しています。

7.5.2 R_BSP_RegisterProtectDisable()

R_BSP_RegisterProtectDisable()はレジスタ書き込み保護機能の無効化を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_BSP_RegisterProtectDisable()が使用されている箇所は、以下の通りです。

- cellular_rts_ctrl.c … 67 行目、85 行目

7.5.3 R_BSP_RegisterProtectEnable()

R_BSP_RegisterProtectEnable()はレジスタ書き込み保護機能の有効化を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_BSP_RegisterProtectEnable()が使用されている箇所は、以下の通りです。

- cellular_rts_ctrl.c … 72 行目、89 行目

7.6　r_sci_rx の変更箇所

7.6.1 R_SCI_Open()

R_SCI_Open()は SCI チャネルの有効化および実行時にコールバック関数を登録し、送信完了割り込みの監視を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_SCI_Open()が使用されている箇所は、以下の通りです。

- cellular_sci_ctrl.c　…　69 行目

R_SCI_Open()実行時に登録されているコールバック関数は、以下で定義されています。r_sci を使用しない場合、適切な処置に置き換えてください。

- cellular_sci_ctrl.c　…　118~165 行目 (cellular_uart_callbResponse() 関数)

7.6.2 R_SCI_Close()

R_SCI_Open()は SCI チャネル無効化を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_SCI_Close()が使用されている箇所は、以下の通りです。

- cellular_sci_ctrl.c　…　105 行目

7.6.3 R_SCI_Send()

R_SCI_Send()はデータ送信処理を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_SCI_Send()が使用されている箇所は、以下の通りです。

- r_cellular_sendsocket.c　…　217 行目、248 行目
- r_cellular_writecertificate.c　…　195 行目、230 行目
- cellular_execute_at_cmd.c　…　145 行目、178 行目

7.6.4 R_SCI_Receive()

R_SCI_Receive()はデータ受信処理を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_SCI_Receive()が使用されている箇所は、以下の通りです。

- r_cellular_receive_task.c　…　275 行目、777 行目、1617 行目 1655 行目、1848 行目、1859 行目、1909 行目、1923 行目

7.6.5 R_SCI_Control()

R_SCI_Control()は対象のチャネルに対して、CTS/HW 制御の有効化および送信プライオリティの設定を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_SCI_Control()が使用されている箇所は、以下の通りです。

- cellular_sci_ctrl.c … 79 行目 (CTS/HW 制御設定)、82 行目 (送信プライオリティ設定)

7.7 r_irq_rx の変更箇所

7.7.1 R_IRQ_Open()

R_IRQ_Open()は IRQ の有効化および実行時にコールバック関数を登録し、対象端子 (RING 端子)の監視を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

RING 端子は PSM (Power Saving Mode) 時以外は監視していない為、PSM を使用しない場合は R_IRQ_Open()の置き換えは不要となります。

R_IRQ_Open()が使用されている箇所は、以下の通りです。

- cellular_irq_ctrl.c … 60 行目

R_IRQ_Open()実行時に登録されているコールバック関数は、以下で定義されています。r_irq を使用しない場合、適切な処置に置き換えてください。

- cellular_sci_ctrl.c … 97~117 行目 (cellular_ring_callbResponse() 関数)

7.7.2 R_IRQ_Close()

R_IRQ_Open()は IRQ の無効化を行います。使用するマイコンに応じて、適切な処置に置き換えてください。

R_IRQ_Close()が使用されている箇所は、以下の通りです。

- cellular_irq_ctrl.c … 86 行目

7.8 FreeRTOS の変更箇所

7.8.1 xTaskCreate()

xTaskCreate()はタスクの生成を行います。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xTaskCreate()が使用されている箇所は、以下の通りです。

- cellular_create_task.c　…　64 行目
  ※71 行目の判定文も使用する OS に合わせて修正してください。

7.8.2 vTaskDelay()

vTaskDelay()はタスクの遅延実行を行います。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vTaskDelay()が使用されている箇所は、以下の通りです。

- cellular_delay_task.c　…　54 行目

7.8.3 xTaskGetTickCount()

xTaskGetTickCount()はシステム・クロックのカウントの取得を行います。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xTaskGetTickCount()が使用されている箇所は、以下の通りです。

- cellular_get_tickcount.c　…　52 行目

7.8.4 vTaskSuspend()

vTaskSuspend()はタスクの一時停止時を行います。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vTaskSuspend()が使用されている箇所は、以下の通りです。

- cellular_delete_task.c　…　53 行目

7.8.5 vTaskDelete()

vTaskDelete()はタスクの削除を行います。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vTaskDelete()が使用されている箇所は、以下の通りです。

- cellular_delete_task.c　…　54 行目

7.8.6 xEventGroupCreate()

xEventGroupCreate()はイベントグループ生成時に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xEventGroupCreate()が使用されている箇所は、以下の通りです。

- cellular_create_event_group.c　…　54 行目

7.8.7 xEventGroupWaitBits()

xEventGroupWaitBits()はイベントグループのイベントビット待ち時に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xEventGroupWaitBits()が使用されている箇所は、以下の通りです。

- cellular_get_event_flg.c　…　57 行目、65 行目
  ※72 行目の判定文も使用する OS に合わせて修正してください。

7.8.8 xEventGroupSetBitsFromISR()

xEventGroupSetBitsFromISR()はイベントグループのイベントビットを、割り込みルーチンから設定する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xEventGroupSetBitsFromISR()が使用されている箇所は、以下の通りです。

- cellular_set_event_flg.c　…　55 行目
  ※59 行目の判定文も使用する OS に合わせて修正してください。

7.8.9 xEventGroupSync()

xEventGroupSync()はイベントグループのイベントビットが設定されるのを待つ際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xEventGroupSync()が使用されている箇所は、以下の通りです。

- cellular_syncro_event_group.c　…　60 行目、67 行目

7.8.10 vEventGroupDelete()

vEventGroupDelete()はイベントグループを削除する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vEventGroupDelete()が使用されている箇所は、以下の通りです。

- cellular_delete_event_group.c　…　53 行目

7.8.11 xSemaphoreCreateMutex()

xSemaphoreCreateMutex()は mutex を生成する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xSemaphoreCreateMutex()が使用されている箇所は、以下の通りです。

- cellular_create_semaphore.c　…　54 行目

7.8.12 xSemaphoreTake()

xSemaphoreTake()はセマフォを取得する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xSemaphoreTake()が使用されている箇所は、以下の通りです。

- cellular_take_semaphore.c …　52 行目

7.8.13 xSemaphoreGive()

xSemaphoreGive()はセマフォを解放する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

xSemaphoreGive()が使用されている箇所は、以下の通りです。

- cellular_give_semaphore.c …　53 行目

7.8.14 vSemaphoreDelete()

vSemaphoreDelete()はセマフォを削除する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vSemaphoreDelete()が使用されている箇所は、以下の通りです。

- cellular_delete_semaphore.c　…　56 行目

7.8.15 pvPortMalloc()

pvPortMalloc()はヒープメモリを確保する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

pvPortMalloc()が使用されている箇所は、以下の通りです。

- cellular_malloc.c　…　52 行目

7.8.16 vPortFree()

vPortFree()は確保したヒープメモリを解放する際に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

vPortFree()が使用されている箇所は、以下の通りです。

- cellular_free.c　…　53 行目

7.8.17 taskENTER_CRITICAL()

taskENTER_CRITICAL()はクリティカルセクション開始時に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

taskENTER_CRITICAL()が使用されている箇所は、以下の通りです。

- cellular_interrupt_ctrl.c　…　53 行目

7.8.18 taskEXIT_CRITICAL()

taskEXIT_CRITICAL()はクリティカルセクション終了時に使用しています。FreeRTOS 以外の OS を使用する場合、適切な処置に置き換えてください。

taskEXIT_CRITICAL()が使用されている箇所は、以下の通りです。

- cellular_interrupt_ctrl.c　…　77 行目

8. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2023/12/08	-	新規作成

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。